საქართველოს ტექნიკური უნივერსიტეტი

გულნარა ჯანელიძე, თამარ მეფარიშვილი, ია აფციაური

ობიექტზე ორიენტირებული დაპროგრამება c++ ენის ბაზაზე

მეთოდური მითითებები

ლაბორატორიული სამუშაოების შესასრულებლად

(წიგნს თან ერთვის ტესტები)

თბილისი

2021

უაკ. 681.3.06

მეთოდურ სახელმძღვანელოში წარმოდგენილია ლაბორატორიული სამუშაოების შესრულების თანამიმდევრობა საგანში „ობიექტზე ორიენტირებული დაპროგრამება C++ ენის ბაზაზე“.

მეთოდური სახელმძღვანელო განკუთვნილია ინფორმატიკის სფეროს სტუდენტებისთვის და ასევე, დაპროგრამების ობიექტზე ორიენტირებული მიდგომების შესწავლის მსურველთათვის.

ტექნიკური უნივერსიტეტი (განახლებული გამოცემა, 2021 წ. ელექტრონული სახელმძღვანელოს სახით. პირველი გამოცემა 2014 წ. ბეჭდური სახით)

ISBN 978-9941-0-6426-5

შინაარსი

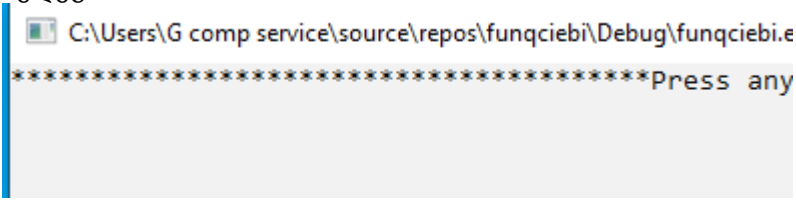
ლაბორატორიული სამუშაო 1
თემა: ფუნქციებთან მუშაობა

განვიხილოთ ამოცანები ფუნქციების გამოყენებით:

1. შექმენით ფუნქცია **void** print40star(**void**), რომელიც დაბეჭდავს 40 სიმბოლო '*' -ს.

```
#include <iostream>
using namespace std;
void print40star(void);
int main()
{
    system("color F0");
  print40star();
}
void print40star(void) {
  int i; /* simboloebis mTvleli */
  for (i = 1; i <= 40; i++)
        cout << "*";
}
```

შედეგი:



2. შექმენით ფუნქცია **void** f(**int** k); რომელიც ერთ სტრიქონად გამოიტანს ეკრანზე k ცალ შებრუნებულ სლეშს (\').

```
#include <iostream>
using namespace std;
```

```
void f(int);
int main()
{
    system("color F0");
  int k; /* striqonSi simboloTa raodenoba */
  cout << "ShemoitaneT k : " << endl;
  cin >> k;
  f(k);
}

void f(int p) {
  int i; /* simboloebis mTvleli */
  for (i = 1; i <= p; i++)
         cout << "/";
                cout << "\n";

}
```

შედეგი:

```
Microsoft Visual Studio Debug Console
hemoitaneT k :
5
/////

:\Users\G comp service\source\repos\funqciebi\Debug\funqciebi.exe (process
o automatically close the console when debugging stops, enable Tools->Optio
e when debugging stops.
ress any key to close this window . . .
```

3. შექმენით ფუნქცია, რომელიც დაბეჭდავს თქვენს სახელსა და გვარს. 3 ჯერ გამოიძახეთ ფუნქცია ძირითად პროგრამაში.

```
#include <iostream>
using namespace std;
void me_var(void);
int main()
{
    system("color F0");
  me_var();
  me_var();
```

```
  me_var();


}

void me_var(void) {
  cout << "me var sandro lomjaria"<<endl; //აქ აკრიფეთ თქვენი სახელი და გვარი

}
```

შედეგი:

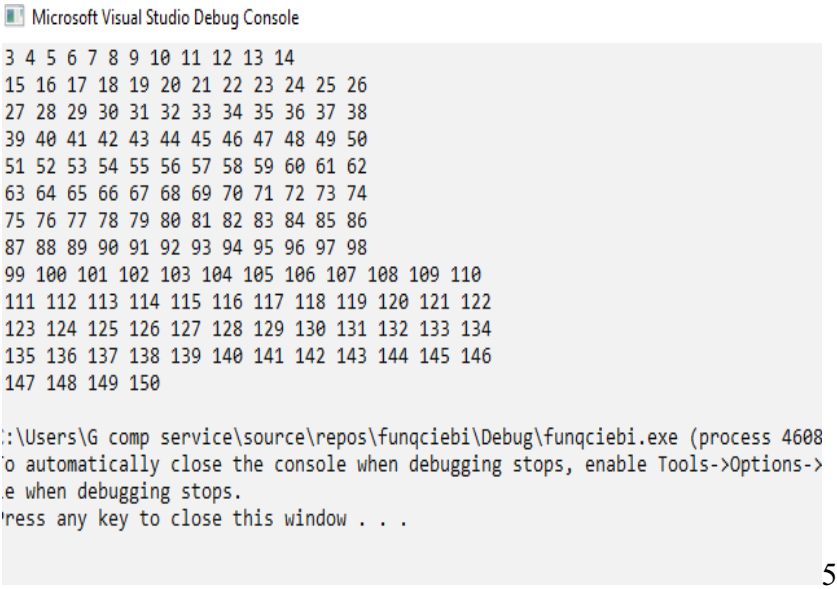
```
ıe var sandro lomjaria
ıe var sandro lomjaria
ıe var sandro lomjaria

:\Users\G comp service\source\repos\funqciebi\Debug\funqciebi.exe (process 4824) exited wi
o automatically close the console when debugging stops, enable Tools->Options->Debugging->
.e when debugging stops.
ress any key to close this window . . .
```

4. შექმენით ფუნქცია, რომელიც 12 სვეტად გამოიტანს ეკრანზე მთელ რიცხვებს 3-დან 150-მდე (ჩათვლით). გამოიძახეთ ფუნქცია ძირითად პროგრამაში.

```
#include <iostream>
using namespace std;
void print(void);
int main()
{
    system("color F0");
  print();

}

void print(void) {
  int i; /* mTeli ricxvi */
  int k; /* ricxvebis mTvleli */
  for (i = 3, k = 1; i <= 150; i++, k++) {
        cout << " " << i;
        if (k % 12 == 0) cout << "\n";
```

```
    }
    cout << "\n";
}
```

შედეგი:



5.

5. შექმენით ფუნქცია, რომელიც დააბრუნებს მთელი რიცხვის კუბს. ეს რიცხვი ფუნქციის პარამეტრია. გამოიყენეთ ფუნქცია `main`-ში და დაბეჭდეთ 1-დან 20-მდე რიცხვთა კუბები.

```
#include <iostream>
using namespace std;
int cube(int);
int main()
{
    system("color F0");
  int a; /* mTeli ricxvi */
  int n; /* funqciiT dabrunebuli mniSvneloba */
  for (a = 1; a <= 20; a++) {
        n = cube(a);
        cout << a << " ^ 3 = " << n << "\n";
```

```
  }

}

int cube(int x) {
  return x * x*x;
}
```

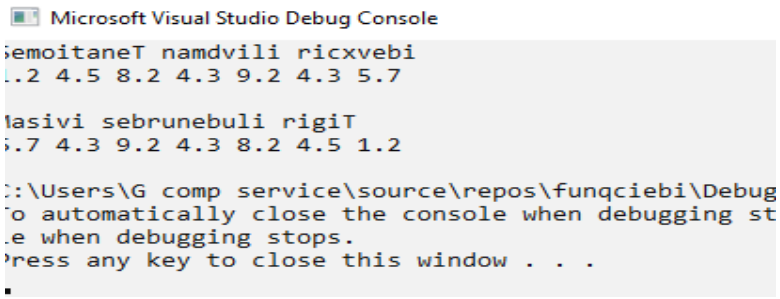
შედეგი:

6. დაწერეთ ფუნქცია, რომლის პარამეტრი არის ნამდვილ რიცხვთა მასივი და რომელიც დაბეჭდავს ამ მასივის ელემენტებს შებრუნებული რიგით. მასივში N რიცხვია.

```
#include <iostream>
using namespace std;
const int N = 7;
void bechdva(float a[]);
int main()
{
```

```
    system("color F0");
  float mass[N]; /* ganacxadi masivze */
  int i; /* masivis elementis indeqsi */
  cout << "SemoitaneT namdvili ricxvebi \n";
        for (i = 0; i < N; i++) cin >> mass[i];
  bechdva(mass);
}

void bechdva(float a[]) {
  int i;
  cout << "\nMasivi sebrunebuli rigiT\n";
  for (i = N - 1; i >= 0; i--)
        cout << a[i] << " ";
  cout << "\n";
}
```

შედეგი:

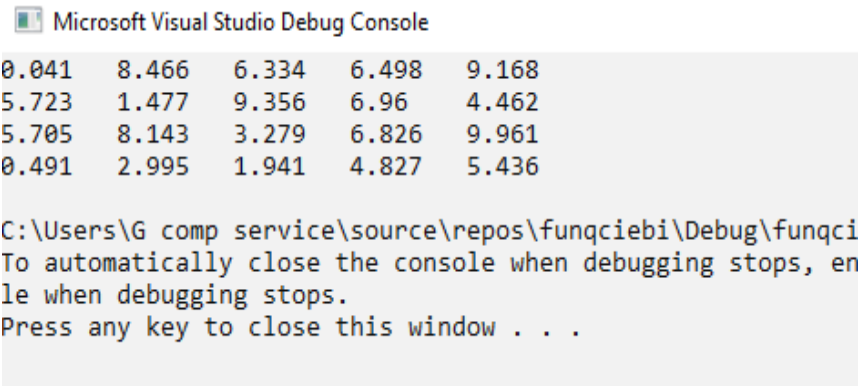


7. დაწერეთ 2 ფუნქცია: პირველმა უნდა შეავსოს $K \times P$ ორგანზომილებიანი მასივი (მატრიცა) შემთხვევითი ნამდვილი რიცხვებით `[0,10]` დიაპაზონიდან, მეორემ – დაბეჭდოს ეს მატრიცა სტრიქონ-სტრიქონ. K და P – შესაბამისად მატრიცის სტრიქონებისა და სვეტების რაოდენობა – შემოიღეთ კონსტანტების სახით. გამოიძახეთ ორივე ფუნქცია ძირითად პროგრამაში.

```
#include <iostream>
using namespace std;
const int K = 4, P = 5;
```

```
void inputArray(float a[][P]);
void printArray(float a[][P]);
int main()
{
    system("color F0");
  float matrix[K][P];
  inputArray(matrix);
  printArray(matrix);
}

void inputArray(float a[][P]) {
  int i, j;
  for (i = 0; i < K; i++)
        for (j = 0; j < P; j++)
              a[i][j] = rand() % 10001 / 1000.;
}
void printArray(float a[][P]) {
  int i, j;
  for (i = 0; i < K; i++) {
        for (j = 0; j < P; j++)
              cout << a[i][j] << "\t";
        cout << "\n";
  }
}
```

შედეგი:



დავალება:

1. შექმენით ფუნქცია, რომლის პარამეტრი არის მთელი რიცხვების მასივი. მასივის ელემენტთა რაოდენობა K წარმოადგენს კონსტანტას. ფუნქციამ უნდა დაადგინოს და დააბრუნოს მასივის:
   ა) 4-ის ჯერადი ელემენტების რაოდენობა. თუ ასეთი ელემენტები მასივში არ არის, დააბრუნოს –1;
   ბ) დადებითი ელემენტების რაოდენობა. თუ ასეთი ელემენტები მასივში არ არის, დააბრუნოს –1;
   გ) უდიდესი ელემენტის ინდექსი;
   დ) უმცირესი ელემენტის მნიშვნელობა.
2. შექმენით ფუნქცია, რომლის პარამეტრიც არის მთელ რიცხვთა მატრიცა $N \times M$. მატრიცის განზომილებები N და M წარმოადგენს კონსტანტებს. ფუნქციამ უნდა გამოითვალოს და დააბრუნოს მატრიცის 3-ის ჯერადი რიცხვების ჯამი.
3. შექმენით ფუნქცია, რომლის პარამეტრი არის ნამდვილ რიცხვთა მატრიცა $N \times M$. მატრიცის განზომილებები N და M წარმოადგენს კონსტანტებს. ფუნქციამ უნდა გამოითვალოს და დააბრუნოს მატრიცის:
   – არაუარყოფითი ელემენტების რაოდენობა;
   – იმ ელემენტების ჯამი, რომლებიც ეკუთვნის [-10, 15] შუალედს;
   – იმ ელემენტების რაოდენობა, რომლებიც არ ეკუთვნის [2, 30] შუალედს.
4. შექმენით ფუნქცია, რომლის პარამეტრი არის მთელი მატრიცა $M \times K$. მატრიცის განზომილებები M და K წარმოადგენს კონსტანტებს. ფუნქციამ უნდა დაადგინოს და დაბეჭდოს მასივის უდიდესი ელემენტის ინდექსები.

ლაბორატორიული სამუშაო 2
თემა: კლასი, ობიექტი. მარტივი კლასის შექმნა

კლასი წარმოადგენს მთავარ ინსტრუმენტულ საშუალებას C++ ენაში ობიექტზე ორიენტირებული დაპროგრამებისათვის. კლასი ძალიან ჰგავს სტრუქტურას, რომელშიც დაჯგუფებული მონაცემები წარმოადგენენ რაღაც ობიექტის აღწერას. ამ მონაცემებით სარგებლობენ ფუნქციები, რომელთაც მეთოდებს უწოდებენ.

ობიექტი არის არსება, როგორიც შეიძლება იყოს თანამშრომელი, წიგნი, სტუდენტი და სხვა. კლასს გამოვიყენებთ ამათუიმ ობიექტის განსაზღვრისათვის. კლასში უნდა ჩავრთოთ ობიექტის შესახებ იმდენი ინფორმაცია, რამდენსაც მოითხოვს ამოცანა. შეიძლება შეიქმნას კლასი ერთი პროგრამისთვის და გამოყენებულ იქნას სხვადასხვა პროგრამებისთვის.

განვიხილოთ კლასის ზოგადი სტრუქტურა:

```
class class_name

{
  int data_member; // მონაცემ-ელემენტი
  void show_member(int); // ფუნქცია-ელემენტი
};
```

როგორც ვხედავთ კლასს უნდა ჰქონდეს უნიკალური სახელი, შემდეგ იწერება ფიგურული ფრჩხილები, კლასის ელემენტები და იხურება ფიგურული ფრჩხილები.

კლასის გამოცხადების შემდეგ შეიძლება გამოვაცხადოთ ამ კლასის ტიპის ცვლადები, რომელთაც ჰქვია ობიექტები. ობიექტების გამოძახება ზოგადად ხდება შემდეგი სახით:

```
class_name object_one, object_two, object_three;
```

განვიხილოთ მაგალითი: შევქმნათ კლასი ***employee***, რომელიც შეიცავს მონაცემების და მეთოდების განსაზღვრას:

```
class employee

{
  public:
  char name[64] ;
  long employee_id;
  float salary;
  void show_employee(void)

  {
    cout << "saxeli: " << name << endl;
    cout << "TanamSromlis nomeri: " << employee_id << endl;
    cout << "xelfasi: " << salary << endl;
  };
};
```

მოცემულ მაგალითში კლასი შეიცავს სამ ცვლადს და ერთ ფუნქცია-ელემენტს. ყურადღება მიქციეთ ***public*** ჭდის გამოყენებას კლასის განსაზღვრაში. კლასის ელემენტები შეიძლება იყოს საჯარო და კერძო, რაზეც

დამოკიდებულია როგორ მიმართავს ჩვენი პროგრამა კლასის ელემენტებს. მოცემულ მაგალითში ყველა ელემენტი არის საჯარო, რაც ნიშნავს, რომ პროგრამა ნებისმიერ ელემენტს მიმართავს ოპერატორ წერტილის გამოყენებით.

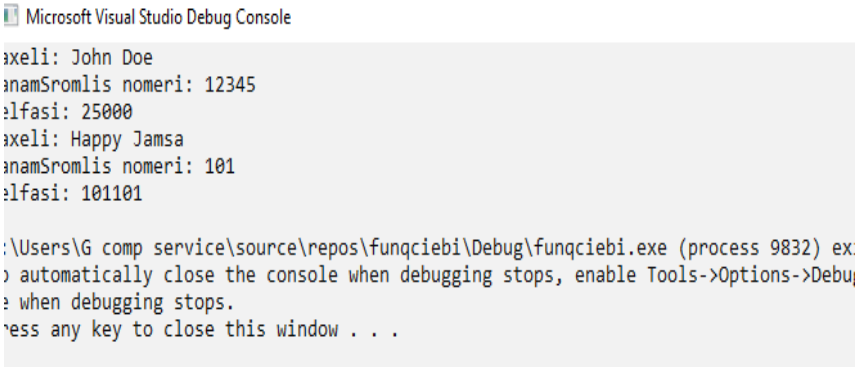
კლასის განსაზღვრის შემდეგ პროგრამაში შეგიძლიათ გამოაცხადოთ ობიექტები შემდეგი სახით:

employee worker, boss, secretary ;

განვიხილოთ პროგრამა, რომელიც ქმნის ***employee*** კლასის ორ ობიექტს. ოპერატორ წერტილის გამოყენებით პროგრამა ანიჭებს მონაცემებს მნიშვნელობებს. შემდგომ პროგრამა გამოიყენებს ***show_employee*** ფუნქციას თანამშრომლის შესახებ ინფორმაციის გამოსატანად:

```
#include <iostream>
#include <string>
using namespace std;
class employee
{
public:
  char name[64];
  long employee_id;
  float salary;
  void show_employee(void)
  {
         cout << "saxeli: " << name << endl;
         cout << "TanamSromlis nomeri: " << employee_id <<
endl;
         cout << "xelfasi: " << salary << endl;
  };
};

int main()
{
    system("color F0");
  employee worker, boss;
```

```
strcpy_s(worker.name, "John Doe");
worker.employee_id = 12345;
worker.salary = 25000;
strcpy_s(boss.name, "Happy Jamsa");
boss.employee_id = 101;
boss.salary = 101101.00;
worker.show_employee();
boss.show_employee();

}
```



როგორც ვხედავთ პროგრამა აცხადებს *employee* ტიპის ორ ობიექტს: *worker, boss.* შემდგომ იყენებს ოპერატორ წერტილს ელემენტებისათვის მნიშვნელობების მისანიჭებლად და *show_employee* ფუნქციის გამოსაძახებლად.

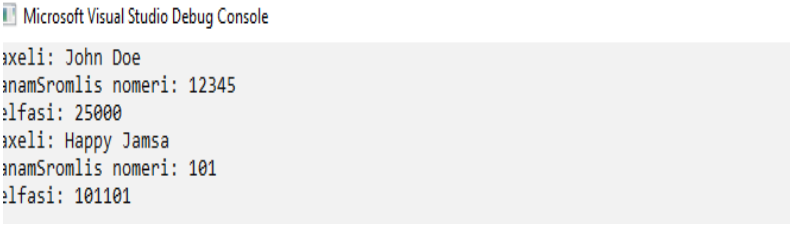
როგორც ვხედავთ ობიექტზე ორიენტირებულ დაპროგრამებაში ჩვენი პროგრამები ფოკუსირდება ობიექტებზე და ამ ობიექტებზე შესასრულებელ ოპერაციებზე.

განხილულ პროგრამაში name მონაცემი გამოვაცხადოთ როგორც სტრიქონის ტიპი. ამის გათვალისწინებით პროგრამა მიიღებს სახეს:

```
#include <iostream>
```

```
#include <string>
using namespace std;
class employee
{
public:
  string name;
  long employee_id;
  float salary;
  void show_employee(void)
  {
        cout << "saxeli: " << name << endl;
        cout << "TanamSromlis nomeri: " << employee_id <<
endl;
        cout << "xelfasi: " << salary << endl;
  };
};
int main()
{
    system("color F0");
  employee worker, boss;
  worker.name= "John Doe";
  worker.employee_id = 12345;
  worker.salary = 25000;
  boss.name= "Happy Jamsa";
  boss.employee_id = 101;
  boss.salary = 101101.00;
  worker.show_employee();
  boss.show_employee();
}
```

შედეგი:



დასკვნა:

- კლასის გამოცხადებისათვის პროგრამამ უნდა მიუთითოს კლასის სახელი, კლასის მონაცემ-ელემენტები და კლასის ფუნქციები(მეოდები);
- კლასის გამოცხადებას უზრუნველყოფს შაბლონი, რომლის დახმარებით თქვენი პროგრამა შეძლებს შექმნას ამ კლასის ტიპის ობიექტები;
- პროგრამა კლასის მონაცემ-ელემენტებს ანიჭებს მნიშვნელობებს ოპერატორ-წერტილის მეშვეობით;
- პროგრამა კლასის ფუნქცია-ელემენტს იძახებს ოპერატორ-წერტილის გამოყენებით.

დავალება: შექმენით კლასი სტუდენტი, მონაცემებით: გვარი, სახელი, ჩარიცხვის წელი, ჯგუფის ნომერი. კლასში განსაზღვრეთ ფუნქცია სტუდენტის მონაცემების გამოსატანად. მთავარ ფუნქციაში გამოაცხადეთ მოცემული კლასის ორი ობიექტი. ობიექტების ცვლადებს მიანიჭეთ მნიშვნელობები და შემდეგ გამოიძახეთ კლასის ფუნქცია ცალკეული ობიექტის მონაცემების გამოსატანად.

ლაბორატორიული სამუშაო 3
თემა: კლასის მეთოდების განსაზღვრა კლასის გარეთ

წინა მასალაში შექმნილ *employee* კლასში ფუნქცია განსაზღვრული იყო თვით კლასის შიგნით (ჩაშენებული (*inline*) ფუნქცია). ჩაშენებული ფუნქციის კლასის შიგნით განსაზღვრამ კლასის აღწერაში შეიძლება უწესრიგობა შეიტანოს. ალტერნატივის სახით ფუნქციის პროტოტიპი შეიძლება განვათავსოთ კლასის შიგნით, ხოლო შემდეგ ფუნქცია განვსაზღვროთ კლასის გარეთ. ამ შემთხვევაში კლასის განსაზღვრა მიიღებს შემდეგ სახეს:

```
class employee
{
public:
  string name;
  long employee_id;
  float salary;
  void show_employee(void); //ფუნქციის პროტოტიპი
};
```

რადგან სხვადასხვა კლასებში შეიძლება გამოვიყენოთ ფუნქციები ერთნაირი სახელებით, ამიტომ ფუნქციას უნდა დავურთოთ კლასის სახელი და გლობალური ნებართვის ოპერატორი (::). ამ შემთხვევაში ფუნქციის განსაზღვრა გამოიყურება შემდეგი სახით:

```
void employee:: show_employee (void) //კლასის სახელი,
                                     ფუნქციის სახელი
```

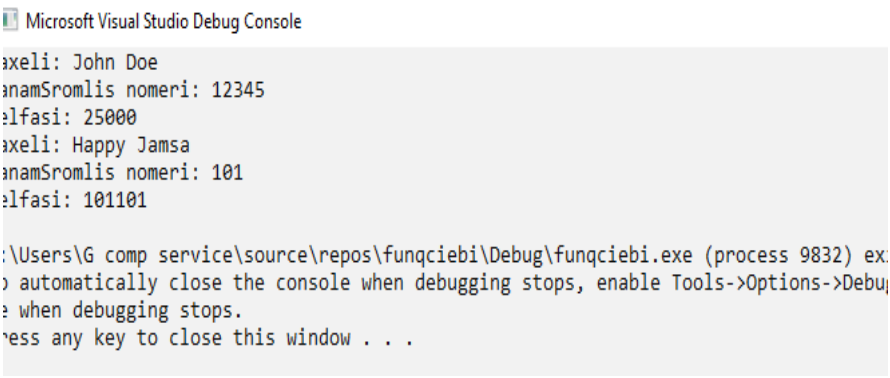
```
{
  cout << "saxeli: " << name << endl;
cout << "TanamSromlis  nomeri: " << employee_id << endl;
  cout << "xelfasi: " << salary << endl;
};
```

შემდეგი პროგრამა *show_employee* ფუნქციის განსაზღვრას განათავსებს კლასის გარეთ. კლასის სახელის მისათითებლად გამოიყენებს გლობალური ნებართვის ოპერატორს:

```
#include <iostream>
#include <string>
using namespace std;
class employee
{
public:
  string name;
  long employee_id;
  float salary;
  void show_employee(void);
};
void employee::show_employee(void)
{
  cout << "saxeli: " << name << endl;
  cout << "TanamSromlis nomeri: " << employee_id << endl;
  cout << "xelfasi: " << salary << endl;
};


int main()
{
    system("color F0");
  employee worker, boss;
  worker.name=John Doe";
  worker.employee_id = 12345;
  worker.salary = 25000;
  boss.name= "Happy Jamsa";
  boss.employee_id = 101;
  boss.salary = 101101.00;
  worker.show_employee();
```

```
        boss.show_employee();

    }
```

შედეგი:



კლასის მეთოდები

კლასი პროგრამას საშუალებას აძლევს ობიექტის მონაცემები და ობიექტის ფუნქციები(მეთოდები), რომლებიც ოპერირებენ ამ მონაცემებით, დაჯგუფებულ იქნას ერთ ცვლადში. ჩვენ გვაქვს ობიექტის მეთოდების განსაზღვრის ორი შესაძლებლობა. პირველი – ფუნქციის მთელი კოდი შეიძლება ჩართოთ კლასის განსაზღვრის შიგნით. შეიძლება ეს მოხერხებულად მოგვეჩვენოს, მაგრამ როდესაც კლასები რთულდება და მოიცავს რამდენიმე მეთოდს, ფუნქციის ოპერატორებმა შეიძლება უწესრიგობა შეიტანოს კლასის განსაზღვრაში. ამდენად უმეტესობა პროგრამებში ფუნქცია განისაზღვრება კლასის გარეთ. კლასის განსაზღვრაში უნდა იყოს ფუნქციის პროტოტიპი, რომელიც შეიცავს ფუნქციის სახელს, დასაბრუნებელი მნიშვნელობის ტიპს და პარამეტრების ტიპებს. კლასის

გარეთ ფუნქციის განსაზღვრა ზოგადად გამოიყურება შემდეგი სახით:

return_type class_name::function_name(parameters)
{ // ოპერატორები }

განვიხილოთ მაგალითი

პროგრამა ქმნის *dog* კლასს, რომელიც შეიცავს რამდენიმე მონაცემს და *show_breed* ფუნქციას. ფუნქცია განსაზღვრულია კლასის გარეთ. შემდეგ პროგრამა ქმნის *dog* ტიპის ორ ობიექტს და გამოიტანს ინფორმაციას ცალკეული ძაღლის შესახებ:

```
#include "pch.h"
#include <iostream>
#include <string>
#include<stdlib.h>
using namespace std;
class dogs
{
public:
  string breed;
  int average_weight;
  int average_height;
  void show_breed(void);
};

void dogs::show_breed(void)

{
  cout << "jishi: " << breed << endl;
  cout << "sashualo cona: " << average_weight << endl;
  cout << "sashualo simagle: " << average_height << endl;
}

int main()
{
```

```
        system("color F0");
        dogs happy, matt;
        happy.breed= "dalmatineli";
        happy.average_weight = 58;
        happy.average_height = 24;
        happy.matt= "kolli";
        matt.average_weight = 22;
        matt.average_height = 15;
        happy.show_breed();
        matt.show_breed();
        system("pause");
}
```

```
C:\Users\G comp service\source\repos\projeqt3\Debug\projeqt3.exe
jishi: dalmatineli
sashualo cona: 58
sashualo simagle: 24
jishi: kolli
sashualo cona: 22
sashualo simagle: 15
Press any key to continue . . .
```

ამ პროგრამაში გამოყენებულია ჭდე *public*, რომელიც კლასის ელემენტებს მისაწვდომს გახდის მთელი პროგრამისათვის.

დასკვნა:

მოცემული მასალის შესწავლის შედეგად თქვენ უნდა იცოდეთ, რომ:

- ობიექტი წარმოადგენს არსებას, რომელთან მიმართებით თქვენი პროგრამა ასრულებს სხვადასხვა ოპერაციას;
- პროგრამა C++ ენაზე წარმოადგენს ობიექტებს, რომელიც გადმოცემულია კლასების მეშვეობით;
- კლასი სტრუქტურის მსგავსად შეიცავს ელემენტებს. კლასის ელემენტები შეიძლება იყოს მონაცემები ან

ფუნქციები(მეთოდები), რომლებიც ოპერირებენ ამ მონაცემებით;

- ცალკეულ კლასს აქვს უნიკალური სახელი;
- კლასის განსაზღვრის შემდეგ თქვენ შეგიძლიათ გამოაცხადოთ ამ კლასის ობიექტები, სადაც გამოიყენებთ კლასის სახელს ტიპის სახით;
- კლასის ელემენტებზე მიმართვისათვის(როგორც მონაცემებზე, ასევე ფუნქციებზე) თქვენი პროგრამები გამოიყენებენ ოპერატორ წერტილს;
- ფუნქცია შეიძლება განისაზღვროს კლასის შიგნით ან გარეთ. თუ ფუნქციას განსაზღვრავთ კლასის განსაზღვრის გარეთ, უნდა მიუთითოთ კლასის სახელი და გლობალური ნებართვის სახელი, მაგალითად: *class:: function.*

დავალება: შექმენით კლასი ზღვა, რომელშიც გამოაცხადეთ მონაცემები: სახელი, სიღრმე, ფართობი. კლასში გამოაცხადეთ ფუნქციის პროტოტიპი, რომელიც გამოიტანს ზღვის მონაცემებს, ხოლო თვით ფუნქცია განსაზღვრეთ კლასის გარეთ. მთავარ ფუნქციაში შექმენით ზღვა კლასის სამი ობიექტი და ფუნქციის გამოძახებით გამოიტანეთ ცალკეული ზღვის მონაცემები.

ლაბორატორიული სამუშაო 4
თემა: კერძო და საჯარო მონაცემები

მოცემული მასალიდან თქვენ შეისწავლით თუ როგორ მართავენ public(საჯარო) და private(კერძო) ატრიბუტები პროგრამაში კლასის ელემენტებზე წვდომას. პროგრამა საჯარო ელემენტებს მიმართავს ნებისმიერი ფუნქციიდან. მეორეს მხრივ პროგრამამ შეიძლება მიმართოს კერძო ელემენტებს მხოლოდ მოცემული კლასის ფუნქციებიდან.

ინფორმაციის დაფარვა. თქვენს პროგრამას არ მოეთხოვება იცოდეს, თუ როგორ მუშაობენ მეთოდები. პროგრამამ უნდა იცოდეს მხოლოდ თუ რა ამოცანას ასრულებს მეთოდები. მაგალითად, გვაქვს კლასი file. თქვენმა პროგრამამ უნდა იცოდეს მხოლოდ ის, რომ ეს კლასი უზრუნველყოფს file.print მეთოდს, რომელიც ბეჭდავს მიმდინარე ფაილს ფორმატირებულ ასლს, ან file.delete, რომელიც წაშლის ფაილს. თქვენს პროგრამას არ მოეთხოვება როგორ მუშაობს ეს ორი მეთოდი. სხვა სიტყვებით, პროგრამა კლასს უნდა განიხილავდეს, როგორც „შავ ყუთს“. პროგრამამ იცის რომელი მეთოდები უნდა გამოიძახოს და რომელი პარამეტრები გადასცეს მას, მაგრამ პროგრამამ არაფერი იცის რეალურ სამუშაოზე, რომელიც სრულდება კლასის შიგნით. ინფორმაციის დაფარვა პროცესია, რომლის შედეგად პროგრამას წარედგინება მხოლოდ მინიმალური ინფორმაცია, რომელიც საჭიროა კლასის გამოსაყენებლად. კლასის საჯარო და კერძო ელემენტები დაგეხმარებათ მიიღოთ ინფორმაცია, რომელიც

დაფარულია პროგრამაში. წინა მეცადინეობაზე შექმნილ კლასებში გამოყენებული იყო public ჭდე, ამიტომ პროგრამა კლასის ნებისმიერ ელემენტს მიმართავდა ოპერატორ წერტილის გამოყენებით.

კლასის შექმნისას თქვენ შეიძლება გქონდეთ ელემენტები, რომელთა მნიშვნელობებიც გამოიყენება მხოლოდ კლასის შიგნით, მაგრამ მათზე მიმართვა პროგრამიდან არ არის აუცილებელი. ასეთი ელემენტები არის კერძო და ისინი დამალული უნდა იქნან პროგრამისაგან. თუ თქვენ არ გამოიყენებთ public ჭდეს, დუმილით კლასის ყველა ელემენტი ჩაითვლება კერძოდ. პროგრამა კერძო ელემენტებს ვერ მიმართავს ოპერატორ წერტილის გამოყენებით. კლასის კერძო ელემენტებზე მიმართვა შეუძლიათ მხოლოდ თვით ამ კლასის ელემენტებს.

კლასის შექმნისას ელემენტები უნდა დაყოთ საჯაროდ და კერძოდ, როგორც ქვემოთაა ნაჩვენები:

```
class some_class
{
public:
  int some_variable;
  void initialize_private(int, float);   //საჯარო ელემენტები
  void show_data(void);
private:
  int key_value;            // კერძო ელემენტები
  float key_number;
}
```

მოცემულ შემთხვევაში პროგრამას შეუძლია გამოიყენოს ოპერატორი წერტილი საჯარო

ელემენტებზე მიმართვისთვის, როგორც ქვემოთ არის ნაჩვენები:

```
some_class object; // შეიქმნას ობიექტი
object.some_variable = 1001;
object.initialize_private(2002, 1.2345);
object.show_data()
```

თუ პროგრამიდან ვეცდებით მივმართოთ key_value ან key_number კერძო ელემენტებს წერტილის გამოყენებით, კომპილატორი შეგვატყობინებს სინტაქსურ შეცდომაზე.

ელემენტების კერძო სახით გამოცხადებით თქვენ დაიცავთ კლასის ელემენტებს პირდაპირი წვდომისაგან. ასეთ ელემენტებს პროგრამა მნიშვნელობებს ვერ მიანიჭებს ოპერატორ წერტილის გამოყენებით. იმის ნაცვლად, რომ მიანიჭოს მნიშვნელობა, პროგრამამ უნდა გამოიძახოს კლასის მეთოდი.

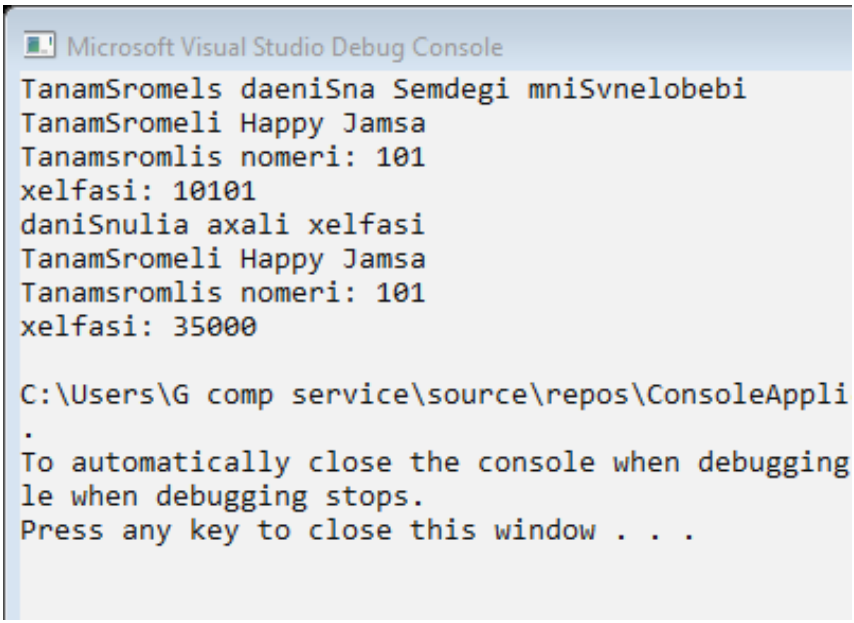
კლასის მეთოდები, რომლებიც მართავენ მონაცემთა ელემენტებზე წვდომას, წარმოადგენენ ინტერფეისულ ფუნქციებს. კლასების შექმნისას თქვენ გამოიყენებთ ასეთ ფუნქციებს კლასების მონაცემების დასაცავად.

შემდეგი განვიხილოთ პროგრამა, რომელიც იყენებს კლასის საჯარო და კერძო ნაწილს:

```
#include <iostream>
#include <string>
using namespace std;
class employee

{
public:
 int assign_values(string, long, float);
```

```
      void show_employee(void);
      int change_salary(float);
      long get_id(void);
    private:
    string name;
      long employee_id;
      float salary;
    };
    int employee::assign_values(string emp_name, long
emp_id, float emp_salary)

    {
      name= emp_name;
      employee_id = emp_id;
      if (emp_salary < 5000.0)
      {
            salary = emp_salary;
            return(0);
      }
      else return(-1);
    } // დაუშვებელი ხელფასი

    void employee::show_employee(void)
    {
      cout << "TanamSromeli " << name << endl;
      cout << "Tanamsromlis nomeri: " << employee_id
<< endl;
      cout << "xelfasi: " << salary << endl;
    }
    int employee::change_salary(float new_salary)

    {
      if (new_salary < 5000.0)

      {
            salary = new_salary; return(0);
```

```
    } // წარმატებით შესრულდა

    else return(-1);
  } // დაუშვებელი ხელფასი


  int main()
  {
    system("color F0");
    employee worker;

    if  (worker.assign_values("Happy  Jamsa",  101,
10101.0) == 0)

    {
          cout  <<  "TanamSromels  daeniSna  Semdegi
mniSvnelobebi" << endl;
    }
  worker.show_employee();
    if (worker.change_salary(3500.00) == 0)

    {
          cout  <<  "daniSnulia  axali  xelfasi"  <<
endl;
  worker.show_employee();
    }
    else cout << "naCvenebia dauSvebeli xelfasi" <<
endl;


  }
```



როგორც ხედავთ კლასი იცავს თავის ყველა მონაცემ-ელემენტს, რადგან აცხადებს როგორც კერძოს. პროგრამა ამ ელემენტებზე წვდომას ახორციელებს ინტერფეისული ფუნქციებით.

assign_values მეთოდი ინიციალიზებას უკეთებს კლასის კერძო მონაცემებს. მეთოდი იყენებს პირობის ოპერატორს, რათა დარწმუნდეს, რომ მიენიჭება დასაშვები ხელფასი. მეთოდს show_employee მოცემულ შემთხვევაში გამოაქვს კერძო მონაცემები. change_salary ახორციელებს ხელფასის ახალი ხელფასით შეცვლას, პირობის ჭეშმარიტობის შემთხვევაში. პროგრამის მიერ კერძო მონაცემებზე წვდომა განხორციელებულია ფუნქციებით. პროგრამის კომპილაციის და გაშვების შემდეგ, შეეცადეთ კერძო ელემენტებს მიმართოთ

წერტილით, კომპილატორი შეგატყობინებთ სინტაქსური შეცდომის შესახებ.

კლასის ელემენტებისათვის გამოიყენება გლობალური ნებართვის ოპერატორი იმისათვის, რომ გასაგები იყოს რომელი სახელები რომელ კლასს შეესაბამება. ქვემოთ მოყვანილ ფუნქციაში ჩანს, რომ მონაცემები შეესაბამება employee კლასს:

```
int employee::assign_values(string emp_name, long emp_id, float emp_salary)

{
  employee::name= emp_name;
  employee::employee_id = employee_id;
  if (emp_salary < 5000.0)

{
    employee::salary = emp_salary;

  return(0); // წარმატებით

}

 else
  return(-1);                //დაუშვებელი ხელფასი
}
```

ჩვენს მაგალითში კერძო ელემენტები იყო მონაცემ-ელემენტები. ზოგჯერ საჭიროა, რომ შეიქმნას ფუნქცია, რომელიც უნდა გამოიყენოს კლასის სხვა მეთოდებმა, მაგრამ პროგრამის დანარჩენი ნაწილისათვის ეს

ფუნქცია იყოს დახურული. ასეთ შემთხვევაში ასეთი ფუნქციები უნდა გამოცხადდეს როგორც კლასის კერძო ელემენტები.

განვიხილოთ მაგალითი, სადაც რადიუსი გამოცხადებულია, როგორც კლასის კერძო მონაცემ-ელემენტი, ხოლო ფუნქციები, როგორც საჯარო ელემენტები:

```
#include <iostream>
#include<math.h>
using namespace std;
class Circle
{  private:
      double radius;
   public:
     void  setRadius(double r)
      {  radius = r; }

   public:
   double getArea()
     {  return 3.14 * pow(radius, 2); }
};
 int main()
{Circle c1, c2;
 c1.setRadius(7);
 c2.setRadius(4);
```

```
cout<<c1.getArea()<<endl;
cout<<c2.getArea();
  }
```

დასკვნა:

- კლასის ელემენტები შეიძლება იყოს საერთო და კერძო. პროგრამა საერთო ელემენტებს მიმართავს პირდაპირ, ოპერატორ წერტილის გამოყენებით. კერძო ელემენტებზე მიმართვა ხდება კლასის მეთოდების(ფუნქციების) გამოყენებით.
- დუმილით C++ ყველა ელემენტს კერძოდ ჩათვლის.
- კლასის ელემენტებში შეგიძლიათ გამოიყენოთ კლასის სახელი და გლობალური ნებართვის ოპერატორი, მაგალითად employee::name, სახელების შესაძლო კონფლიქტის თავიდან ასაცილებლად.

დავალება: შექმენით კლასი მართკუთხედი. მართკუთხედის სიგრძე და სიგანე აღწერეთ საჯარო ნაწილში, ხოლო ფართობი და პერიმეტრი კერძო ნაწილში. საჯაროში დაასახელეთ ორი ფუნქციის პროტოტიპი, რომელთაგან ერთი ითვლის მართკუთხედის ფართობს, ხოლო მეორე-პერიმეტრს. ფუნქციები განსაზღვრეთ კლასის გარეთ. შექმენით მართკუთხედი კლასის ორი ობიექტი. ცალკეული ობიექტის ცვლადებს მიანიჭეთ მნიშვნელობები. გამოიძახეთ ცალკეული ობიექტისთვის ფართობის და პერიმეტროს გამოთვლის ფუნქციები. გააკეთეთ ანალიზი.

ლაბორატორიული სამუშაო 5

თემა: კონსტრუქტორი და დესტრუქტორი კონსტრუქტორის შექმნა, გამოძახება. კონსტრუქტორის გადატვირთვა. დესტრუქტორი. მისი შექმნა და გამოყენება

ობიექტის შექმნისას საჭიროა მათი ინიციალიზება. როგორც უკვე განვიხილეთ კერძო ელემენტებზე მიმართვა შესაძლებელია მხოლოდ კლასის ფუნქციებით. კლასის მონაცემების ინიციალიზაციის პროცესის გასამარტივებლად გამოიყენება სპეციალური ფუნქცია, რომელსაც ჰქვია კონსტრუქტორი, რომელიც გაიშვება ყოველი შექმნილი ობიექტისათვის. ხოლო დესტრუქტორი გაიშვება ობიექტის განადგურებისას.

დაიმახსოვრეთ, რომ:

- კონსტრუქტორი არის კლასის მეთოდი, რომელიც პროგრამას უადვილებს კლასის მონაცემ-ელემენტების ინიციალიზაციას;
- კონსტრუქტორს აქვს ისეთივე სახელი, როგორც კლასს;
- კონსტრუქტორს არა აქვს დასაბრუნებელი მნიშვნელობა;
- ყოველთვის, როდესაც პროგრამა ქმნის კლასის ცვლადს, C++ იძახებს კლასის კონსტრუქტორს, თუ იგი არსებობს;
- ობიექტისათვის გამოიყოფა მეხსიერება, ინფორმაციის შესანახად, როდესაც თქვენ ანადგურებთ ამ ობიექტს C++ იძახებს სპეციალურ დესტრუქტორს, რომელიც

ათავისუფლებს მეხსიერებას. ასუფთავებს მას ობიექტის შემდგომ.

- დესტრუქტორს აქვს ისეთივე სახელი, როგორც კლასს, მხოლოდ მას წინ უძღვის ~ სიმბოლო.
- დესტრუქტორს არა აქვს დასაბრუნებელი მნიშვნელობა.

კონსტრუქტორი წარმოიდგინეთ როგორც ფუნქცია, რომელიც გეხმარებათ თქვენ ობიექტის აშენებაში. ასევე დესტრუქტორი არის ფუნქცია, რომელიც გეხმარებათ ობიექტის განადგურებაში. დესტრუქტორი გამოიყენება მაშინ თუ ობიექტის განადგურებისას საჭიროა იმ მეხსიერების გათავისუფლება, რომელსაც იკავებს ობიექტი.

მარტივი კონსტრუქტორის შექმნა

შევქმნათ კლასი employee, შესაბამისად კოსტრუქტორსაც ექნება სახელი employee. კოსტრუქტორს არა აქვს დასაბრუნებელი მნიშვნელობა და იგი არც უნდა მიეთითოს.

```
class employee

{
public:
  employee(string, long, float); //კონსტრუქტორი
  void show_employee(void);
  int change_salary(float);
  long get_id(void);
private:
```

```
  string  name;
   long employee_id;
   float salary;
};
```

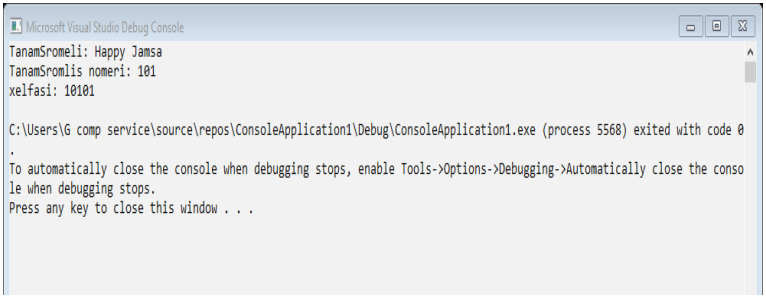
მოცემულ პროგრამში თქვენ უნდა განსაზღვროთ კონსტრუქტორი ისევე, როგორც კლასის სხვა ნებისმიერი მეთოდი:

```
employee::employee(string name, long employee_id, float salary)

{
   employee::name, name ;
   employee::employee_id = employee_id;
   if (salary < 50000.0)
   employee::salary = salary;
   else          //დაუშვებელი ხელფასი
   employee::salary = 0.0;
}
```

როგორც ხედავთ კონსტრუქტორი არ აბრუნებს მნიშვნელობას და არც გამოიყენება void ტიპი. მოცემულ შემთხვევაში კონსტრუქტორი გამოიყენებს გლობალური ნებართვის ოპერატორს და კლასის სახელს ცალკეული ელემენტის სახელის წინ. განვიხილოთ ჩვენი მაგალითი:

```
#include <iostream>
#include<string>
using namespace std;
class employee
```

```
{
public:
employee (string, long, float);
void show_employee(void);
int change_salary(float);
long get_id(void);
private:
string name;
  long employee_id;
  float salary;
};
employee::employee(string name, long employee_id, float
salary)

{
employee::name= name;
  employee::employee_id = employee_id;
  if (salary < 50000.0)
         employee::salary = salary;
  else // dauSvebeli xelfasi
         employee::salary = 0.0;
}
void employee::show_employee(void)
{
  cout << "TanamSromeli: " << name <<endl;
  cout << "TanamSromlis nomeri: " << employee_id << endl;
  cout << "xelfasi: " << salary << endl;
}
int main()
{
  system("color F0");
  employee worker("Happy Jamsa", 101, 10101.0);
worker.show_employee();

}
```



მიაქციეთ ყურადღება, რომ worker ობიექტის გამოცხადების შემდეგ მოდის მრგვალი ფრჩხილები და საწყისი მნიშვნელობები, ისე როგორც ფუნქციის გამოცხადებისას. **როდესაც თქვენ იყენებთ კონსტრუქტორს, მას გადასცემთ პარამეტრებს ობიექტის გამოცხადებისას:**

employee worker("Happy Jamsa", 101, 10101.0);

თუ პროგრამაში მოითხოვება შეიქმნას employee კლასის რამდენიმე ობიექტი, თქვენ შეგიძლიათ ინიციალიზება გაუკეთოთ ცალკეულ მათგანს კონსტრუქტორის მეშვეობით, როგორ ქვემოთ არის ნაჩვენები:

employee worker("Happy Jamsa", 101, 10101.0);

employee secretary("John Doe", 57, 20000.0);

employee manager("Jane Doe", 1022, 30000.0);

ობიექტის ინიციალიზება კონსტრუქტორის მეშვეობით ზოგადად გამოიყურება შემდეგი სახით:

class_name object(valuel, value2, value3)

კონსტრუქტორი და პარამეტრები დუმილით

კონსტრუქტორში შეიძლება მითითებულ იქნას პარამეტრის მნიშვნელობა დუმილით. განვიხილოთ მაგალითი, სადაც employee კონსტრუქტორი იყენებს ხელფასს, რომლის დუმილით მნიშვნელობა არის 1000, თუმცა პროგრამაში უნდა მიეთითოს თანამშრომლის სახელი და ნომერი:

```
employee::employee(string name, long employee_id, float salary = 10000.00)

{
 employee.name= name;
  employee::employee_id = employee_id;
  if (salary < 50000.0)
    employee::salary = salary;
  else //დაუშვებელი ხელფასი
    employee::salary = 0.0;
}
```

კონსტრუქტორის გადატვირთვა

კლასი შეიძლება შეიცავდეს რამდენიმე კონსტრუქტორს. მაგალითად ქვემოთ მოყვანილი ორივე კონსტრუქტორს:

employee(string, long, float);

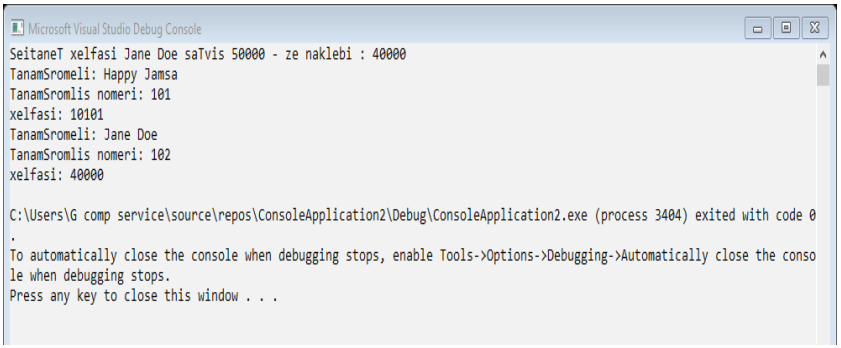
employee(string, long);

ამას ეწოდება კონსტრუქტორის გადატვირთვა.

განვიხილოთ ჩვენი პროგრამის რეალიზაცია, კონსტრუქტორის გადატვირთვის გათვალისწინებით:

```
#include <iostream>
#include <string>
using namespace std;
class employee
{
public:
  employee(string, long, float);
  employee(string, long);
  void show_employee(void);
private:
  string name;
  long employee_id;
  float salary;
};
employee::employee(string name, long employee_id, float
salary)
{
  employee::name = name;
  employee::employee_id = employee_id;
  if (salary < 50000.0) employee::salary = salary;
  else // dauSvebeli xelfasi
        employee::salary = 0.0;
}
employee::employee(string name, long employee_id)
{
  employee::name = name;
  employee::employee_id = employee_id;
  do
  {
        cout << "SeitaneT xelfasi " << name << " saTvis
50000 - ze naklebi : ";
        cin >> employee::salary;
  } while (salary >= 50000.0);
}
void employee::show_employee(void)
{
  cout << "TanamSromeli: " << name << endl;
  cout << "TanamSromlis nomeri: " << employee_id << endl;
```

```
  cout << "xelfasi: " << salary << endl;
}

int main()
{
  system("color F0");
  employee worker("Happy Jamsa", 101, 10101.0);
  employee manager("Jane Doe", 102);
  worker.show_employee();
  manager.show_employee();
}
```



პროგრამის გაშვებისას გამოვა მოთხოვნა: შეიტანეთ ხელფასი Jane Doe - სათვის. ხელფასის შეტანის შემდეგ გამოვა ინფორმაცია ორივე თანამშრომლის შესახებ.

დესტრუქტორი

დესტრუქტორი ავტომატურად გაიშვება თითოეულ ჯერზე, როდესაც პროგრამა ანადგურებს ობიექტს. მოცემული მომენტისათვის თქვენ შეგიძლიათ შექმნათ და გაანადგუროთ ობიექტები პროგრამის შესრულების მომენტში. ასეთ შემთხვევებში აზრი აქვს დესტრუქტორის გამოყენებას.

დესტრუქტორი არის კლასის ფუნქცია-წევრი, რომელიც ავტომატურად გამოიძახება მაშინ, როდესაც ობიექტი გამოდის მოქმედების არედან: დესტრუქტორი მონიშნავს ასეთი ობიექტის მიერ დაკავებულ მეხსიერებას, როგორც თავისუფალს. დესტრუქტორის დასახელება შედგება: ~ (ტილდა) სიმბოლოსაგან, რომელსაც მოჰყვება კლასის დასახელება. დესტრუქ-ტორი არ აბრუნებს მნიშვნელობას და მას არა აქვს პარამეტრები. კლასში უნდა იყოს აღწერილი მხოლოდ ერთი დესტრუქტორი. დესტრუქტორების გადატვირთვა დაშვებული არ არის.

დესტრუქტორის ზოგადი სახეა:

~class_name (void)

{დესტრუქტორის ოპერატორები}

კონსტრუქტორისგან განსხვავებით თქვენ არ შეგიძლიათ გადასცეთ პარამეტრები დესტრუქტორს. განვსაზღვროთ დესტრუქტორი **employee** კლასისათვის:

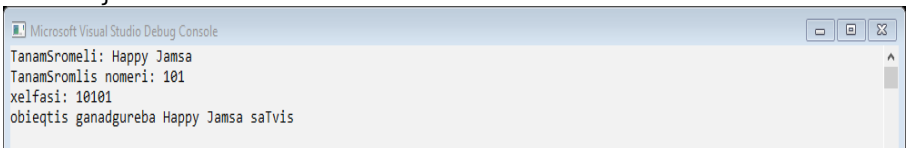
```
void employee:: ~ employee(void )

{
  cout << "ობიექტის  განადგურება " << name <<”სათვის”<<
endl;
}
```

მოცემულ შემთხვევაში დესტრუქტორს უბრალოდ გამოაქვს ეკრანზე შეტყობინება იმის შესახებ, რომ c++ ანადგურებს ობიექტს. პროგრამის დასრულებისას

დესტრუქტორი ავტომატურად გამოიძახება ცალკეული ობიექტისათვის. განვიხილოთ პროგრამა:

```
#include <iostream>
#include <string>
using namespace std;
class employee
{
public:
  employee(string, long, float);
  ~employee(void);
  void show_employee(void);
  int change_salary(float);
  long get_id(void);
private:
  string name;
  long employee_id;
  float salary;
};
employee::employee(string name, long employee_id, float
salary)
{
  employee::name = name;
  employee::employee_id = employee_id;
  if (salary < 50000.0) employee::salary = salary;
  else // dauSvebeli xelfasi
         employee::salary = 0.0;
}
employee:: ~employee(void)
{
  cout << "obieqtis ganadgureba " << name << " saTvis" <<
  endl;

}
void employee::show_employee(void)
{
  cout << "TanamSromeli: " << name << endl;
```

```
  cout << "TanamSromlis nomeri: " << employee_id << endl;
  cout << "xelfasi: " << salary << endl;
}
int main()
{
  system("color F0");
  employee worker("Happy Jamsa", 101, 10101.0);
  worker.show_employee();
  }
```



როგორც ხედავთ პროგრამა ავტომატურად იძახებს დესტრუქტორს, დესტრუქტორის ფუნქციის ყოველგვარი ცხადი გამოძახების გარეშე.

ამდენად, როდესაც პროგრამა დაიწყებს მეხსიერების განაწილებას ობიექტების შიგნით თქვენ აღმოაჩენთ, რომ დესტრუქტორი მოხერხებული საშუალებაა ობიექტის განადგურებისას მეხსიერების განთავისუფლებისთვის.

დასკვნა:

- კონსტრუქტორი და დესტრუქტორი კლასის სპეციალური ფუნქციებია, რომელთაც პროგრამა ავტომატურად იძახებს ობიექტის შექმნის ან განადგურების დროს;
- კონსტრუქტორს პროგრამა გამოიძახებს თითოეულ ჯერზე ობიექტის შექმნისას. კონსტრუქტორს იგივე სახელი აქვს, რაც კლასს;
- კონსტრუქტორს არა აქვს დასაბრუნებელი მნიშვნელობა და საერთოდ იგი არ უნდა მიეთითოს;
- როდესაც თქვენი პროგრამა ქმნის ობიექტს, მას შეუძლია გადასცეს პარამეტრები კონსტრუქტორს ობიექტის გამოცხადების დროს;

- C++ ენა კონსტრუქტორის გადატვირთვის საშუალებას იძლევა და შესაძლებელია პარამეტრებისათვის გამოყენებულ იქნას მნიშვნელობები დუმილით;
- უმეტესობა პროგრამები კონსტრუქტორს იყენებს კლასის მონაცემ-ელემენტების ინიციალიზებისათვის;
- მარტივი პროგრამები არ მოითხოვენ დესტრუქტორის გამოყენებას;
- დესტრუქტორი არის სპეციალური ფუნქცია, რომელსაც პროგრამა იძახებს ავტომატურად თითოეულ ჯერზე ობიექტის განადგურებისას;
- დესტრუქტორს აქვს იგივე სახელი როგორც კლასს, მხოლოდ მის წინ იწერება ტილდა ნიშანი;
- დესტრუქტორი არ აბრუნებს მნიშვნელობას და მას არა აქვს პარამეტრები.

დავალება 1. შექმენით კლასი სტუდენტი. საჯარო ნაწილში იყოს კონსტრუქტორი პარამეტრებით: სტუდენტის გვარი, სახელი, პირადი ნომერი, საშუალო ქულა, სწავლების წელი. თვით ეს მონაცემები გამოაცხადეთ კლასის კერძო ნაწილში. შექმენით ფუნქცია ამ მონაცემების გამოსატანად. მთავარ ფუნქციაში გამოაცხადეთ სტუდენტი კლასის ორი ობიექტი. გამოიტანეთ ობიექტების მონაცემები ეკრანზე.

დავალება 2: შექმენით კლასი სტუდენტი. ღია ნაწილში იყოს ორი კონსტრუქტორი, პირველის პარამეტრებია: სტუდენტის გვარი, პირადი ნომერი და საშუალო ქულა, ხოლო მეორესი: გვარი და პირადი ნომერი. თვით ეს მონაცემები გამოაცხადეთ დახურულ ნაწილში. შექმენით ამ კლასის ორი ობიექტი. პირველი - სამი პარამეტრით, ხოლო მეორე - ორით. სტუდენტის ინფორმაცია გამოიტანეთ ეკრანზე. პროგრამაში გამოიყენეთ დესტრუქტორი.

ლაბორატორიული სამუშაო 6
თემა: ოპერატორების გადატვირთვა

ოპერატორის გადატვირთვა მდგომარეობს ოპერატორის აზრის შეცვლაში განსაზღვრულ კლასთან მისი გამოყენებისას. მოცემულ სამუშაოში თქვენ განსაზღვრავთ string კლასს და გადატვირთავთ ოპერატორ პლუს და მინუსს.

<u>დაიმახსოვრეთ შემდეგი ძირითადი საკითხები:</u>

- ოპერატორების გადატვირთვა საჭიროა თქვენი პროგრამის წაკითხვადობის გასაუმჯობესებლად, ოპერატორების გადატვირთვა საჭიროა მაშინ როცა ეს ამარტივებს პროგრამის გააზრებას;
- ოპერატორების გადასატვირთად პროგრამები იყენებენ გასაღებურ სიტყვას *operator*;
- ოპერატორის ხელახალი განსაზღვრისას თქვენ მიუთითებთ ფუნქციას, რომელსაც c++ იძახებს თითოეულ ჯერზე, როდესაც კლასი იყენებს გადატვირთულ ოპერატორს. ეს ფუნქცია თავის მხრივ ასრულებს შესაბამის ოპერაციას;
- თუ პროგრამა გადატვირთავს ოპერატორს განსაზღვრული კლასისათვის, მაშინ ამ ოპერატორის აზრი იცვლება მხოლოდ მითითებული კლასისათვის, პროგრამის დანარჩენი ნაწილი განაგრძობს ამ ოპერატორის გამოყენებას მისი სტანდარტული ოპერაციების შესასრულებლად;

– c++ უმეტესი ოპერატორების გადატვირთვის საშუალებას იძლევა, გარდა ოთხისა, რომელიც მოცემულია ცხრილში(გვ.50).

პლუს და მინუს ოპერატორების გადატვირთვა

განვიხილოთ მაგალითი, სადაც განსაზღვრულია string კლასი. ეს კლასი შეიცავს ერთ მონაცემ-ელემენტს, ეს არის სიმბოლოების სტრიქონი. გარდა ამისა შეიცავს რამდენიმე მეთოდს:

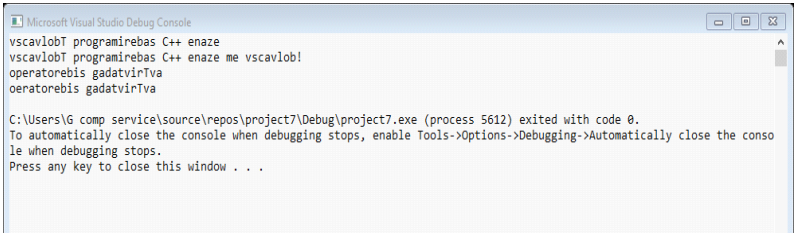
```
class string
{
public:
  string(char *); // კონსტრუქტორი
  void str_append(char *);
  void chr_minus(char);
  void show_string(void);
private:
  char data[256] ;
};
```

str_append ფუნქცია უზრუნველყოფს კლასის სტრიქონისათვის მითითებული სიმბოლოების დამატებას. *chr_minus* ამოაგდებს სტრიქონიდან მითითებული სიმბოლოს ყოველ შესვლას. განვიხილოთ პროგრამა მთლიანობაში, სადაც შექმნილია ორი ობიექტი, რომელიც წარმოადგენს სიმბოლოების სტრიქონს:

```
#include <iostream>
#include<string>
using namespace std;
class strings

{
```

```
public:
  strings(char *); // კონსტრუქტორი
  void str_append(char *);
  void chr_minus(char);
void show_strings(void);
private:
  char data[256] ;
};

  strings::strings(char *str)

  {
        strcpy_s(data, str);
  }

  void strings::str_append(char *str)
  {
        strcat_s(data, str);
  }

  void strings::chr_minus(char letter)

  {
        char temp[256];
        int i, j;
        for (i = 0, j = 0; data[i]; i++)
  if (data[i] != letter)
        temp[j] = NULL; // temp დასასრული
  //გადავაკოპიროთ temp-ის შემადგენლობა უკან data-ში
        strcpy_s(data, temp);
  }

              void strings::show_strings(void)

        {
              cout << data << endl;
        }

        int main()
{
  system("color F0");
strings title("vscavlobT programirebas C++ enaze");
  strings lesson("operatorebis gadatvirTva");
```

```
    title.show_strings();
    title.str_append(" me vscavlob!");
    title.show_strings();
    lesson.show_strings();
    lesson.chr_minus('p');
    lesson.show_strings();

}
```



როგორც ვხედავთ პროგრამა გამოიყენებს str_append ფუნქციას სტრიქონულ ცვლად title-სათვის სიმბოლოების დასამატებლად. პროგრამა გამოიყენებს chr_minus ფუნქციას ცალკეული “P” სიმბოლოს წასაშლელად lesson სიმბოლური სტრიქონიდან. მოცემულ შემთხვევაში პროგრამა იძახებს ფუნქციებს ამ ოპერაციების შესასრულებლად. თუ გამოვიყენებთ ოპერატორების გადატვირთვას, პროგრამა შეასრულებს იდენტურ ოპერაციებს (+) და (-) ოპერატორების დახმარებით.

ოპერატორის გადატვირთვისას გამოიყენება operator გასაღებური სიტყვა, თავისი პროტოტიპით და ფუნქციის განსაზღვრით. განვიხილოთ კლასი, სადაც გამოყენებულია operator გასაღებური სიტყვა, რათა დაენიშნოს პლუს და მინუს ოპერატორები str_append და chr_minus ფუნქციებს string კლასის შიგნით:

```
class string

{
public:
  string(char *); // კონსტრუქტორი
  void operator +(char *);
  void operator -(char); //კლასის ოპერატორების
განსაზღვრა
  void show_string(void);
private:
  char data[256];
};
```

როგორც ხედავთ კლასი გადატვირთავს პლუს და მინუს ოპერატორებს. როგორც ვთქვით, როდესაც კლასი გადატვირთავს ოპერატორს, მან უნდა უჩვენოს ფუნქცია, რომელიც რეალიზებას უკეთებს ოპერაციას, რომელიც შეესაბამება ამ ოპერატორს. პლუს ოპერატორის შემთხვევაში ასეთი ფუნქციის განსაზღვრას ექნება შემდეგი სახე:

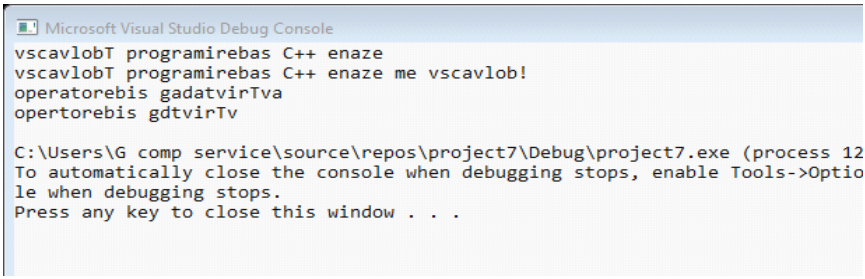
```
void string::operator +(char *str)

{
  strcat(data, str);
}
```

როგორც ხედავთ, ამ ფუნქციის განსაზღვრა არ შეიცავს სახელს, ეს კლასის გადატვირთული ოპერატორია. ამ ფუნქციის კოდი დარჩა იგივე. განვიხილოთ წინა პროგრამა გადატვირთული ოპერატორების გამოყენებით:

```
#include <iostream>
```

```
#include<string>
using namespace std;
class strings

{
public:
  strings(char *); // konstruqtori
  void operator +(char *);
  void operator -(char); //klasis operatorebis gansazRvra
void show_strings(void);
private:
  char data[256] ;
};

  strings::strings(char *str)

  {
        strcpy_s(data, str);
  }

  void strings::operator +(char *str)
  {
        strcat_s(data, str);
  }

  void strings::operator -(char letter)

  {
        char temp[256];
        int i, j;
        for (i = 0, j = 0; data[i]; i++)
              if (data[i] != letter)
                    temp[j++] = data[i];
        temp[j] = NULL;
        strcpy_s(data, temp);
  }

              void strings::show_strings(void)

        {
              cout << data << endl;
        }
```

```
        int main()
{
  system("color F0");
strings title("vscavlobT programirebas C++ enaze");
  strings lesson("operatorebis gadatvirTva");
  title.show_strings();
  title + " me vscavlob!";
  title.show_strings();
  lesson.show_strings();
  lesson - 'a';
  lesson.show_strings();
}
```



როგორც ხედავთ პროგრამა იყენებს გადატვირთულ ოპერატორებს:

title + " me vscavlob!"; // დაემატოს ტექსტი **"me vscavlob!"**

lesson - 'a'; // წაიშალოს სიმბოლო 'a'

(შეგახსენებთ, რომ ფუნქცია strcat, აღწერილი string.h-ში, ორპარამეტრიანია. strcat(s1,s2) შესრულების შედეგად s2 სტრიქონი მიუერთდება s1-ს და გადაბმული სტრიქონი განთავსდება s1-ში, ამასთან, s2 სტრიქონი არ შეიცვლება. ცხადია, რომ s1-ის განზომილება თავიდანვე

უნდა შეირჩეს ისე, რომ შერწყმით მიღებული ახალი სტრიქონი დაეტიოს s1-ის მეხსიერებაში.)

ცხრილში მოყვანილია ოპერატორები, რომლებიც არ გადაიტვირთება:

ოპერატორი	დანიშნულება	მაგალითი
.	ელემენტის ამორჩევა	object.member
.*	ელემენტზე მიმთითებელი	object.*member
: :	ხედვის არეს ნებართვა	classname::member
? :	პირობის ოპერატორი	c=(a>b)?a:b;

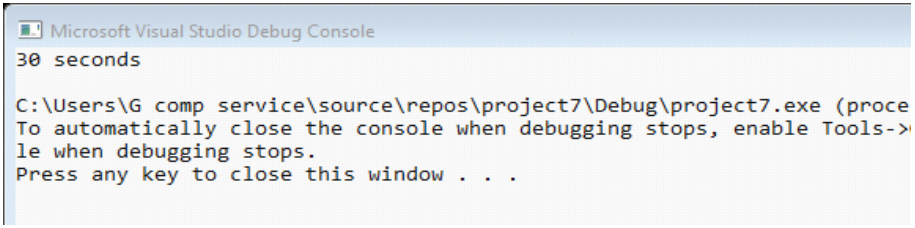
ოპერატორების გადატვირთვა არის შესაძლებლობა, რათა ოპერატორს მიენიჭოს ახალი არსი განსაზღვრულ კლასში მისი გამოყენებისას.

განვიხილოთ მაგალითები:

1) + ოპერატორის გადატვირთვა;

```
#include "pch.h"
#include <iostream>
#include<string>
#include<stdlib.h>
using namespace std;
class Counter
{
public:
  Counter(int sec)
  {
        seconds = sec;
  }

  void display()
  {
        cout << seconds << " seconds" << endl;
```

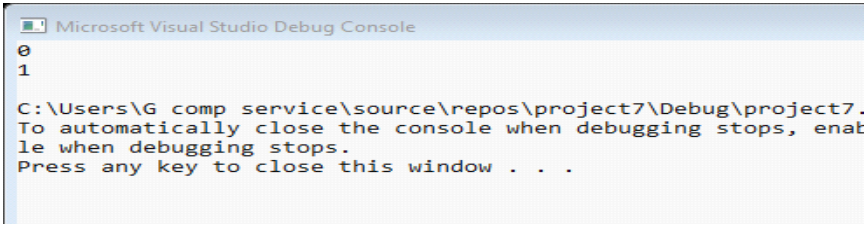
```
  }
  int seconds;
};
Counter operator + (Counter c1, Counter c2)
{
        return Counter(c1.seconds + c2.seconds);

}
        int main()
{
                system("color F0");
                Counter c1(20);
                Counter c2(10);
                Counter c3 = c1 + c2;
                c3.display(); // 30 seconds
return 0;
}
```



2) == ოპერატორის გადატვირთვა:

```
#include <iostream>
#include<string>
using namespace std;
class Counter
{
public:
  Counter(int sec)
  {
        seconds = sec;

  }
  void display()
  {
        cout << seconds << " seconds" << endl;
```

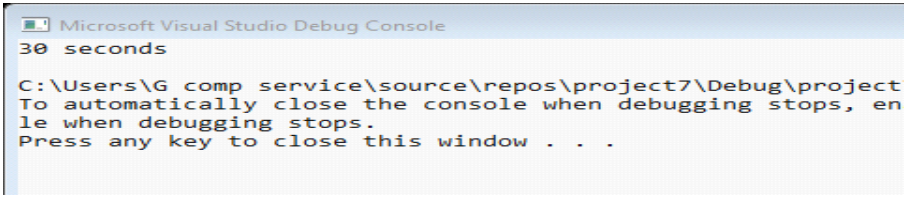
```
                }
  int seconds;
};
bool operator == (Counter c1, Counter c2)
{
  return c1.seconds == c2.seconds;
}
bool operator != (Counter c1, Counter c2)
{
        return c1.seconds != c2.seconds;
}
bool operator > (Counter c1, Counter c2)
{
        return c1.seconds > c2.seconds;}

bool operator < (Counter c1, Counter c2)
  {
        return c1.seconds < c2.seconds;
        }
        int main()
{
                system("color F0");
                Counter c1(20);
                Counter c2(10);
                bool b1 = c1 == c2; // false
                bool b2 = c1 > c2; // true
                cout << b1 << std::endl;
                cout << b2 << std::endl;
                return 0;
        }
```



3) += ოპერატორის გადატვირთვა:

```
#include <iostream>
#include<string>
```

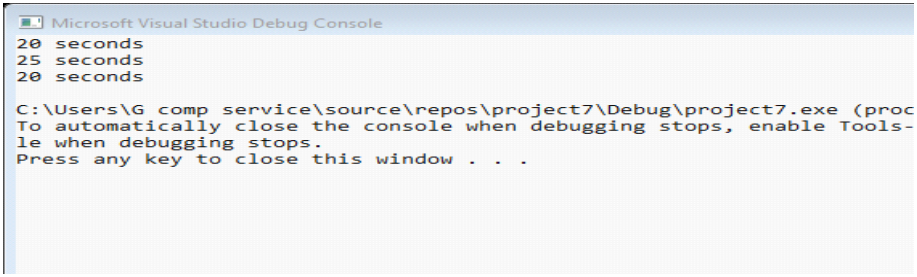
```
using namespace std;
class Counter
{
public:
  Counter(int sec)
  {
        seconds = sec;

  }
  void display()
  {
        cout << seconds << " seconds" << endl;
              }
  Counter& operator += (Counter c2)

  {

        seconds += c2.seconds;

        return *this;

  }

  int seconds;

};

        int main()
{
              system("color F0");
              Counter c1(20);

              Counter c2(10);

              c1 += c2;

              c1.display(); // 30 seconds
              return 0;

}
```



4) -- ოპერატორის გადატვირთვა:

```
#include <iostream>
#include<string>
using namespace std;
class Counter
{
public:
  Counter(int sec)
  {
         seconds = sec;
                }
  void display()
  {
         cout << seconds << " seconds" << endl;
                }

  // პრეფიქსული ოპერატორი

Counter& operator++ ()

{
  seconds += 5;

  return *this;
}

Counter& operator-- ()

{
  seconds -= 5;
  return *this;

}
// პოსტფიქსული ოპერატორი
```

```
Counter operator++ (int)
{
        Counter prev = *this;

  ++*this;

  return prev;
}

Counter operator-- (int)
{
        Counter prev = *this;

  --*this;

  return prev;
}
int seconds;

};

        int main(){
               system("color F0");
               Counter c1(20);
               Counter c2 = c1++;
               c2.display(); // 20 seconds
               c1.display(); // 25 seconds
               --c1;
               c1.display(); // 20 seconds
               return 0;

}
```



დასკვნა:

- ოპერატორის გადატვირთვისათვის უნდა განსაზღვროთ კლასი, რომელსაც დაენიშნება ეს ოპერატორი;
- ოპერატორის გადატვირთვა მოქმედებს მხოლოდ იმ კლასისათვის, რომელშიც იგი განისაზღვრება;
- კლასის ოპერატორის გადატვირთვისათვის გამოიყენება გასაღებური სიტყვა operator კლასის მეთოდის განსაზღვრისათვის, რომელსაც C++ იძახებს თითოეულ ჯერზე, როდესაც კლასის ცვლადი გამოიყენებს ამ ოპერატორს.

ლაბორატორიული სამუშაო 7
თემა: სტატიკური ფუნქციები და მონაცემ-ელემენტები

აქამდე განხილულ მასალაში ჩვენ მიერ შექმნილ ცალკეულ ობიექტს ჰქონდა თავისი მონაცემთა ნაკრები. შეიძლება იყოს სიტუაციები, როდესაც ერთიდაიმავე კლასის ობიექტებმა ერთობლივად უნდა გამოიყენონ ერთი ან რამდენიმე მონაცემ-ელემენტი. მაგალითად ვწერთ ხელფასის პროგრამას, რომელიც იკვლევს სამუშაო დროს 1000 თანამშრომლისათვის. სადაბეგვრო განაკვეთის განსაზღვრისათვის პროგრამამ უნდა იცოდეს პირობა, რომელშიც მუშაობს თითოეული თანამშრომელი. დავუშვათ ამისათვის გამოიყენება ცვლადი კლასი state_of_work. თუ ყველა თანამშრომელი მუშაობს ერთნაირ პირობებში, პროგრამას შეუძლია ერთობლივად გამოიყენოს ეს მონაცემები employee ტიპის ყველა ობიექტისათვის. ამდენად, თქვენი პროგრამა მოიშორებს რა ერთნაირი ინფორმაციის 999 ასლს, შეამცირებს მეხსიერების საჭირო რაოდენობას. კლასის ელემენტის ერთობლივი გამოყენებისათვის თქვენ ეს ელემენტი უნდა გამოაცხადოთ როგორც static(სტატიკური).

მასალაში თქვენ შეისწავლით შემდეგი ძირითად კონცეფციებს:

– C++ საშუალებას იძლევა გვქონდეს ერთნაირი ტიპის ობიექტები, რომლებიც ერთობლივად იყენებენ კლასის ერთ ან რამდენიმე ელემენტს;

- თუ თქვენი პროგრამა მიანიჭებს მნიშვნელობას ერთობლივად გამოსაყენებელ ელემენტს, მაშინ ამ კლასის ყველა ობიექტი მაშინვე მიიღებს წვდომას ამ ახალ მნიშვნელობაზე;
- კლასის ერთობლივად გამოსაყენებელი მონაცემ-ელემენტის შესაქმნელად კლასის სახელის წინ უნდა დაწეროთ გასაღებური სიტყვა static;
- მას შემდეგ რაც პროგრამა კლასის ელემენტს გამოაცხადებს როგორც static-ს, მან უნდა გამოაცხადოს გლობალური ცვლადი(კლასის განსაზღვრის გარეთ), რომელიც შეესაბამება კლასის ამ ერთობლივად გამოსაყენებელ ელემენტს;
- თქვენს პროგრამას შეუძლია გამოიყენოს გასაღებური სიტყვა static, რათა კლასის მეთოდი გახდეს გამოძახებადი მაშინაც კი, როცა პროგრამას შესაძლოა ჯერ კიდევ არა აქვს გამოცხადებული მოცემული კლასის რომელიმე ობიექტი.

მონაცემ ელემენტების ერთობლივი გამოყენება

გამოვაცხადოთ ელემენტები, როგორც საჯარო ან კერძო და შემდეგ ტიპის წინ დავუწეროთ გასაღებური სიტყვა static:

```
private:
static int shared_value;
```

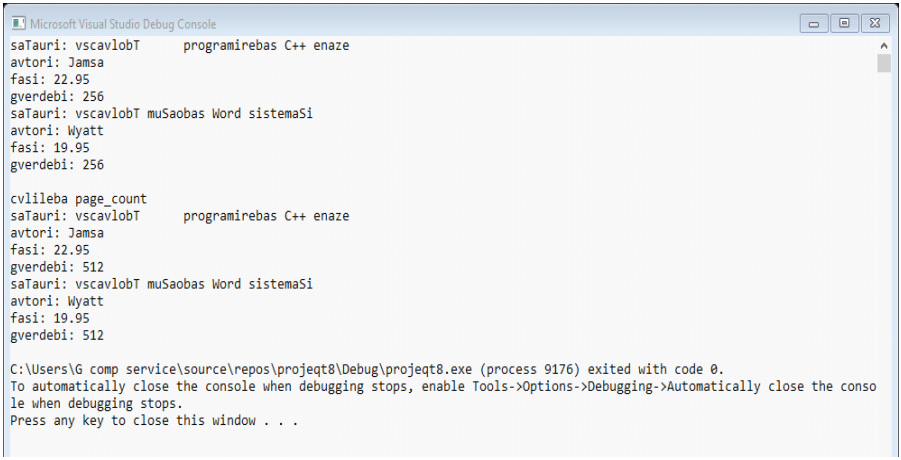
კლასის გამოცხადების შემდეგ ელემენტი უნდა გამოცხადდეს როგორც ცვლადი კლასის გარეთ, როგორც ქვემოთ არის მოცემული:

int class_name::shared_value;

განვიხილოთ პროგრამა, რომელიც განსაზღვრავს book_series კლასს, ერთობლივად გამოყენებული ელემენტია – page_count, რომელიც ერთნაირია ყველა ობიექტისათვის (წიგნისათვის). თუ პროგრამა ცვლის ამ ელემენტის მნიშვნელობას, ცვლილება მაშინვე აისახება კლასის ყველა ობიექტში:

```
#include <iostream>
#include <string>
using namespace std;
class book_series
{
public:
  book_series(string, string, float);
  void show_book(void);
  void set_pages(int);
private:
  static int page_count;
  string title;
string author;
  float price;
};
int book_series::page_count;
void book_series::set_pages(int pages)
{
  page_count = pages;
}
book_series::book_series(string title, string author,
  float price)
{
book_series::title=title;
book_series::author=author;
```

```
  book_series::price = price;
}
void book_series::show_book(void)
{
  cout << "saTauri: " << title << std::endl;
  cout << "avtori: " << author << std::endl;
  cout << "fasi: " << price << std::endl;
  cout << "gverdebi: " << page_count <<std::endl;
}

int main()
{
  system("color F0");
  book_series programming("vscavlobT       programirebas
C++ enaze", "Jamsa", 22.95);
          book_series    word("vscavlobT    muSaobas    Word
sistemaSi", "Wyatt", 19.95);
                  word.set_pages(256);
  programming.show_book();
  word.show_book();
  cout   <<   std::endl   <<   "cvlileba   page_count   "   <<
std::endl;
  programming.set_pages(512);
  programming.show_book();
  word.show_book();

}
```



როგორც ხედავთ, კლასი *page_count* ელემენტს აცხადებს, როგორც *static int* ტიპს. კლასის განსაზღვრისთანავე პროგრამა *page_count* ელემენტს აცხადებს როგორც გლობალურ ცვლადს. როდესაც პროგრამა ცვლის page_count ელემენტს, ცვლილება მაშინვე აისახება book_series კლასის ყველა ობიექტში.

ელემენტების გამოყენება *public static* ატრიბუტებით, თუ ობიექტები არ არსებობენ

შეიძლება იყოს სიტუაციები, როდესაც პროგრამას ჯერ კიდევ არ შეუქმნია ობიექტი, მაგრამ საჭიროა ელემენტის გამოყენება. ელემენტის გამოყენებისათვის პროგრამამ უნდა გამოაცხადოს იგი როგორც *public* და *static*. მაგალითად, შემდეგი პროგრამა იყენებს *page_count* ელემენტს book_series კლასიდან, მაშინაც კი თუ ამ კლასის ობიექტები არ არსებობს:

```
#include <iostream>
#include <string>
```

```
using namespace std;
class book_series
{
public:
	static int page_count;
private:

	string title;
string author;
	float price;
};
int book_series::page_count;


int main()
{
	system("color F0");
	book_series::page_count = 256;
	cout << "page_count –is mimdinare mniSvneloba tolia " <<
book_series::page_count <<endl;


}
```

ამ შემთხვევაში, რადგან კლასი განსაზღვრავს *page_count* ელემენტს როგორც ღიას(საჯაროს), პროგრამას შეუძლია მიმართოს ამ ელემენტს მაშინაც კი, თუ *book_series* კლასის ობიექტები არ არსებობენ.

სტატიკური ფუნქცია-ელემენტების გამოყენება

თუ ჩვენ შევქმნით სტატიკურ მეთოდს, პროგრამა გამოიძახებს ამ მეთოდს მაშინაც კი თუ ობიექტები არ

არის შექმნილი. მაგალითად, თუ კლასი შეიცავს მეთოდს, რომელიც შეიძლება გამოყენებულ იქნას კლასის გარეთ მონაცემებისათვის, ეს მეთოდი შეიძლება წარმოადგინოთ სტატიკური სახით.

განვიხილოთ პროგრამა სადაც შექმნილია კლასი menu, რომელშიც საჯარო ნაწილში გამოცხადებულია ტექსტის გამოტანის text_screen მეთოდი, კლასს არა აქვს ობიექტი, მაგრამ პროგრამა ამ მეთოდს გამოიყენებს:

```
#include "pch.h"
#include <iostream>
#include <stdlib.h>
using namespace std;
class menu
{
public:
  static void text_screen(void);
  // aq SeiZleba iyos sxva meTodebic
  private:
         int number_of_menu_options;
};
void menu::text_screen(void)
{
  cout << "ekranze teqstis gamotana";
}

int main()
{
  system("color F0");

         menu::text_screen();
  }
```

```
Microsoft Visual Studio Debug Console
ekranze teqstis gamotana
C:\Users\G comp service\source\repos\projeqt8\Deb
To automatically close the console when debuggin
le when debugging stops.
Press any key to close this window . . .
```

როგორც ხედავთ, თუ კლასი შეიცავს მეთოდს,

რომელსაც თქვენ მოინდომებთ გამოიყენოთ კლასის ობიექტის გარეშე, მისი პროტოტიპის წინ განათავსეთ გასაღებური სიტყვა – static და ეს მეთოდი გამოაცხადეთ, როგორც საჯარო:

public:
static void text_screen(void);

ასეთი ფუნქციის გამოძახებისათვის პროგრამის შიგნით გამოიყენეთ გლობალური ნებართვის ოპერატორი, როგორც ქვემოთ არის ნაჩვენები:

menu::text_screen();

დასკვნა:

- როდესაც თქვენ გამოაცხადებთ კლასის ელემენტს static სახით, მაშინ ასეთი ელემენტი შეიძლება გამოყენებულ იქნას მოცემული კლასის ყველა ობიექტის მიერ;
- მას შემდეგ როცა თქვენს პროგრამაში კლასის ელემენტს გამოაცხადებთ static სახით, კლასის განსაზღვრის გარეთ უნდა გამოცხადდეს გლობალური ცვლადი, რომელიც შეესაბამება კლასის ერთობლივად გამოყენებად ელემენტს;
- თუ თქვენ აცხადებთ ელემენტს, როგორც public და static, თქვენ პროგრამას შეუძლია გამოიყენოს ასეთი ელემენტი მაშინაც კი, თუ მოცემული კლასის ობიექტები არ არსებობენ. ამ ელემენტზე მიმართვისათვის უნდა იქნას გამოყენებული გლობალური ნებართვის ოპერატორი, მაგალითად: class_name::member_name;
- თუ გამოაცხადებთ საერთო სტატიკურ ფუნქცია-ელემენტს თქვენ პროგრამას შეუძლია გამოიძახოს ეს

ფუნქცია, მაშინაც კი, თუ ამ კლასის ობიექტები არ არსებობენ. მოცემული ფუნქციის გამოსაძახებლად პროგრამამ უნდა გამოიყენოს გლობალური ნებართვის ოპერატორი, მაგალითად:

```
menu::text_screen().
```

დავალება: განსაზღვრეთ კლასი თანამშრომელი, მონაცემებით: გვარი, თანამდებობა, მუშაობის სტაჟი. ერთობლივად გამოყენებული ელემენტი იყოს ხელფასი, შექმენით ამ კლასის ორი ობიექტი. პროგრამამ გამოიტანოს მონაცემები ორივე ობიექტისათვის. შემდეგ პროგრამამ შეცვალოს საერთო ელემენტის მნიშვნელობა და კვლავ გამოიტანოს მონაცემები. გააანალიზეთ პროგრამის მუშაობის შედეგები. პროგრამაში დაამატეთ სტატიკური ფუნქცია და გამოიძახეთ მთავარი ფუნქციიდან.

ლაბორატორიული სამუშაო 8
თემა: მემკვიდრეობითობა. მარტივი მემკვიდრეობითობის მაგალითები

ობიექტზე ორიენტირებული დაპროგრამების მიზანი მდგომარეობს თქვენ მიერ შექმნილი კლასის განმეორებით გამოყენებაში, რაც ეკონომიას გაუკეთებს თქვენს დროს და ძალისხმევას. თუ თქვენ უკვე შექმენით რაიმე კლასი, შესაძლებელია იყოს სიტუაციები, რომ ახალ კლასს სჭირდებოდეს უკვე არსებული კლასის ბევრი ან სულაც ყველა თავისებურება და საჭიროა დაემატოს მხოლოდ ერთო ან რამდენიმე მონაცემ-ელემენტი ან ფუნქცია. ასეთ შემთხვევაში C++ საშუალებას იძლევა შეიქმნას ახალი ობიექტი უკვე არსებული ობიექტის მახასიათებლების გამოყენებით. სხვა სიტყვებით რომ ვთქვათ ახალი კლასი იქნება უკვე არსებული კლასის მემკვიდრე.

მოცემულ მასალაში თქვენ შეისწავლით შემდეგ ძირითად კონცეფციებს:

– თუ პროგრამა გამოიყენებს მემკვიდრეობითობას, მაშინ ახალი კლასის წარმოქმნისათვის საჭიროა საბაზო კლასი, ანუ ახალი კლასი ხდება საბაზო კლასის ელემენტების მემკვიდრე;

– წარმოებული კლასის ელემენტების ინიციალი-ზაციისათვის თქვენმა პროგრამამ უნდა გამოიძახოს საბაზო და წარმოებული კლასების კონსტრუქტორები;

– ოპერატორ წერტილის გამოყენებით პროგრამა მიმართავს საბაზო და წარმოებული კლასების ელემენტებს;

– დამატებით C++ წარმოადგენს protected(დაცულ) ელემენტებს, რომლებიც მისაწვდომია საბაზო და წარმოებული კლასებისათვის;

– სახელების კონფლიქტის თავიდან ასაცილებლად თქვენ პროგრამას შეუძლია გამოიყენოს გლობალური ნებართვის ოპერატორი, რომლის წინაც მითითებული იქნება საბაზო ან წარმოებული კლასის სახელი.

მემკვიდრეობითობა არის ობიექტზე ორიენტირებული დაპროგრამების ფუნდამენტური კონცეფცია.

დავუშვათ გვაქვს employee საბაზო კლასი:

```
class employee

{
public:
  employee(string, string, float);
  void show_employee(void);
private:
  string name;
  string position;
  float salary;
};
```

დავუშვათ პროგრამას სჭირდება manager კლასი, რომელიც ამატებს employee კლასში შემდეგ მონაცემებს:

```
float annual_bonus;
string company_car;
int stock_options;
```

ასეთ შემთხვევაში პროგრამას შეუძლია აირჩიოს ორი ვარიანტი: პირველი – პროგრამა შექმნის ახალ manager კლასს, სადაც დუბლირებული იქნება employee კლასის მრავალი ელემენტი, მეორე – პროგრამა ქმნის manager კლასს employee საბაზო კლასიდან, რითაც მცირდება პროგრამის მოცულობაც და არ მოხდება კოდის დუბლირება თქვენი პროგრამის შიგნით. ამ კლასის განსაზღვრისათვის უნდა დაწეროთ გასაღებური სიტყვა class, შემდეგ სახელი manager, ორი წერტილი და სახელი employee, როგორც ქვემოთ არის ნაჩვენები:

```
class manager : public employee {
// აქ განისაზღვრება ელემენტები};
```

გასაღებური სიტყვა public, რომელიც წინ უძღვის employee კლასს, უჩვენებს, რომ employee კლასის საერთო ელემენტები აგრეთვე საერთოა manager კლასშიც. ქვემოთ წარმოდგენილია manager კლასის შექმნის ოპერატორები:

```
class manager : public employee

{
public:
  manager(string, string, string, float, float, int);
  void show_manager(void);
private:
  float annual_bonus;
  string company_car;
  int stock_options;
};
```

საბაზო კლასის კერძო ელემენტები წარმოებული კლასისათვის მისაწვდომია მხოლოდ საბაზო კლასის ფუნქციებით.

განვიხილოთ პროგრამა:

```
#include <iostream>
#include <string>
using namespace std;
class employee
{
public:
  employee(string, string, float);
  void show_employee(void);
private:
  string name;
string position;
  float salary;
};
employee::employee(string  name,  string  position,  float
salary)
{
  employee::name= name;
  employee::position= position;
  employee::salary = salary;
}
void employee::show_employee(void)
{
  cout << "saxeli: " << name << endl;
  cout << "Tnamdeboba: " << position << endl;
  cout << "xelfasi: " << salary << endl;
}

class manager : public employee  // manager klasis Seqmna
sabazo employee klasidan
{
public:
  manager(string, string, string, float, float, int);
  void show_manager(void);
private:
  float annual_bonus;
```

```
        string company_car;
        int stock_options;
      };
      manager::manager(string name, string position, string
company_car,
        float salary, float bonus, int stock_options) :
employee(name,
              position, salary)
      {
        manager::company_car= company_car;
        manager::annual_bonus = bonus;
        manager::stock_options = stock_options;
      }
      void manager::show_manager(void)
      {
        show_employee();
        cout << "firmis manqana: " << company_car << endl;
        cout << "yovelwliuri premia:" << annual_bonus << endl;
        cout << "safondo opcia: " << stock_options << endl;
      }

      int main()
      {
        system("color F0");

              employee worker("BeriZe Noe", "programisti",
35000);
              manager boss("SaxvaZe Miriani", "vice-prezidenti
", "Lexus",
                     50000.0, 5000, 1000);
              worker.show_employee();
              boss.show_manager();
      }
```

```
Microsoft Visual Studio Debug Console
saxeli: BeriZe Noe
Tnamdeboba: programisti
xelfasi: 35000
saxeli: SaxvaZe Miriani
Tnamdeboba: vice-prezidenti
xelfasi: 50000
firmis manqana: Lexus
yovelwliuri premia:5000
safondo opcia: 1000

C:\Users\G comp service\source\repos\project9\Debug\projec
To automatically close the console when debugging stops, e
le when debugging stops.
Press any key to close this window . . .
```

ყურადღება მიაქციეთ manager კონსტრუქტორს. როდესაც ქმნით ახალ კლასს საბაზო კლასიდან, წარმოებული კლასის კონსტრუქტორმა უნდა გამოიძახოს საბაზო კლასის კონსტრუქტორი, რისთვისაც წარმოებული კლასის კონსტრუქტორის შემდეგ განათავსეთ ორი წერტილი, შემდეგ მიუთითეთ საბაზო კლასის კონსტრუქტორი მოთხოვნილი პარამეტრებით:

```
manager::manager(string name, string position, string
company_car, float salary, float bonus, int stock_options)
: employee(name, position, salary) //საბაზო კლასის
კონსტრუქტორი

{
manager::company_car= company_car;
manager::annual_bonus = bonus;
manager::stock_options = stock_options;
}
```

მიაქციეთ ყურადღება, რომ show_manager ფუნქცია იძახებს show_employee ფუნქციას, რომელიც არის employee კლასის ელემენტი. რამდენადაც manager კლასი არის employee კლასიდან წარმოებული, manager კლასს შეუძლია მიმართვა employee კლასის საჯარო ელემენტებზე ისევე როგორც ყველა ელემენტი განსაზღვრული რომ ყოფილიყო manager კლასის შიგნით.

მაგალითი. პროგრამა ქმნის library _card კლასს book კლასიდან:

```
#include <iostream>
#include<string>
using namespace std;
class book
{
public:
  book(string, string, int);
  void show_book(void);
private:
  string title;
  string author;
  int pages;
};
book::book(string title, string author, int pages)
{
  book::title= title;
  book::author= author;
  book::pages = pages;
}
void book::show_book(void)
{
  cout << "dasaxeleba: " << title << endl;
  cout << "avtori: " << author << endl;
  cout << "gverdebi: " << pages << endl;
}
class library_card : public book
{
public:
  library_card(string, string, int, string, int);
  void show_card(void);
private:
  string catalog;
  int checked_out;
};
library_card::library_card(string title, string author,
int pages, string catalog, int checked_out) : book(title,
author, pages)
{
  library_card::catalog= catalog;
  library_card::checked_out = checked_out;
}
void library_card::show_card(void)
{
```

```
        show_book();
        cout << "katalogi: " << catalog << endl;
        if (checked_out) cout << "statusi: Semowmebuli" << endl;
        else cout << "statusi: Tavisufali" << endl;
      }
      int main()
      {
         system("color F0");
          library_card   card("vscavlobT   programirebas   C++
enaze", "Jamsa", 272, "101CPP", 1);
          card.show_card();
      }
```

როგორც წინა მაგალითში, მიაქციეთ ყურადღება, რომ library _card კონსტრუქტორი იძახებს book კლასის კონსტრუქტორს, book კლასის ელემენტების ინიციალიზაციისთვის. ასევე, მიაქციეთ ყურადღება book კლასის show_book ფუნქცია-ელემენტის გამოყენებას show_card ფუნქციაში. რამდენადაც library_card კლასი მემკვიდრეობს book კლასის მეთოდებს, show_card ფუნქციას შეუძლია გამოიძახოს show_book მეთოდი

ოპერატორ წერტილის დახმარების გარეშე, თითქოს ეს მეთოდი ყოფილიყოს library _card კლასის მეთოდი.

რა არის დაცული ელემენტები

როგორც ვიცით, წარმოებულ კლასის შეუძლია მიმართოს საბაზო კლასის საერთო ელემენტებს, თითქოს ისინი განსაზღვრული იყოს წარმოებულ კლასში. მეორეს მხრივ წარმოებულ კლასს პირდაპირ არ შეუძლია მიმართოს საბაზო კლასის კერძო ელემენტებს. ამისათვის მან უნდა გამოიყენოს საბაზო კლასის ფუნქციები. საბაზო კლასის დაცული ელემენტებს უჭირავთ შუალედური მდგომარეობა საჯაროსა და კერძოს შორის. თუ ელემენტი არის დაცული, წარმოებული კლასის ობიექტები მას მიმართავენ როგორც საჯაროს. თქვენი პროგრამის დანარჩენი ნაწილისათვის დაცული ელემენტები იქნება ისევე როგორც კერძო.

ქვემოთ განსაზღვრულია book კლასი, რომელიც იყენებს protected ჭდეს, რათა book კლასიდან წარმოებულ კლასს ნება დაერთოს მიმართოს title, author და pages ელემენტებს პირდაპირ, წერტილის გამოყენებით:

```
class book

{
public:
  book(string, string, int) ;
  void show_book(void) ;
protected:
  string title;
```

```
  string author;
  int pages;
};
```

როგორც შეამჩნიეთ მემკვიდრეობითობა ამარტივებს პროგრამირებას იმ შემთხვევაში, თუ წარმოებულ კლასებს შეუძლიათ მიმართონ საბაზო კლასის ელემენტებს ოპერატორ წერტილით. ასეთ შემთხვევაში პროგრამას შეუძლია გამოიყენოს კლასის დაცული ელემენტები. წარმოებულ კლასს შეუძლია მიმართოს საბაზო კლასის დაცულ ელემენტებს პირდაპირ, ოპერატორ წერტილის გამოყენებით. მხოლოდ პროგრამის დანარჩენ ნაწილს შეუძლია მიმართოს დაცულ ელემენტებს მხოლოდ ამ კლასის ფუნქციებით.

სახელთა კონფლიქტის პრობლემის გადაწყვეტა

ერთი კლასიდან მეორე კლასის შექმნის დროს შესაძლებელია გვქონდეს სიტუაციები, როდესაც წარმოებულ კლასში ელემენტის სახელი იგივეა, რაც საბაზო კლასში. მაგალითად კლასი book და library_card იყენებს price ელემენტს. book კლასში price ელემენტს შეესაბამება წიგნის გასაყიდი ფასი 22.95. library_card კლასში price შეიძლება შეიცავდეს ბიბლიოთეკის ფასდაკლებას, მაგალითად 18.50. დავუშვათ show_card ფუნქციამ უნდა გამოიტანოს ორივე ფასი. მაშინ მან უნდა გამოიყენოს შემდეგი ოპერატორები:

```
cout << "biblioTekis fasi" << price << endl;
cout << "gasayidi fasi:" << book::price << endl;
```

დასკვნა:

- მემკვიდრეობითობა არის არსებული საბაზო კლასიდან ახალი კლასის წარმოების უნარი;
- წარმოებული კლასი არის ახალი კლასი, ხოლო საბაზო კლასი - არსებული კლასი;
- როდესაც თქვენ ქმნით სხვა კლასს საბაზო კლასიდან, წარმოებული კლასი ხდება საბაზო კლასის ელემენტების მემკვიდრე;
- საბაზოდან ახალი კლასის წარმოქმნისთვის წარმოებული კლასის განსაზღვრა დაიწყეთ class გასაღებური სიტყვით, რომელსაც მოსდევს კლასის სახელი, ორი წერტილი და საბაზო კლასის სახელი, მაგალითად class dalmatian:dog;
- წარმოებულ კლასს შეუძლია მიმართოს საბაზო კლასის საერთო ელემენტებს ისე როგორც თავისივე კლასის ელემენტებს. საბაზო კლასის კერძო მონაცემებზე წვდომისათვის წარმოებული კლასი იყენებს საბაზო კლასის ფუნქციებს;
- წარმოებული კლასის კონსტრუქტორში პროგრამამ უნდა გამოიძახოს საბაზო კლასის კონსტრუქტორი წარმოებული კლასის კონსტრუქტორის სათაურის შემდეგ ორი წერტილის, საბაზო კლასის კონსტრუქტორის სახელის და შესაბამისი პარამეტრების მითითებით;
- რათა წარმოებული კლასისათვის უზრუნველყოფილ იქნას საბაზო კლასის განსაზღვრულ ელემენტებზე პირდაპირი წვდომა და ამავე დროს ეს ელემენტები დაცული იყოს პროგრამის დანარჩენი ნაწილისთვის, C++ უზრუნველყოფს კლასის protected (დაცულ) ელემენტებს. წარმოებულ კლასს შეუძლია მიმართოს საბაზო კლასის დაცულ ელემენტებს როგორც საჯაროს. პროგრამის დანარჩენი ნაწილისთვის დაცული ელემენტები კერძოს ექვივალენტურია;

– თუ წარმოებულ და საბაზო კლასში არის ელემენტები ერთნაირი სახელებით, მაშინ წარმოებული კლასის ფუნქციის შიგნით C++ გამოიყენებს წარმოებული კლასის ელემენტებს. თუ წარმოებული კლასის ფუნქციებს სჭირდებათ საბაზო კლასის ელემენტებზე მიმართვა, უნდა გამოიყენოთ გლობალური ნებართვის ოპერატორი, მაგალითად base_class::member

დავალება: შექმენით კლასი ბაკალავრი მონაცემებით: გვარი, სახელი, ფაკულტეტი, დაგროვებული კრედიტების რაოდენობა. კლასისთვის განსაზღვრეთ ფუნქცია ამ მონაცემების გამოსატანად. შექმენით ბაკალავრი კლასის მემკვიდრე კლასი: მაგისტრანტი, რომელსაც დამატებით ექნება ორი მონაცემი: ბაკალავრიატის კვალიფიკაცია და სწავლების წელი. ამავე კლასისთვის განსაზღვრეთ ფუნქცია მონაცემების გამოსატანად. გამოაცხადეთ ამ კლასების თითო ობიექტი და გამოიტანეთ ინფორმაცია სტუდენტების შესახებ.

ლაბორატორიული სამუშაო 9
თემა: მრავლობითი მემკვიდრეობითობა

C++ ენაში შესაძლებელია შეიქმნას კლასი რამდენიმე საბაზო კლასისგან. როსდესაც კლასი მემკვიდრეობას იღებს რამდენიმე კლასისგან უნდა გამოვიყენოთ მრავლობითი მემკვიდრეობითობა.

მოცემულ მასალაში შეისწავლით შემდეგ ძირითად კონცეფციებს:

- თუ თქვენ ქმნით კლასს რამდენიმე საბაზო კლასისგან, მაშინ მიიღებთ მრავლობითი მემკვიდრეობის უპირატესობას;
- მრავლობითი მემკვიდრეობის დროს წარმოებული კლასი მიიღებს ორი ან მეტი კლასის ატრიბუტებს;
- მრავლობითი მემკვიდრეობითობის გამოყენებისას კლასის შექმნისათვის წარმოებული კლასის კონსტრუქტორმა უნდა გამოიძახოს ყველა საბაზო კლასის კონსტრუქტორი;
- წარმოებული კლასიდან ახალი კლასის შექმნისას თქვენ ქმნით მემკვიდრეობითობის იერარქიას(კლასთა იერარქიას).

მრავლობითი მემკვიდრეობა ობიქტზე ორიენტირებული დაპროგრამების მძლავრი ინსტრუმენტია.

მარტივი მაგალითი:

დავუშვათ გვაქვს კლასი computer_screen:

```
class computer_screen

{
public:
```

```
  computer_screen(string, long, int, int);
  void show_screen(void);
private:
  string type ;
  long colors;
  int x_resolution;
  int y_resolution;
};
```

დავუშვათ გვაქვს, აგრეთვე, კლასი mother_board:

```
class mother_board

{
public:
  mother_board(int, int, int);
  void show_mother_board(void);
private:
  int processor;
  int speed;
  int RAM;
};
```

ამ ორი კლასის გამოყენებით შეიძლება ავაგოთ კლასი computer, რაც ნაჩვენებია ქვემოთ:

```
class computer : public computer_screen, public mother_board

{
public:
  computer(string, int, float, string, long, int, int, int, int, int);
  void show_computer(void);
private:
string  name;
```

```
  int hard_disk;
  float floppy;
};
```

როგორც ხედავთ ეს კლასი უჩვენებს თავის საბაზო კლასებს ორი წერტილის შემდეგ, რომელიც მოსდევს computer კლასის სახელს:

```
class computer : public computer_screen, public
mother_board//საბაზო კლასები
```

ქვემოთ წარმოდგენილია პროგრამა სრული სახით:

```
#include <iostream>
#include<string>
using namespace std;
class computer_screen
{
public:
  computer_screen(string, long, int, int);
  void show_screen(void);
private:
  string type;
  long colors;
  int x_resolution;
  int y_resolution;
};

computer_screen::computer_screen(string type, long
colors, int
  x_res, int y_res)
{
  computer_screen::type = type;
  computer_screen::colors = colors;
  computer_screen::x_resolution = x_res;
  computer_screen::y_resolution = y_res;
}
```

```
      void computer_screen::show_screen(void)
      {
        cout << "ekranis tipi: " << type << endl;
        cout << "ferebi: " << colors << endl;
        cout << "dashveba: " << x_resolution << " X " <<
y_resolution
             << endl;
      }
      class mother_board
      {
      public:
        mother_board(int, int, int);
        void show_mother_board(void);
      private:
        int processor;
        int speed;
        int ram;
      };
      mother_board::mother_board(int processor, int speed, int
ram)
      {
        mother_board::processor = processor;
        mother_board::speed = speed;
        mother_board::ram = ram;
      }
      void mother_board::show_mother_board(void)
      {
        cout << "procesori: " << processor << endl;
        cout << "sixSire: " << speed << "MH" << endl;
        cout << "operatiuli: " << ram << " MB" << endl;
      }
      class computer : public computer_screen, public
mother_board
      {
      public:
        computer(string, int, float, string, long, int, int,
int, int, int);
        void show_computer(void);
      private:
```

```
        string name;
        int hard_disk;
        float floppy;
      };
      computer::computer(string name, int hard_disk, float
floppy, string screen, long colors, int x_res, int y_res, int
processor, int speed, int
        ram) : computer_screen(screen, colors, x_res, y_res),
        mother_board(processor, speed, ram)
      {
        computer::name= name;
        computer::hard_disk = hard_disk;
        computer::floppy = floppy;
      }
      void computer::show_computer(void)
      {
        cout << "tipi: " << name << endl;
        cout << "myari diski: " << hard_disk << "GB" << endl;
        cout << "moqnili diski: " << floppy << "MB" << endl;
        show_mother_board();
        show_screen();
      }
      int main()
      {
         system("color F0");
         computer my_pc("Compaq", 212, 1.44, "SVGA", 16000000,
640, 480, 486, 66, 8);
         my_pc.show_computer();
        }
```

```
ipi: Compaq
yari diski: 212GB
oqnili diski: 1.44??
rocesori: 486
ixSire: 66?H
peratiuli: 8 ??
kranis tipi: SVGA
erebi: 16000000
ashveba: 640 X 480
```

თუ თქვენ გააანალიზებთ computer კლასის კონსტრუქტორს აღმოაჩენთ, რომ იგი იძახებს mother_board და computer_screen კლასების კონსტრუქტორებს, როგორც ნაჩვენებია ქვემოთ:

computer::computer(string name, int hard_disk, float floppy, string screen, long colors, int x_res, int y_res, int processor, int speed, int RAM) : computer_screen(screen, colors, x_res, y_res), mother_board(processor, speed, RAM)

კლასთა იერარქიის აგება

ზოგჯერ საჭიროა შეიქმნას კლასი, იმ კლასისგან, რომელიც თავის მხრივ წარმოებულია რამდენიმე საბაზო კლასისგან. მაგალითად ჩვენ გვჭირდება computer კლასი გამოვიყენოთ საბაზოდ და შევქმნათ ახალი კლასი workstation:

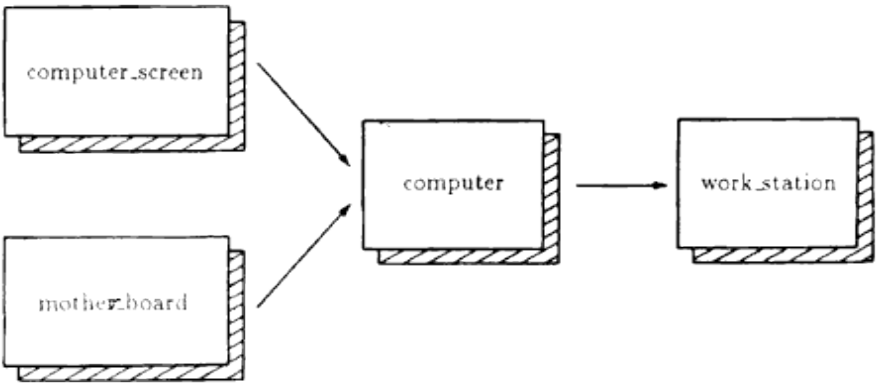
```
class work_station : public computer

{
public:
  work_station (string operating_system, string name, int hard_disk, float floppy, string screen, long colors, int x_res, int y_res, int processor, int speed, int RAM);
```

```
  void show_work_station(void);
private:
  string operating_system;
};
```

work_station კლასის კონსტრუქტორი უბრალოდ იძახებს computer კლასის კონსტრუქტორს, რომელიც თავის მხრივ იძახებს computer_screen და mother_board კონსტრუქტორებს:

```
work_station::work_station ( string operating_system, string name, int hard_disk, float floppy, string screen, long colors, int  x_res, int y_res, int processor, int speed, int RAM) : computer (name, hard_disk, floppy, screen, colors, x_res, y_res,  processor, speed, RAM)

{
work_station::operating_system= operating_system;
}
```

მოცემულ შემთხვევაში computer კლასი გამოდის საბაზო კლასის როლში. მხოლოდ, როგორც იცით იგი იყო წარმოებული computer_screen და mother_board კლასებიდან. შესაბამისად work_station კლასი მემკვიდრეობად მიიღებს სამივე კლასის თვისებებს. ახალი კლასების „დაბადებას“ მივყავართ კლასების იერარქიამდე, როგორც ნაჩვენებია ნახაზზე:



დასკვნა:

- მრავლობითი მემკვიდრეობითობა არის უნარი წარმოებულ კლასს გადაეცეს რამდენიმე საბაზო კლასის მახასიათებლები;
- ახალი მემკვიდრე კლასის შექმნისას ახალი კლასის სახელის შემდეგ უნდა დაისვას ორი წერტილი და მიეთითოს საბაზო კლასების სახელები, რომელიც გამოყოფილია მძიმეებით, მაგალითად: class cabbit : public cat, public rabbit;
- წარმოებული კლასის კონსტრუქტორის განსაზღვრისას უნდა გამოიძახოთ ყველა საბაზო კლასის კონსტრუქტორები, საჭირო პარამეტრების გადაცემით;
  – წარმოებული კლასიდან შეიძლება ახალი კლასის შექმნა. ამგვარად ჩვენი პროგრამა შექმნის კლასთა იერარქიას.

დავალება: შექმენით კლასი ბაკალავრი მონაცემებით: გვარი, ფაკულტეტი, დაგროვებული კრედიტების რაოდენობა. შექმენით მისი მემკვიდრე კლასი: მაგისტრანტი, რომელსაც დამატებით ექნება ორი მონაცემი: ბაკალავრიატის კვალიფიკაცია და სწავლების წელი. შექმენით მაგისტრანტის მემკვიდრე კლასი დოქტორანტი, რომელსაც დამატებით ექნება ორი მონაცემი: მაგისტრანტის კვალიფიკაცია და გამოქვეყნებული სტატიების რაოდენობა. შესაბამისად, კლასებში შექმენით ფუნქციები მონაცემების გამოსატანად. შექმენით ამ კლასების თითო ობიექტი და გამოიტანეთ ინფორმაცია სტუდენტების შესახებ.

ლაბორატორიული სამუშაო 10
თემა: საჯარო მემკვიდრეობითობის მაგალითები. კერძო და დაცული მემკვიდრეობითობა

როგორც წინა მასალაში განვიხილეთ არსებული კლასის საფუძველზე შექმნილ კლასს ეწოდება წარმოებული ან მემკვიდრე კლასი, ხოლო იმ კლასს, რომლის საფუძველზე შეიქმნა ახალი, უწოდებენ – საბაზოს ან წინაპარ კლასს. ყოველი მემკვიდრე კლასი თვითონ შეიძლება გახდეს საბაზო კლასი მომავალი მემკვიდრე კლასებისთვის. მარტივი მემკვიდრეობითობის დროს წარმოებული კლასი მიიღება მხოლოდ ერთი საბაზო კლასის საფუძველზე. რთული მემკვიდრეობითობის შემთხვევაში წარმოებული კლასი იქმნება რამდენიმე კლასის საფუძველზე. მემკვიდრეობითობის მექანიზმის გამოყენებით აიგება რთული იერარქიული სტრუქტურები.

ის ფაქტი, რომ Circle კლასი წარმოიშვა Point კლასის საფუძველზე აღინიშნება შემდეგი სახით:

```
class Circle : public Point{ ……..};
```

ამას ეწოდება ღია მემკვიდრეობითობა (Public inheritance).

protected (დაცული) წვდომის რეჟიმი შემოტანილია მემკვიდრეობითობასთან დაკავშირებით. საჯარო მემკვიდრეობითობის დროს საბაზო კლასის საჯარო და დაცულ ელემენტებს მემკვიდრე კლასის ობიექტები მიმართავენ როგორც საჯაროს, ხოლო საბაზო კლასის დაცული ელემენტები პროგრამის სხვა ნაწილისათვის არის როგორც კერძო ელემენტები. საბაზო კლასის

კერძო ელემენტებზე მემკვიდრე კლასს პირდაპირი წვდომა არა აქვს, მათზე მიმართვა შეუძლია მხოლოდ საბაზო კლასის ფუნქციებს.

ზემოაღნიშნული საკითხების განმტკიცებისათვის განვიხილოთ მაგალითი:

```
#include <iostream>
#include<string>
using namespace std;
class base { //sabazo klasi
public:
  int get_i();
  int get_j();
  void set_ij(int, int);
  void show();
private:
  int i, j;
};
class derived : public base // warmoebuli klasi
{
public:
  void make_k();
  void show();
private:
  int k;
};
//realizebis nawili
int base::get_i() { return i; }
int base::get_j() { return j; }
void base::set_ij(int a, int b) { i = a; j = b; }
void base::show() { cout << "i = " << i << " j = " << j;
}
void derived::make_k() { k = get_i()*get_j(); }
void derived::show() {
  base::show();
  cout << "k = " << k << endl;
}
```

```
int main()
{
   system("color F0");
   base b;
   b.set_ij(2, 3);
   cout << "Object b of base class : ";
   b.show();
   derived d;
   d.set_ij(5, 6); d.make_k();
   cout << "\n\nObject d of derived class :"; d.show();
}
```

```
ject b of base class : i = 2 j = 3

ject d of derived class :i = 5 j = 6k = 30

\users\g comp service\source\repos\project klasebi\Debug\project klasebi
```

derived კლასი არის base კლასის მემკვიდრე და მიიღება ღია(საჯარო) მემკვიდრეობითობით:

class derived : public base {…..};

base კლასის ყველა ელემენტი მემკვიდრეობით გადაეცემა derived კლასს. ეს ნიშნავს, რომ derived კლასის ღია ინტერფეისი შეიცავს base კლასის ღია ფუნქცია-წევრებს: get_i, get_j, set_ij და show (ანუ ფუნქციების გამოყენება შეუძლიათ derived კლასის ობიექტებს და საკუთარ ფუნქცია-წევრებს: make_k და show. გარდა ამისა derived კლასის ობიექტი ხასიათდება მონაცემებით i, j და k.

base კლასის მონაცემები i და j აღწერილია როგორც კლასის private (კერძო) წევრები. ამიტომ წარმოებულ derived კლასში მათზე პირდაპირი წვდომა აკრძალულია, ისევე როგორც პროგრამის დანარჩენ

ნაწილში. derived კლასში ამ წევრებზე წვდომისათვის გამოყენებულია base კლასის ფუნქციები: get_i და get_j:

```
void derived::make_k(){k=get_i()*get_j();}
```

თუ იგივე მაგალითში base კლასის i და j მონაცემებზე წვდომის რეჟიმს შევცვლით დაცულით (protected), მაშინ წარმოებულ კლასს i და j მონაცემებზე ექნება პირდაპირი წვდომა:

```
void derived ::make_k() {k=i*j;}
```

ამავე დროს პროგრამის დანარჩენი ფუნქციებისთვის i და j ისევ დახურული მონაცემებია ანუ მათზე პირდაპირი წვდომა არა აქვთ.

პროგრამა წარმოვადგინოთ ამ ცვლილების გათვალისწინებით:

```
#include <iostream>
#include<string>
using namespace std;
 class base {
 public:
        int get_i();
        int get_j();
        void set_ij(int, int);
        void show();
 protected:
        int i, j;
 };
 class derived : public base
 {
 public:
        void make_k();
        void show();
 private:
        int k;
 };
 int base::get_i() {
```

```
            return i;
        }
        int base::get_j() { return j; }
        void base::set_ij(int a, int b) { i = a; j = b; }
        void base::show() { cout << "i = " << i << " j = " << j;
        }
        void  derived::make_k()  {  k  =  i  *  j;  }  //pirdapiri
mimarTva      i da j elementebze
              void derived::show() {
              base::show();
              cout << "k = " << k << endl;
        }

      int main()
      {
         system("color F0");
         derived d;
         d.set_ij(3, 7); d.make_k();
         cout << "Object d of derived class :";
         d.show();
      }
```

```
Object d of derived class :i = 3 j = 7k = 21

c:\users\g comp service\source\repos\project klasebi\Debug\project klasebi.exe
To automatically close the console when debugging stops, enable Tools->Options-
le when debugging stops.
```

მივაქციოთ ყურადღება იმ ფაქტსაც, რომ base კლასის set_ij ფუნქციის გამოყენება შეუძლია როგორც ამ კლასის ობიექტს, ასევე derived კლასის d ობიექტს. საინტერესოა სახელების კონფლიქტის გადაწყვეტის საკითხი: ორივე derived და base კლასში გამოცხადებულია ერთი და იგივე სახელის მქონე show ფუნქციები. რეალიზაციის ნაწილში მათი განმარტებებია:

void base::show() {cout<<"i="<<i<<" j="<<j;}

და

void derived::show() {base::show();

cout<<"k="<<k<<endl;}

derived კლასის show ფუნქცია ბეჭდავს ამ კლასის ყველა მონაცემს. ამასთან i და j მონაცემების დასაბეჭდად გამოიძახება base კლასის show ფუნქცია - base::show(); კლასებში სახელთა კონფლიქტის გადაჭრის მიზნით აუცილებელია იმის მითითება, რომ გამოძახებული show ფუნქცია არის base კლასის ხედვის არეში - base::

დასკვნა:

- public მემკვიდრეობითობის შემთხვევაში საბაზო კლასის public ელემენტებზე წვდომა აქვს პროგრამის ყველა ფუნქციას. საბაზო კლასის private წევრებზე წვდომა აქვთ მხოლოდ საბაზო კლასის ფუნქცია-წევრებს და მეგობარ-ფუნქციებს, რომელსაც შემდგომ მასალაში განვიხილავთ.
- protected (დაცული) წვდომის დონე წარმოადგენს წვდომის შუალედურ დონეს public და private წვდომის დონეებს შორის. საბაზო კლასის protected ელემენტებზე წვდომა აქვთ მხოლოდ: საბაზო კლასის ფუნქციებს და მეგობარ ფუნქციებს; წარმოებული კლასის ფუნქციებს და მეგობარ-ფუნქციებს.
- წარმოებული კლასის ფუნქციებს შეუძლიათ საბაზო კლასის public და protected ელემენტებზე მიმართვა მათი სახელების მითითებით. ამასთან უნდა გვახსოვდეს, რომ protected წვდომის მონაცემები არღვევენ საბაზო კლასის ინკაფსულაციას: საბაზო

კლასის დაცული ელემენტების შეცვლამ შეიძლება გამოიწვიოს ყველა წარმოებული კლასის მოდიფიცირების აუცილებლობა.

საბაზო და წარმოებული კლასის ობიექტებს აქვთ საერთო ნაწილი – საბაზო კლასის ატრიბუტები (მონაცემები) და ფუნქციები. ამიტომ public მემკვიდრეობითობით მიღებული წარმოებული კლასის ობიექტი შეიძლება განვიხილოთ, როგორც საბაზო კლასის ობიექტიც, მაგრამ საბაზო კლასის ობიექტი წარმოებული კლასის ობიექტს არ წარმოადგენს.

წარმოებული კლასის შექმნისას მემკვიდრეობითობის ტიპი შეიძლება იყოს როგორც public, ასევე protected და private.

მნიშვნელოვანია, რომ მემკვიდრეობითობის ტიპის მიუხედავად ვერცერთი მემკვიდრე ვერ მიიღებს წვდომას საბაზო კლასის private მონაცემებთან.

protected მემკვიდრეობითობის დროს საბაზო კლასის:

- public ელემენტები წარმოებულ კლასში ხდებიან protected – მათზე პირდაპირი წვდომა აქვთ მხოლოდ წარმოებული კლასის ფუნქცია-წევრებს და მეგობარ ფუნქციებს(შემდეგში განვიხილავთ);
- protected ელემენტები მემკვიდრე კლასშიც რჩებიან protected;
- private ელემენტებზე პირდაპირი წვდომა წარმოებულ კლასში არ არის. მათზე წვდომა ხორციელდება საბაზო კლასის public ან protected ფუნქციების საშუალებით.

private მემკვიდრეობითობის დროს საბაზო კლასის:

- public ელემენტები წარმოებულ კლასში ხდებიან protected – მათზე პირდაპირი წვდომა აქვთ მხოლოდ წარმოებული კლასის ფუნქცია-წევრებს და მეგობარ ფუნქციებს;
- protected ელემენტები წარმოებულ კლასში ხდებიან private;
- private ელემენტები წარმოებულ კლასში „არ ჩანან“. მათზე წვდომა ხერხდება საბაზო კლასის public ან protected ფუნქციების გამოყენებით.

დავალება: შექმენით კლასი მართკუთხედი, მონაცემებით: სიგრძე და სიგანე. მასში განსაზღვრეთ ფუნქცია, რომელიც გამოითვლის მართკუთხედის ფართობს. შექმენით მისი მემკვიდრე კლასი მართკუთხა პარალელეპიპედი, რომელსაც მონაცემებში დაემატება სიმაღლე, ხოლო ფუნქციაში გამოითვლება მართკუთხა პარალელეპიპედის მოცულობა. შექმენით ამ კლასების თითო ობიექტი. გამოიძახეთ ობიექტებისთვის შესაბამისი ფუნქციები. შედეგები გამოიტანეთ ეკრანზე.

ლაბორატორიული სამუშაო 11
თემა: მეგობარი კლასები და მეგობარი ფუნქციები

როგორც უკვე იცით, პროგრამას შეუძლია მიმართოს კლასის კერძო ელემენტებს მხოლოდ ამავე კლასის ფუნქცია-ელემენტების მეშვეობით. ყველა სიტუაციაში სადაც ეს შესაძლებელია საჯაროს ნაცვლად კერძო ელემენტების გამოყენებით თქვენ ამცირებთ პროგრამის შესაძლებლობას დააზიანოს კლასის ელემენტების მნიშვნელობა. მაგრამ ზოგჯერ თქვენ გჭირდებათ მწარმოებლურობის გაზრდა იმის ხარჯზე, რომ კლასს ნება დართოთ პირდაპირ მიმართოს სხვა კლასის კერძო ელემენტებს. ასეთ შემთხვევაში მცირდება დაყოვნებები. მსგავს სიტუაციებში C++ საშუალებას იძლევა კლასი განისაზღვროს მეგობრის(friend) რანგში სხვა კლასისთვის და მეგობარ-კლასს ნებას რთავს მიმართოს კერძო ელემენტებს. მოცემულ მასალაში მოცემულია თუ როგორ გაიგებს თქვენი პროგრამა, რომ ორი კლასი არის მეგობარი.

მასალაში შეისწავლით შემდეგ ძირითად კონცეფციებს:

- გასაღებური friend სიტყვის გამოყენებით კლასი იტყობინება, რომელია მისი მეგობარი ანუ იმას, რომ სხვა კლასები შეძლებენ პირდაპირ მიმართონ მის კერძო ელემენტებს;
- კლასის კერძო ელემენტები იცავენ კლასის მონაცემებს, შესაბამისად, თქვენ უნდა შეზღუდოთ კლასი-მეგობრების წრე მხოლოდ იმ კლასებით,

რომელთაც ნამდვილად სჭირდებათ პირდაპირი წვდომა საძიებო კლასის კერძო ელემენტებზე;

– C++ საშუალებას იძლევა შეიზღუდოს მეგობრული წვდომა ფუნქციების განსაზღვრული ნაკრებით.

კერძო ელემენტები კლასის დაცვის და შეცდომების შემცირების იძლევა საშუალებას. ამდენად, თქვენ უნდა შეზღუდოთ მეგობარ-კლასების გამოყენება იმდენად, რამდენადაც იგი შესაძლებელია.

კლასის მეგობრების განსაზღვრა

იმის მისათითებლად, რომ ერთი კლასი არის მეორეს მეგობარი, ამ სხვა კლასის განსაზღვრის შიგნით უნდა განათავსოთ გასაღებური სიტყვა friend და შესაბამისი მეგობარი-კლასის სახელი. ქვემოთ მოყვანილ მაგალითში book კლასი არის librarian კლასის მეგობარი. ამიტომ librarian კლასის ობიექტებს შეუძლიათ პირდაპირ მიმართონ book კლასის კერძო ელემენტებს ოპერატორ წერტილის გამოყენებით:

```
 class book
{
public:
  book (string, string, string);
  void show_book(void);
  friend librarian;
private:
  string title ;
  string author;
  string catalog;
};
```

როგორც ხედავთ მეგობრის მისათითებლად საჭიროა მხოლოდ ერთი ოპერატორი კლასის განსაზღვრაში. ქვემოთ წარმოდგენილ პროგრამაში librarian კლასი არის book კლასის მეგობარი. პროგრამა გამოიყენებს librarian კლასის change_catalog ფუნქციას განსაზღვრული წიგნის კატალოგის ბარათის ნომრის შესაცვლელად:

```
#include <iostream>
#include<string>
using namespace std;
class librarian;
class book
{
public:
  book(string, string, string);
  void show_book(void);
  friend librarian;
private:
  string title;
  string author;
  string catalog;
};
book::book(string title,string author, string catalog)
{
  book::title= title;
book::author= author;
  book::catalog= catalog;
}
void book::show_book(void)
{
  cout << "dasaxeleba: " << title << endl;
  cout << "avtori: " << author << endl;
  cout << "katalogi: " << catalog << endl;
}
class librarian
{
public:
  void change_catalog(book *,string);
  string get_catalog(book);
```

```
};
void librarian::change_catalog(book *this_book, string
new_catalog)
{
  this_book->catalog, new_catalog;
}
string librarian::get_catalog(book this_book)
{
  string catalog;
  catalog= this_book.catalog;
  return(catalog);
}

int main()
{
  book programming("vscavlobT daprogramebas C++ enaze",
         "Jamsa", "P101");
  librarian library;
  programming.show_book();
  library.change_catalog(&programming,    "leqciebi    C++
101");
  programming.show_book();
}
```

```
lasaxeleba: vscavlobT daprogramebas C++ enaze
avtori: Jamsa
catalogi: P101
lasaxeleba: vscavlobT daprogramebas C++ enaze
avtori: Jamsa
catalogi: P101

:\Users\G comp service\source\repos\project klasebi\Debug\project kla
o automatically close the console when debugging stops, enable Tools-
```

როგორც ხედავთ, პროგრამა გადასცემს book ობიექტს librarian კლასის change_catalog ფუნქციაში მისამართით. რამდენადაც ეს ფუნქცია ცვლის book კლასის ელემენტს, პროგრამამ უნდა გადასცეს პარამეტრი მისამართით, ხოლო შემდეგ გამოიყენოს მიმთითებელი ამ კლასის ელემენტზე მიმართვისათვის. ჩაატარეთ ექსპერიმენტი: შეეცადეთ წაშალოთ friend

ოპერატორი book კლასის განსაზღვრიდან. რამდენადაც librarian კლასს უკვე აღარ აქვს წვდომა book კლასის კერძო ელემენტებზე, კომპილატორი მოგვცემს სინტაქსური შეცდომის შესახებ შეტყობინებას book კლასის კერძო მონაცემებზე ყოველი მიმართვისას.
ქვემოთ წარმოდგენილია კლასის მეგობარად გამოცხადების ზოდაგი სახე:

class abbott

{
public:
friend costello;
// საერთო ელემენტები
private:
// კერძო ელემენტები};

რით განსხვავდება მეგობრები დაცული(protected) ელემენტებისგან

წარმოებულ კლასს შეუძლია პირდაპირ მიმართოს საბაზო კლასის დაცულ ელემენტებს, ოპერატორ წერტილის გამოყენებით. გახსოვდეთ, რომ კლასის დაცულ ელემენტებს შეუძლია მიმართოს მხოლოდ იმ კლასებმა, რომლებიც არიან წარმოებული ამ საბაზო კლასიდან ანუ რომლებიც არიან საბაზო კლასის მემკვიდრეები. კლასის დაცული ელემენტები პროგრამის სხვა დანარჩენი ნაწილისთვის არიან როგორც კერძოები. მეგობარი-კლასები ერთმანეთთან არ არიან დაკავშირებულნი მემკვიდრეობითობით. მემკვიდრული კავშირის არქონის შემთხვევაში სხვა კლასის კერძო ელემენტებზე წვდომის მიღების ერთადერთი ხერხი არის

კომპილატორის ინფორმირება, რომ მოცემულ კლასი არის მეგობარი.

მეგობრების რაოდენობის შეზღუდვა

დავუშვათ, წინა პროგრამაში წარმოდგენილი librarian კლასი შეიცავს მრავალ განსხვავებულ ფუნქციას. დავუშვათ, მხოლოდ change_catalog და get_catalog ფუნქციებს სჭირდებათ წვდომა book კლასის კერძო ელემენტებზე. book კლასის განსაზღვრაში ჩვენ შეგვიძლია შევზღუდოთ წვდომა კერძო ელემენტებზე მხოლოდ ამ ორი ფუნქციით, როგორც ქვემოთ არის ნაჩვენები:

```
class book
{
public:
  book(string, string, string);
  void show_book(void);
  friend string  librarian::get_catalog(book);
  friend void librarian: :change_catalog( book *, string);
private:
  string title;
  string author;
  string catalog;
};
```

როგორც ხედავთ, friend ოპერატორები შეიცავენ იმ მეგობარი ფუნქციის პროტოტიპებს, რომელთაც შეუძლიათ პირდაპირ მიმართონ კერძო ელემენტებს.

მეგობარი-ფუნქციები

თუ პროგრამა იყენებს მეგობრებს კლასის კერძო მონაცემებთან წვდომისთვის, თქვენ შეგიძლიათ შეზღუდოთ მეგობარი კლასის ფუნქცია-ელემენტების

რაოდენობა, რომელთაც შეუძლიათ პირდაპირ მიმართონ კერძო ელემენტებს. მეგობარი-ფუნქციის გამოცხადებისათვის მიუთითეთ friend გასაღებური სიტყვა, რომელსაც მოჰყვება სრული პროტოტიპი, როგორც ქვემოთ არის ნაჩვენები:

public:
friend class_name::function_name(parameter types);

მხოლოდ მეგობარ ფუნქცია-ელემენტებს შეუძლიათ პირდაპირ მიმართონ კლასის კერძო ელემენტებს, ოპერატორ წერტილის გამოყენებით.

ერთი კლასიდან მეორეზე მიმართვისას კლასების განსაზღვრის მიმდევრობა უნდა იყოს დაცული, წინააღმდეგ შემთხვევაში პროგრამა მოგვცემს შეტყობინებას სინტაქსური შეცდომის შესახებ. მოცემულ შემთხვევაში book კლასის განსაზღვრა გამოიყენებს ფუნქციების პროტოტიპებს, რომლებიც განსაზღვრულია librarian კლასში. შესაბამისად, librarian კლასის განსაზღვრა წინ უნდა უძღოდეს book კლასის განსაზღვრას. თუ გააანალიზებთ librarian კლასს, აღმოაჩენთ, რომ იგი მიმართავს book კლასს:

```
class librarian
{
public:
  void change_catalog(book *, string);
  string get_catalog(book);
};
```

რამდენადაც თქვენ არ შეგიძლიათ book კლასის განსაზღვრა განათავსოთ librarian კლასის განსაზღვრის

წინ, C++ საშუალებას იძლევა გამოცხადდეს book კლასი ანუ შეატყობინოს კომპილატორს, რომ ასეთი კლასი არის, ხოლო მოგვიანებით განისაზღვროს იგი. კლასი გამოვაცხადოთ შემდეგი სახით:

class class_name;

მაგალითად:

class book; // კლასის გამოცხადება

შემდეგი პროგრამა გამოიყენებს მეგობარ ფუნქციებს librarian კლასის წვდომის შეზღუდვისათვის book კლასის კერძო მონაცემებზე. ყურადღება მიაქციეთ კლასების განსაზღვრის მიმდევრობას:

```
#include <iostream>
#include<string>
using namespace std;
class book;
class librarian
{
public:
  void change_catalog(book *,string);
  string get_catalog(book);
};
class book
{
public:
  book(string, string, string);
  void show_book(void);
  friend string librarian::get_catalog(book);
  friend void librarian::change_catalog(book *, string);
private:
  string title;
     string author;
  string catalog;
};
book::book(string title, string author, string catalog)
{
  book::title= title;
```

```
  book::author= author;
  book::catalog= catalog;
}
void book::show_book(void)
{
  cout << "dasaxeleba: " << title << endl;
  cout << "avtori: " << author << endl;
  cout << "katalogi: " << catalog << endl;
}
void  librarian::change_catalog(book  *this_book,  string
new_catalog)
{
  this_book->catalog=new_catalog;
}
string librarian::get_catalog(book this_book)
{
  string catalog;
  catalog  = this_book.catalog;
  return(catalog);
}


int main()
{
  system("color F0");
  book     programming("vswavlobT     programirebas     C++
enaze","Jamsa", "P101");
  librarian library;
  programming.show_book();
  library.change_catalog(&programming, "leqcia C++ 101");
  programming.show_book();
}
```

```
dasaxeleba: vswavlobT programirebas C++ enaze
avtori: Jamsa
katalogi: P101
dasaxeleba: vswavlobT programirebas C++ enaze
avtori: Jamsa
katalogi: leqcia C++ 101
```

როგორც ხედავთ, პროგრამა თავიდან იყენებს განცხადებას, რათა შეატყობინოს კომპილატორს, რომ book

კლასი განსაზღვრული იქნება შემდგომ. რადგან book კლასზე განაცხადი გაკეთებულია, librarian კლასის განსაზღვრა შეიძლება გადაიგზავნოს book კლასზე, რომელიც პროგრამაში ჯერ კიდევ არ არის განსაზღვრული.

დასკვნა:

– C++ -ში მეგობრების გამოყენება სხვა კლასის კერძო ელემენტებზე პირდაპირი მიმართვის საშუალებას იძლევა, ოპერატორ წერტილის გამოყენებით;

– ერთი კლასის მეორე კლასის მეგობრად გამოცხადებისათვის, მეორე კლასის გამოცხადებაში უნდა უჩვენოთ friend გასაღებური სიტყვა, რომლის შემდეგაც იწერება პირველი კლასის სახელი;

– კლასის სხვა კლასთან მიმართებით მეგობრად გამოცხადების შემდეგ, მეგობარი-კლასის ყველა ფუნქცია-ელემენტს შეუძლია მიმართოს ამ სხვა კლასის კერძო ელემენტებს;

– მეგობარი მეთოდების შეზღუდვის მიზნით, C++ საშუალებას იძლევა მითითებულ იქნა მეგობარი ფუნქციები. მეგობარი ფუნქციის გამოცხადებისათვის უნდა მიეთითოს friend გასაღებური სიტყვა, რომლის შემდეგაც იწერება ფუნქციის პროტოტიპი;

– მეგობარი ფუნქციების გამოცხადებისას უნდა დაიცვათ კლასების განსაზღვრის მიმდევრობა. თუ საჭიროა კომპილატორზე შეტყობინება, რომ რომ იდენტიფიკატორი წარმოადგენს კლასის სახელს, რომელსაც პროგრამა განსაზღვრავს მოგვიანებით,

თქვენ შეგიძლიათ გამოიყენოთ შემდეგი სახის ოპერატორი: class class_name.

ქვემოთ მოყვანილ მაგალითში frd() ფუნქცია გამოცხადებულია როგორც cl კლასის მეგობარი:

```
  class cl {
...
public:
friend void frd();
...
};
```

ერთ-ერთი მიზეზი, რომლის გამოც C++ ენა უშვებს ფუნქცია-მეგობრების არსებობას დაკავშირებულია ისეთ სიტუაციასთან, როდესაც ორმა კლასმა უნდა გამოიყენოს ერთიდაიგივე ფუნქცია. განვიხილოთ მაგალითი, სადაც განსაზღვრულია ორი კლასი – line და box. line კლასი შეიცავს ყველა საჭირო მონაცემს და ნებისმიერი მოცემული სიგრძის ჰორიზონტალური პუნქტირი მონაკვეთის დასახაზ კოდს. box კლასი შეიცავს საჭირო კოდს და მონაცემებს იმისათვის, რომ გამოისახოს მართკუთხედი ზედა მარცხენა და ქვედა მარჯვენა წერტილებით, მოცემული ფერით. ორივე კლასი გამოიყენებს same_color() ფუნქციას იმისათვის, რომ განისაზღვროს დახატულია თუ არა მონაკვეთი და მართკუთხედი ერთიდაიგივე ფერით. ამ კლასების გამოცხადება მოცემულია პროგრამის შემდეგ ფრაგმენტში:

```
class line;
class box {
int color; // მართკუთხედის ფერი
```

```
int upx, upy; //ზედა მარცხენა კუთხე
int lowx, lowy; //ქვედა მარჯვენა კუთხე
public:
friend int same_color (line l, box b);
void set_color(int c);
void define_box (int x1, int y1, int x2, int y2);
void show_box();
};
class line {
int color;
int startx, starty;
int len;
public:
friend int same_color (line I, box b);
void set_color(int c);
void define_line (int x, int y, int I);
void show_line();
};
```

same_color() ფუნქცია არც ერთი კლასის წევრი არ არის, მაგრამ არის ორივეს მეგობარი. იგი აბრუნებს შეტყობინებას: “Are the same color” როდესაც line ტიპის ობიექტი და box ტიპის ობიექტი დახაზულია ერთიდაიმავე ფერით, ხოლო შეტყობინებას: – “Not the same color” წინააღმდეგ შემთხვევაში. ქვემოთ წარმოდგენილია same_color() ფუნქციის განსაზღვრის კოდი:

```
int same_color (line I, box b)
{
if (I.color == b.color) cout << " Are the same color”;
```

```
    else cout<<"Not the same color";
  }
```

რადგან same_color() ფუნქცია არის ორივე კლასის მეგობარი მას ექნება ორივე კლასის კერძო მონაცემებზე წვდომა. უფრო მეტიც, მივაქციოთ ყურადღება, რომ რამდენადაც same_color() ფუნქცია არ არის წევრი, გამოყენებისას არ არის საჭირო კლასის სახელის გამოყენება. იმავე მიზნით შეიძლებოდა შეგვექმნა ფუნქცია-წევრი public წვდომის სპეციფიკატორით, რომელიც დააბრუნებდა line და box ტიპის ობიექტების ფერებს და კიდევ ერთი ფუნქცია ამ ფერების შესადარებლად. მაგრამ ასეთი მიდგომა მოითხოვს ფუნქციის დამატებით გამოძახებას, რაც მოცემულ შემთხვევაში არაეფექტურია.

ქვემოთ მოყვანილი პროგრამა დემონსტრირებას უკეთებს line და box კლასებს და უჩვენებს თუ როგორ მიიღებს მეგობარი ფუნქცია წვდომას კლასის კერძო წევრებთან:

```
#include <iostream.h>
#include <conio.h>
class line;
class box {
int color;
int upx, upy;
int lowx, lowy;
public:
friend int same_color (line l, box b);
void set_color(int c);
void define_box (int x1, int y1, int x2, int y2);
```

```
void show_box();
};
class line {
int color;
int startx, starty;
int len;
public:
friend int same_color (line I, box b);
void set_color(int c);
void define_line (int x, int y, int I);
void show_line();
};
int same_color (line I, box b)
{
if (I.color == b.color) cout << " Are the same color”;
        else cout<<”Not the same color";
        }
void box::set_color(int c)
{
color = c;
}
void line::set_color(int c)
{
color = c;
}
void box::define_box (int x1, int y1, int x2, int y2)
{
upx = x1;
upy = y1;
lowx = x2;
```

```
lowy = y2;
}
void box::show_box()
{
int i;
textcolor(color);
gotoxy(upx, upy);
for (i=upx; i<=lowx; i++) cout<<"-";
gotoxy(upx, lowy-1);
for (i=upx; i<=lowx; i++) cout<<"-";
gotoxy(upx, upy);
for (i=upy; i<=lowy; i++) {
cout<<" | ";
gotoxy(upx , i);
}
gotoxy(lowx, upy);
for (i=upy; i<=lowy; i++) {
cout<<"I";
gotoxy(lowx, i);
}
}
void line::define_line (int x, int y, int I)
{
startx = x;
starty = y;
len = I;
}
void line::show_line()
{
int i;
```

```
textcolor(color);
gotoxy(startx, starty);
for (i=0; i<len; i++) cout<<"-";
}
int main()
{
box  b;
line  I;
b.define_box (10, 10, 15, 15);
b.set_color(3);
b.show_box();
I.define_line (2, 2, 10);
I.set_color(2);
I.show_line();
same_color (I, b);
I.define_line (22, 22, 10);
I.set_color(3);
I.show_line();
same_color (I, b);
        cout << "\nPress a key.";
getch();
}
```

არსებობს მეგობარ ფუნქციებთან მიმართებით ორი მნიშვნელოვანი შეზღუდვა:

- წარმოებული კლასები არ მემკვიდრეობენ მეგობარ ფუნქციებს;
- მეგობარი ფუნქციები არ შეიძლება გამოცხადებულ იქნან static და extern გასაღებური სიტყვებით.

განვიხილოთ მაგალითი, სადაც Sum ფუნქცია ორივე კლასის მეგობარია, ამიტომ მას პირდაპირი წვდომა აქვს ორივე კლასის კერძო მონაცემებთან.

```
#include <iostream>
using namespace std;
class b;
class a{
    friend int sum (a,b);
private:
   int i;
  public:
    a() {cout<<"sheitaneT i: ";
 cin>>i;
}
};
class b {
  friend int sum (a,b);
private:
  int y;
public:
 b() {
  cout<<"sheitaneT y:";
cin>>y;
}
};
int sum ( a first, b second) {
return (first.i+second.y);
}
int main()
{a first;
```

b second;
cout<<sum (first, second)<<endl;
cin.get();
return 0;
}
შედეგი:

```
sheitaneT i: 12
sheitaneT y:16
28
```

განვიხილოთ მაგალითი მეგობარ კლასებზე:

```
#include <iostream>
using namespace std;
class XYZ {
private:
   char ch='A';
   int num = 11;
public:

   friend class ABC;
};
class ABC {
public:
   void disp(XYZ obj){
      cout<<obj.ch<<endl;
      cout<<obj.num<<endl;
   }
};
int main() {
   ABC obj;
   XYZ obj2;
   obj.disp(obj2);
   return 0;
```

}

შედეგი:

A

11

დავალება: შექმენით ორი კლასი: თსუ და სტუ, რომელთაც ექნებათ კერძო მონაცემი სტუდენტების რაოდენობა. დანარჩენი მონაცემები და ფუნქციები განსაზღვრეთ თქვენი სურვილით. მეგობარმა ფუნქციამ შეადაროს ერთმანეთს თსუ-ს და სტუ-ს სტუდენტების რაოდენობები და გამოიტანოს შესაბამისი შეტყობინება: ტოლია თუ განსხვავებულია.

ლაბორატორიული სამუშაო 12
თემა: პოლიმორფიზმი. ვირტუალური ფუნქციები

ზოგადად პოლიმორფიზმი არის ობიექტის ფორმის შეცვლის უნარი. თუ ამ ტერმინს დავყოფთ ნაწილებად, აღმოვაჩენთ, რომ, პოლი არის ბევრი, ხოლო მორფიზმი ფორმის ცვლილება. შესაბამისად, პოლიმორფული ობიექტი არის ისეთი ობიექტი, რომელსაც შეუძლია მიიღოს სხვადასხვა ფორმა. მოცემული სამუშაო გაგაცნობთ პოლიმორფიზმის არსს და შეგასწავლით როგორ გამოვიყენოთ პროგრამაში პოლიმორფული ობიექტები კოდის შემცირებისა და გამარტივებისათვის.

მასალაში შეისწავლით შემდეგ ძირითად კონცეფციებს:

- პოლიმორფიზმი წარმოადგენს ობიექტის უნარს შეიცვალოს ფორმა პროგრამის შესრულების დროს;
- C++ ამარტივებს პოლიმორფული ობიექტების შექმნას;
- პოლიმორფული ობიექტების შექმნისათვის თქვენმა პროგრამამ უნდა გამოიყენოს ვირტუალური(*virtual)* ფუნქციები;
- ვირტუალური(*virtual)* ფუნქცია – ეს არის საბაზო კლასის ფუნქცია, რომლის სახელის წინ განთავსებულია გასაღებური სიტყვა *virtual;*
- საბაზო კლასიდან ნებისმიერ წარმოებულს შეუძლია გამოიყენოს ან გადატვირთოს ვირტუალური ფუნქცია;
- პოლიმორფული ობიექტების შექმნისათვის უნდა გამოიყენოთ საბაზო კლასის ობიექტზე მიმთითებელი.

განვიხილოთ მაგალითი, სადაც ნაჩვენებია, რომ პოლიმორფიზმის რეალიზება ხდება ვირტუალური ფუნქციით, რომელიც განისაზღვრება საბაზო კლასში და მისი გადატვირთვა შეიძლება წარმოებულ კლასში. პოლიმორფული ობიექტის შესაქმნელად გამოიყენება საბაზო კლასზე მიმთითებელი:

```
class A {
  public:
    virtual void f() = 0;
}

class B: A{
  public:
    void f() {};
}
class C: A{
  public:
    void f() {};
}
A * b = new B();
A * c = new C();
b->f(); // აქ იქნება გამოძახებული B კლასის მეთოდი
c->f(); // აქ იქნება გამოძახებული  C კლასის მეთოდი
```

დავუშვათ, პროგრამისტს, რომელიც მუშაობს სატელეფონო კომპანიაში სჭირდება დაწეროს პროგრამა, რომელიც სახეს უცვლის სატელეფონო ოპერაციებს. როგორც ვიცით, ტელეფონის გამოყენებისას სრულდება ოპერაციები: ნომრის აკრეფა, ზარი, გაწყვეტა, დაკავების

ინდიკაცია. ამ ოპერაციების დახმარებით, თქვენ განსაზღვრავთ შემდეგ კლასს:

```
class phone

{
public:
  void dial(char "number) { cout << "nomris akrefa " << number << endl; }
  void answer(void) { cout << "pasuxis molidini" << endl; }
  void hangup(void) { cout << "zari Sesrulebulia – kurmilis dadeba" << endl; }
  void ring(void) { cout << "zari,zari,zari" << endl;)
  phone(string number) { phone::number=number; };
private:
  string number;
);
```

ქვემოთ წარმოდგენილია პროგრამა, რომელიც გამოიყენებს *phone* კლასს ობიექტ – ტელეფონის შესაქმნელად:

```
#include <iostream>
#include <string>
using namespace std;
class phone
{
public:
  void dial(string number) {
         cout << "nomris akrefa "<<number << endl;
  }
  void answer(void) {
         cout << "pasuxis molodini" << endl;
  }
  void hangup(void) {
```

```
  cout << "zari Sesrulebulia – kurmilis dadeba" << endl;
  }
                void ring(void) {
                        cout << "zari, zari, zari" << endl;
                }
        phone(string number) {
                number= number;
        };
private:
  char number[13];
  };

int main()
{
  system("color F0");
  phone telephone("555-1212");
  telephone.dial("212-555-1212");
}
```

```
nomris akrefa 212-555-1212

C:\Users\G comp service\source\repos\polimorfizmi\Debug\polimorfizmi
```

მოცემული პროგრამა არ აკეთებს განსხვავებას დისკურ და ღილაკებიან ტელეფონებს შორის. ამავე დროს მას არა აქვს ფასიანი ტელეფონის მხარდაჭერა. აქედან გამომდინარე ლოგიკურია მივიღებთ გადაწყვეტილებას შევქმნათ *touch_tone* და pay_phone კლასები, *phone* კლასიდან, როგორც ქვემოთ არის ნაჩვენები:

```
class touch_tone : phone

{
public:
  void dial(string  number) { cout << "Rilakebianze nomris
akrefa " << number << endl; }
  touch_tone(string number) : phone(number) { }
};
```

```
class pay_phone : phone

{
public:
  void dial(string number)

  {
    cout << "gTxovT gadaixadoT " << amount << " TeTri" <<
endl;
    cout << "nomris akrefa " << number << endl;
  }

  pay_phone(string number, int amount) : phone(number)
{pay_phone::amount = amount; }
private:
  int amount;
};
```

როგორც ხედავთ *touch_tone* და pay_phone კლასები განსაზღვრავენ თავის საკუთარ *dial* მეთოდს. phone კლასის *dial* მეთოდი გათვალიწინებულია დისკური ტელეფონისთვის. შემდეგი პროგრამა გამოიყენებს ამ კლასებს rotary ობიექტის შექმნისათვის:

```
#include <iostream>
#include <string>
using namespace std;
class phone
{
public:
  void dial(string number) {
        cout << "nomris akrefa " << number << endl;
  }
  void answer(void) { cout << "pasuxis molodini" << endl;
}
  void hangup(void) {
```

```
        cout << "zari Sesrulebulia – kurmilis dadeba" << endl;
        }
        void ring(void) { cout << "zari, zari, zari" << endl; }
        phone(string number) {
                phone::number= number;
        };
      protected:
        string number;
      };
        class touch_tone : phone
        {
        public:
                void dial(string number) {
                cout << "Rilakebianze nomris akrefa " << number
<< endl;
                }
                touch_tone(string number) : phone(number) { }
        };
        class pay_phone : phone
        {
        public:
                void dial(string  number) {
                cout  <<  "gTxovT  gadaixadoT  "  <<  amount  <<  "
TeTri" << endl;
                cout << "nomris akrefa "<< number << endl;
                }
                pay_phone(string    number,    int    amount)    :
phone(number)
                {
                pay_phone::amount = amount;
                }
        private:
                int amount;
        };

      int main()
      {
        system("color F0");
        phone rotary("303-555-1212");
        rotary.dial("602-555-1212");
        touch_tone telephone("555-1212");
        telephone.dial("212-555-1212");
        pay_phone city_phone("555-1111", 25);
```

```
    city_phone.dial("212-555-1212");
}
```

გაუშვით პროგრამა კომპილაციაზე და შემდეგ შესრულებაზე, ეკრანზე გამოვა შემდეგი შედეგი:

```
nomris akrefa 602-555-1212
Rilakebianze nomris akrefa 212-555-1212
gTxovT gadaixadoT 25 TeTri
nomris akrefa 212-555-1212
```

განხილული პროგრამა არ გამოიყენებს პოლიმორფულ ობიექტებს. ანუ მასში არ არის ობიექტები, რომლებიც შეიცვლიან ფორმას. მიზანშეწონილია სხვადასხვა ზარის დროს ჩვენი ობიექტი-ტელეფონი იცვლიდეს ფორმას.

პოლიმორფული ობიექტი-ტელეფონის შექმნა

ტელეფონის სხვადასხვა კლასებს შორის არსებობს ერთადერთი განმასხვავებელი ფუნქცია – ეს არის *dial* მეთოდი. პოლიმორფული ობიექტის შესაქმნელად თავდაპირველად უნდა განისაზღვროს საბაზო კლასის ფუნქციები, რომლებიც განსხვავდებიან წარმოებული კლასების ფუნქციებიდან იმით, რომ ისინი ვირტუალურია. მათ პროტოტიპებს წინ უძღვის *virtual* გასაღებური სიტყვა, როგორც ნაჩვენებია ქვემოთ:

```
class phone
{
public:
  virtual void dial(string number) { cout << "nomris akrefa " <<
number << endl; }
  void answer(void) { cout << "pasuxis molodini" << endl; }
  void hangup(void) { cout << "zari Sesrulebulia  - kurmilis
```

```
dadeba" << endl; }
  void ring(void) { cout << "zari, zari, zari" << endl; }
  phone(string number) { strcpy(phone::number, number); };
protected:
  string  number;
};
```

შემდეგ, პროგრამაში შევქნათ საბაზო კლასის ობიექტზე მიმთითებელი. ჩვენს შემთხვევაში მიმთითებელი *phone* საბაზო კლასზე:

```
phone *poly_phone;
```

ობიექტის ფორმის შესაცვლელად უბრალოდ მივანიჭოთ ამ მიმთითებელს წარმოებული კლასის ობიექტის მისამართი, როგორც ქვემოთ არის ნაჩვენები:

```
poly_phone = (phone *) &home_phone;
```

სიმბოლოები *(phone *)*, რომელიც იწერება მინიჭების ოპერატორის შემდგომ, წარმოადგენს ტიპებში მოყვანის ოპერატორს, რომელიც ატყობინებს კომპილატორს, რომ ყველაფერი წესრიგშია, საჭიროა ერთი ტიპის (*touch_tone)* ცვლადის მისამართს მიენიჭოს სხვა ტიპის *(phone)* ცვლადზე მიმთითებელი. რამდენადაც ქვემოთ წარმოდგენილ პროგრამას შეუძლია poly_phone ობიექტზე მიმთითებელს მიანიჭოს სხვადასხვა ობიექტის მისამართი, ამდენად ობიექტს შეუძლია ფორმის შეცვლა, რის გამოც იგი პოლიმორფულია.

```
#include <iostream>
#include <string>
using namespace std;
class phone
```

```
      {
      public:
        virtual void dial(string number) {
        cout << "nomris akrefa " << number << endl;
        }
        void answer(void) { cout << "pasuxis molodini"<<endl;
}
        void hangup(void) {
        cout << "zari Sesrulebulia – kurmilis dakideba" <<
endl;
        }
        void ring(void) { cout << "zari, zari, zari" << endl;
}
        phone(string number) {
               phone::number= number;
        };
      protected:
        string number;
      };
        class touch_tone : phone
        {
        public:
        void dial(string number) {
        cout << "Rilakebianze nomris akrefa " << number <<
endl;
               }
        touch_tone(string number) : phone(number) { }
        };
        class pay_phone : phone
        {
        public:
        void dial(string  number) {
      cout << "gTxovT gadaixadoT " << amount << " TeTri" <<
endl;
        cout << "nomris akrefa "<< number << endl;
               }
        pay_phone(string number, int amount) : phone(number)
               {
               pay_phone::amount = amount;
               }
        private:
```

```
        int amount;
};

int main()
{
  system("color F0");
  pay_phone city_phone("702-555-1212", 25);
  touch_tone home_phone("555-1212");
  phone rotary("201-555-1212");
  //obieqtis Seqmna diskur telefonad
  phone *poly_phone = &rotary;
  poly_phone->dial("818-555-1212");
  // obieqtis formis Secvla Rilakebian telefonad
  poly_phone = (phone *)&home_phone;
  poly_phone->dial("303-555-1212");
  // obieqtis formis Secvla fasian telefonad
  poly_phone = (phone *)&city_phone;
  poly_phone->dial("212-555-1212");
}
```

პროგრამას გავუკეთოთ კომპილაცია და გავუშვათ შესრულებაზე, ეკრანზე გამოვა შემდეგი შედეგი:

```
ıomris akrefa 818-555-1212
ŧilakebianze nomris akrefa 303-555-1212
ĝTxovT gadaixadoT 25 TeTri
ıomris akrefa 212-555-1212
```

რამდენადაც *poly_phone* ობიექტი იცვლის ფორმას პროგრამის შესრულებისას, იგი არის პოლიმორფული.

როგორც ვხედავთ, პოლიმორფული ობიექტის შექმნისას პროგრამამ უნდა გამოიყენოს საბაზო კლასის ობიექტზე მიმთითებელი. შემდგომ პროგრამამ უნდა მიანიჭოს ამ მიმთითებელს წარმოებული კლასის ობიექტის სახელი. ცალკეულ შემთხვევაში, როდესაც პროგრამა მიმთითებელს ანიჭებს სხვა კლასის ობიექტის მისამართს, ამ მიმთითებლის ობიექტი (რომელიც არის

პოლიმორფული) იცვლის ფორმას. პროგრამები აგებენ პოლიმორფულ ობიექტებს, საბაზო კლასის ვირტუალურ ფუნქციებზე დაფუძნებით.

რა არის წმინდად ვირტუალური ფუნქციები

როგორც ვიცით, პოლიმორფული ობიექტის შექმნისათვის პროგრამები განსაზღვრავენ საბაზო კლასის ერთ ან რამდენიმე მეთოდს, როგორც ვირტუალურ ფუნქციას. წარმოებულ კლასს შეუძლია განსაზღვროს თავისი საკუთარი ფუნქცია, რომელიც სრულდება საბაზო კლასის ვირტუალური ფუნქციის ნაცვლად, ან გამოიყენოს საბაზო ფუნქცია. სხვა სიტყვებით რომ ვთქვათ წარმოებულ კლასს შეუძლია არც განსაზღვროს თავისი საკუთარი მეთოდი. პროგრამის მიხედვით ზოგჯერ აზრი არა აქვს საბაზო კლასში ვირტუალური ფუნქციის განსაზღვრას. მაგალითად, წარმოებული ტიპის ობიექტები შეიძლება იმდენად იყოს განსხვავებული, მათ არც კი ესაჭიროებოდეთ საბაზო კლასის მეთოდის გამოყენება. ასეთ შემთხვევაში საბაზო კლასის ვირტუალური ფუნქციისათვის ოპერატორების განსაზღვრის ნაცვლად, თქვენ პროგრამებს შეუძლიათ შექმნან წმინდად ვირტუალური ფუნქცია, რომელიც არ შეიცავს ოპერატორებს.

წმინდად ვირტუალური ფუნქციის შესაქმნელად პროგრამა მიუთითებს ფუნქციის პროტოტიპს, მაგრამ არ მიუთითებს მის ოპერატორებს. მათ ნაცვლად პროგრამა ფუნქციას მიანიჭებს ნულ მნიშვნელობას, როგორც ქვემოთ არის ნაჩვენები:

```
class phone
{
public:
  virtual void dial (string number) =0; // წმინდად ვირტუალური
ფუნქცია
  void answer(void) { cout << "pasuxis molodini" << endl; }
  void hangup(void) { cout << "zari Sesrulebulia – kurmilis
dakideba" << endl; }
  void ring(void) { cout << "zari, zari, zari " << endl; }
  phone(string number) {phone::number= number; };
protected:
  string number; };
```

დასკვნა:

- პოლიმორფულ ობიექტს შეუძლია შეიცვალოს ფორმა პროგრამის შესრულების დროს;
- თქვენ ქმნით პოლიმორფულ ობიექტებს კლასების გამოყენებით, რომლებიც შექმნილია არსებული საბაზო კლასებიდან;
- საბაზო კლასში პოლიმორფული ობიექტისათვის უნდა განსაზღვროთ ერთი ან რამდენიმე ფუნქცია, როგორც ვირტუალური (*virtual)*;
- ზოგადად პოლიმორფული ობიექტები განსხვავდებიან საბაზო კლასის ვირტუალური ფუნქციების გამოყენებით;
- პოლიმორფული ობიექტის შესაქმნელად საჭიროა შექმნათ საბაზო კლასის ობიექტზე მიმთითებელი;
- პოლიმორფული ობიექტის ფორმის შესაცვლელად მიმთითებელი უნდა მიმართოთ სხვადასხვა ობიექტზე;

- წმინდად ვირტუალური ფუნქცია არის საბაზო კლასის ვირტუალური ფუნქცია, რომლისთვისაც საბაზო კლასში არ არის განსაზღვრული ოპერატორები. მათ ნაცვლად საბაზო კლასი ასეთ ფუნქციას ანიჭებს 0 მნიშვნელობას;
- წარმოებული კლასები უნდა უზრუნველყოფდნენ ფუნქციის განსაზღვრას საბაზო კლასის ცალკეული წმინდად ვირტუალური ფუნქციისთვის.

დავალება. შექმენით კლასი ფიგურა, რომელშიც დაცული ჭდით გამოაცხადეთ სამი ნამდვილი ტიპის ცვლადი, ხოლო საჯარო ჭდით განსაზღვრეთ კონსტრუქტორი ორი პარამეტრით და ვირტუალური ფუნქცია „ფართობი“, რომელიც გამოითვლის და გამოიტანს მართკუთხედის ფართობს. შექმენით ფიგურა კლასის მემკვიდრე კლასი „კვადრატი“, რომელშიც ღია ნაწილში განსაზღვრეთ ერთპარამეტრიანი კონსტრუქტორი და ფუნქცია „ფართობი“, რომელიც გამოითვლის კვადრატის ფართობს. პროგრამის მთავარ ფუნქციაში გამოაცხადეთ საბაზო და მემკვიდრე კლასების თითო ობიექტი. შემოიტანეთ საბაზო კლასის მიმთითებელი და მას მიანიჭეთ საბაზო კლასის ობიექტის მისამართი. მიმთითებლის მეშვეობით გამოიძახეთ „ფართობი“ საბაზო კლასიდან. შემდეგ მიმთითებელს მიანიჭეთ მემკვიდრე კლასის ობიექტის მისამართი და მისი მეშვეობით გამოიძახეთ მემკვიდრე კლასის „ფართობი“ ფუნქცია. შედეგები გააანალიზეთ.

ლაბორატორიული სამუშაო 13
თემა: ფუნქციების და კლასების შაბლონების შექმნა
ფუნქციების შაბლონების გამოყენება

ფუნქციების შექმნისას წარმოიშვება სიტუაციები, როდესაც ორი ფუნქცია ასრულებს ერთიდაიგივე მოქმედებას, მაგრამ მუშაობენ სხვადასხვა მონაცემთა ტიპებთან(მაგალითად, ერთი გამოიყენებს int ტიპის პარამეტრებს, ხოლო მეორე – float ტიპის). დავუშვათ, გვაქვს ფუნქცია max სახელით, რომელიც აბრუნებს ორი მთელი რიცხვიდან უდიდესს. თუ მოგვიანებით ჩვენ დაგვჭირდება მსგავსი ფუნქცია, რომელიც აბრუნებს უდიდესს ორი მცოცავწერტილიანი მნიშვნელობიდან, საჭიროა განვსაზღვროთ სხვა ფუნქცია, მაგალითად fmax. მოცემულ მასალაში თქვენ შეისწავლით თუ როგორ გამოვიყენოთ C++ ენის შაბლონები ისეთი ფუნქციების სწრაფად შექმნისათვის, რომლებიც აბრუნებენ სხვადასხვა ტიპის მნიშვნელობებს.

მოცემულ მასალაში თქვენ შეისწავლით შემდეგ კონცეფციებს:

– შაბლონი განსაზღვრავს ოპერატორების ნაკრებს, რომელთა დახმარებით თქვენ პროგრამას მოგვიანებით შეუძლია რამდენიმე ფუნქციის შექმნა;

– ფუნქციების შაბლონი ხშირად გამოიყენება რამდენიმე ფუნქციის სწრაფად განსაზღვრისათვის, რომლებიც ერთნაირი ოპერატორების დახმარებით მუშაობენ სხვადასხვა ტიპის პარამეტრებთან ანუ

აქვთ დასაბრუნებელი მნიშვნელობის სხვადასხვა ტიპი;

– ფუნქციის შაბლონებს აქვთ სპეციფიური სახელები, რომლებიც შეესაბამებიან პროგრამაში თქვენ მიერ გამოყენებულ ფუნქციის სახელებს;

– პროგრამაში ფუნქციის შაბლონის განსაზღვრის შემდგომ, შეიძლება შეიქმნას კონკრეტული ფუნქცია, სადაც გამოიყენება ეს შაბლონი პროტოტიპის დასაწერად, რომელიც შეიცავს მოცემული შაბლონის სახელს, დასაბრუნებელ მნიშვნელობას და პარამეტრების ტიპს;

– კომპილაციის პროცესში C++ ენის კომპილატორი თქვენს პროგრამაში შექმნის ფუნქციებს პროტოტიპებში მითითებული ტიპების გამოყენებით, რომლებიც იგზავნებიან შაბლონის სახელზე.

ფუნქციის საბლონებს აქვთ უნიკალური სინტაქსი, რომელიც ერთი შეხედვით შეიძლება გაუგებარი იყოს. რამდენიმე შაბლონის შექმნის შემდეგ თქვენ მიხვდებით, რომ რეალურად მათი გამოყენება იოლია.

ფუნქციის მარტივი შაბლონის შექმნა

ფუნქციის შაბლონი განსაზღვრავს ტიპობრივად დამოუკიდებელ ფუნქციას. ასეთი შაბლონის დახმარებით პროგრამაში შეიძლება განისაზღვროს კონკრეტული ფუნქციები საჭირო ტიპებით. მაგალითად, ქვემოთ განსაზღვრულია შაბლონი max ფუნქციისათვის, რომელიც აბრუნებს უდიდესს ორი მნიშვნელობიდან:

```
template<class T> T max(T a, T b)
```

```
{
  if (a > b) return(a);
  else return(b);
}
```

T სიმბოლო მოცემულ შემთხვევაში წარმოადგენს შაბლონის ზოგად ტიპს. შაბლონის განსაზღვრის შემდეგ თქვენ პროგრამაში გამოაცხადეთ პროტოტიპი ცალკეული მოთხოვნილი ტიპისათვის. ჩვენს შემთხვევაში:

```
float max(float, float);
int max(int, int);
```

როდესაც C++ კომპილატორს შეხვდება ეს პროტოტიპები, ფუნქციების აგებისას T შაბლონის ტიპს იგი შეცვლის თქვენ მიერ მითითებული ტიპით. float ტიპის შემთხვევაში max ფუნქცია შეცვლის შემდეგ მიიღებს სახეს:

```
float max(float a, float b)

{
  if (a > b) return(a) ;
  else return(b);
}
```

ქვემოთ წარმოდგენილი პროგრამა გამოიყენებს max შაბლონს int და float ტიპის ფუნქციების შესაქმნელად:

```
#include <iostream.h>
template<class T> T max(T a, T b)
{
  if (a > b) return(a);
```

```
  else return(b);
}
float max(float, float);
int max(int, int);
void main(void)
{
  cout << "udidesi 100 da 200 Soris tolia " << max(100, 200) <<
endl;
  cout << "udidesi 5.4321 da 1.2345 Soris tolia " << max(5.4321,
1.2345) << endl;
}
```

რადგან ეს ფუნქცია შექმნილია შაბლონის დახმარებით, იგი ნებას რთავს გამოყენებულ იქნას ერთნაირი სახელი ფუნქციებისათვის, რომლებიც აბრუნებენ განსხვავებული ტიპების მნიშვნელობებს. ამის გაკეთება შეუძლებელი იქნებოდა მხოლოდ ფუნქციების გადატვირთვის გამოყენებით.

შაბლონები, რომლებიც გამოიყენებენ რამდენიმე ტიპს

შემდეგი ოპერატორები ქმნიან შაბლონს show_array ფუნქციისათვის, რომელიც გამოიტანს მასივის ელემენტებს. შაბლონი გამოიყენებს T ტიპს მასივის ტიპის განსაზღვრისათვის და T1 ტიპს count პარამეტრის ტიპის მისათითებლად:

```
template<class T,class T1> void show_array(T *array,T1 count)

{
  T1 index;
 for (index =0; index < count; index++) cout << array[index] << ' ';
  cout << endl;
}
```

როგორც ზემოთ, პროგრამამ უნდა მიუთითოს ფუნქციების პროტოტიპები მოთხოვნილი ტიპებისათვის:

```
void show_array(int *, int);
void show_array(float *, unsigned);
```

ქვემოთ წარმოდგენილი პროგრამა გამოიყენებს შაბლონს ფუნქციის შექმნისათვის, რომლებიც გამოიტანენ int და float ტიპის მასივებს:

```
#include <iostream.h>
template<class T,class T1> void show_array( T *array,T1 count)
{
  T1 index;
  for (index =0; index < count; index++) cout << array[index] “ ' ';
  cout << endl;
}
void show_array(int *, int);
void show_array(float *, unsigned);
void main(void)
{
  int pages[] = { 100, 200, 300, 400, 500 };
  float pricesH = { 10.05, 20.10, 30.15 };
  show_array(pages, 5);
  show_array(prices, 3);
}
```

მოცემულ შემთხვევაში ფუნქცია იყენებს ორ პარამეტრს. ერთი შეესაბამება მასივს, ხოლო მეორე - მასივში ელემენტების რაოდენობას. უფრო უნივერსალური შაბლონი პროგრამას ამ პარამეტრის საკუთარი ტიპის მითითების შესაძლებლობას მისცემდა, როგორც ქვემოთ არის ნაჩვენები:

template<class T, class T1> void array_sort(T array[], T1 elements)
{
// операторы
}

array_sort შაბლონის დახმარებით პროგრამას შეუძლია შექმნას ფუნქცია, რომელიც სორტირებას უკეთებს ***float*** ტიპის პატარა მასივებს (128 ელემენტზე ნაკლები) და ***int*** ტიპის ძალიან დიდი ზომის მასივებს შემდეგი პროტოტიპების გამოყენებით:

void array_sort(float, char);
void array_sort(int, long);

დასკვნა:

- ფუნქციების შაბლონები საშუალებას იძლევა გამოცხადდეს ტიპობრივად დამოუკიდებელი ანუ ზოგადი ფუნქციები;
- როდესაც პროგრამაში მოითხოვება ფუნქციის გამოყენება განსაზღვრული მონაცემთა ტიპებით, მან უნდა მიუთითოს ფუნქციის პროტოტიპი, რომელიც განსაზღვრავს მოთხოვნილ ტიპებს;
- როდესაც C++ კომპილატორს შეხვდება ასეთი ფუნქციის პროტოტიპი, იგი ქმნის ამ ფუნქციის შესაბამის ოპერატორებს, მოთხოვნილი ტიპების გამოყენებით;
- პროგრამამ უნდა შექმნას შაბლონები საერთო ფუნქციებისათვის, რომლებიც მუშაობენ განსხვავებული ტიპებისთვის. სხვა სიტყვებით, თუ თქვენ გამოიყენებთ რომელიმე ფუნქციაში მხოლოდ ერთ

ტიპს, არ არის შაბლონის გამოყენების აუცილებლობა;

– თუ ფუნქცია მოითხოვს რამდენიმე ტიპს, შაბლონი უბრალოდ უნიშნავს ცალკეულ ტიპს უნიკალურ იდენტიფიკატორს, მაგალითად T, T1 და T2. მოგვიანებით კომპილაციის პროცესში კომპილატორი კორექტულად დანიშნავს თქვენ მიერ ფუნქციის პროტოტიპში მითითებულ ტიპებს.

კლასების შაბლონების გამოყენება

მოცემულ მასალაში თქვენ შეისწავლით შემდეგ ძირითად კონცეფციებს:

– template გასაღებური სიტყვის და ტიპების სიმბოლოების (მაგალითად, T, T1 და T2) გამოყენებით პროგრამას შეუძლია შექმნას კლასის შაბლონი – კლასის შაბლონის განსაზღვრას შეუძლია გამოიყენოს ეს სიმბოლოები მონაცემთა ელემენტების გამოსაცხადებლად, პარამეტრების ტიპების და ფუნქციის დასაბრუნებელი მნიშვნელობის საჩვენებლად და ა.შ.

– შაბლონის გამოყენებით კლასის ობიექტების შესაქმნელად პროგრამა აკეთებს მიმართვას კლასის სახელზე, რომლის შემდეგაც კუთხოვან ფრჩხილში იწერება ტიპი (მაგალითად, <int, float>), რომელთაგან თითოეულს კომპილატორი დაუნიშნავს ტიპების სიმბოლოებს და ცვლადის სახელს;

– თუ კლასს აქვს კონსტრუქტორი, რომლის დახმარებით ინიციალიზებას უკეთებთ მონაცემ-

ელემენტებს, შეგიძლიათ გამოიძახოთ ეს კონსტრუქტორი ობიექტის შექმნისას შაბლონის გამოყენებით, მაგალითად:

```
class_name<int,float>values(200);
```

– თუ კომპილატორს შეხვდება ობიექტის გამოცხადება, იგი ქმნის კლასს შაბლონიდან შესაბამისი ტიპების გამოყენებით.

კლასის შაბლონის შექმნა

ზოგიერთი პროგრამისათვის შექმნილი კლასი შეიძლება გამოდგეს სხვა პროგრამისთვისაც. ხშირად კლასები შეიძლება განსხვავდებოდნენ მხოლოდ ტიპებით. სხვა სიტყვებით, ერთი კლასი მუშაობდეს მთელრიცხვა მნიშვნელობებით, ხოლო მოცემულ მომენტში მოთხოვნილმა კლასმა იმუშაოს ***float*** ტიპის მნიშვნელობებით. არსებული კოდის ხელმეორედ გამოყენების მიზნით ***C++*** საშუალებას იძლევა განისაზღვროს კლასების შაბლონი. მოკლედ რომ ვთქვათ, კლასის შაბლონი განვსაზღვროთ როგორც ტიპზე დამოუკიდებელი კლასი, რომელიც მომავალში ემსახურება მოთხოვნილი ტიპების ობიექტების შექმნას. თუ კომპილატორს შეხვდება კლასის შაბლონზე დაფუძნებული ობიექტის გამოცხადება, მაშინ მოთხოვნილი ტიპის კლასის ასაგებად იგი გამოიყენებს გამოცხადებისას მითითებულ ტიპებს. მხოლოდ ტიპებით განსხვავებული კლასების სწრაფი შექმნის თვალსაზრისით კლასების შაბლონები ამცირებენ დაპროგრამების მოცულობას.

დავუშვათ ვქმნით მთელი ტიპის მასივის კლასს, რომელშიც არის მეთოდები წევრთა ჯამის და საშუალო

არითმეტიკულის გამოსათვლელად. კლასი გამოიყურება შემდეგი სახით:

```
class array

{
public:
  array(int size);
  long sum(void);
  int average_value(void);
  void show_array(void);
  int add_value(int);
private:
  int *data;
  int size;
  int index;
};
```

ქვემოთ წარმოდგენილია პროგრამა, რომელიც იყენებს array კლასს int ტიპის მნიშვნელობებთან სამუშაოდ:

```
#include <iostream.h>
#include <stdlib.h>
class array
{
public:
  array(int size);
  long sum(void);
  int average_value(void);
  void show_array(void);
  int add_value(int) ;
private:
  int *data;
```

```
  int size;
  int index;
};

array::array(int size)

{
  data = new int [size];
  if (data == NULL)

  {
    cerr << "arasakmarisia mexsiereba –programa dasruldeba " <<
endl;
    exit(l);
  }

  array:: size = size;
  array::index = 0;
}

long array::sum(void)

{
  long sum = 0;
  for (int i = 0; i < index; i++) sum += data[i];
  return(sum);
}

int array::average_value(void)

{
  long sum = 0;
  for (int i = 0; i < index; i++) sum += data[i];
```

```
  return (sum / index);
}
void array::show_array(void)
{
  for (int i = 0; i < index; i++) cout << data[i] << ' ';
  cout << endl;
}
int array::add_value(int value)
{
  if (index == size) return(-1); // masivi savsea
  else
  {
    data[index] = value;
    index++;
    return(0); //warmatebiT
  }
}
void main(void)

{
  array numbers (100); // 100 elementiani masivi
  int i;
  for (i = 0; i < 50; i++) numbers.add_value(i);
  numbers.show_array();
  cout << "ricxvebis jami tolia " << numbers.sum () << endl;
  cout << "saSualo mniSvneloba tolia " <<
numbers.average_value() << endl;
}
```

როგორც ვხედავთ, პროგრამა ანაწილებს მასივის 100 ელემენტს, შემდეგ შეაქვს მასივში 50 მნიშვნელობა add_value მეთოდის დახმარებით. array კლასში index ცვლადი მიედევნება ელემენტების რაოდენობას, რომელიც შენახულია მოცემულ მომენტში მასივში. თუ

მომხმარებელი შეეცდება დაამატოს იმაზე მეტი ელემენტი რასაც მასივი იტევს, add_value ფუნქცია აბრუნებს შეცდომას. როგორც ხედავთ average_value ფუნქცია იყენებს index ცვლადს მასივის საშუალო მნიშვნელობის განსაზღვრისათვის. პროგრამა ითხოვს მეხსიერებას მასივისათვის new ოპერატორის გამოყენებით.

დავუშვათ პროგრამას, რომელიც მუშაობს მთელ რიცხვებთან, სჭირდება მუშაობა მცოცავწერტილიანი მნიშვნელობების მასივებთან. ერთ-ერთი ხერხი ამ შემთხვევაში მდგომარეობს სხვადასხვა კლასების შექმნაში. მეორეს მხრივ კლასების შაბლონის გამოყენებით თქვენ გვერდს აუვლით კლასების დუბლირებას. ქვემოთ წარმოდგენილია კლასის შაბლონი, რომელიც ქმნის ზოგად array კლასს:

```
template<class T, class T1> class array

{
public:
  array(int size);
  T1 sum (void);
  T average_value(void);
  void show_array(void);
  int add_value(T);
private:
  T *data;
  int size;
  int index;
};
```

ეს შაბლონი განსაზღვრავს T და T1 ტიპების სიმბოლოებს. მთელრიცხვა მასივის შემთხვევაში T შეესა-

ბამება int ტიპს, ხოლო T1 – long ტიპს. ანალოგიურად მცოცავწერტილიანი მნიშვნელობების შემთხვევაში T და T1 უდრის float-ს.

შემდგომში კლასის ცალკეული ფუნქციის წინ უნდა მიუთითოთ ისეთივე ჩანაწერი template სიტყვით. კლასის სახელის შემდეგ უნდა მიუთითოთ კლასის ტიპები, მაგალითად array <T, T1>::average_value. შემდეგი ოპერატორი გვიჩვენებს ამ კლასისათვის average_value ფუნქციის განსაზღვრას:

```
template<class T, class T1> T array<T, T1>::average_value(void)

{
  T1 sum = 0;
  int i;
  for (i = 0; i < index; i++) sum += data[i] ;
  return (sum / index);
}
```

შაბლონის შექმნის შემდეგ შეგიძლიათ შექმნათ მოთხოვნილი ტიპის კლასი, კლასის სახელის მითითებით, რომელსაც თან ახლავს კუთხურ ფრჩხილებში საჭირო ტიპები, როგორც ქვემოთ არის ნაჩვენები:

შაბლონის სახელი//----> array <int, long> numbers (100); <----// *შაბლონის ტიპი*

array <float, float> values(200);

ქვემოთ წარმოდგენილი პროგრამა გამოიყენებს array კლასის შაბლონს ორი კლასის შესაქმნელად,

რომელთაგან ერთი მუშაობს int ტიპის, ხოლო მეორე – float ტიპის მნიშვნელობებთან:

```
#include <iostream.h>
#include <stdlib.h>
template<class T, class T1> class array
{
public:
  array(int size);
  T1 sum(void);
  T average_value(void);
  void show_array(void);
  int add_value(T);
private:
  T *data;
  int size;
  int index;
};
template<class T, class T1> array<T, t1>::array(int size)
{
  data = new T[size];
  if (data == NULL)
  {
    cerr << "მეხსიერება არასაკმარისია – პროგრამა
დასრულდება" << endl;
    exit(l);
  }
  array::size = size;
  array::index = 0;
}
template<class T, class T1> Tl array<T, Tl>::sum(void)
```

```
{
  T1 sum = 0;
  for (int i = 0; i < index; i++) sum += data[i];
  return(sum);
}
template<class T, class T1> T array<T, T1>::average_value(void)
{
  Tl sum =0;
  for (int i = 0; i < index; i++) sum += data[i];
  return (sum / index);
}
template<class T, class T1> void array<T, T1>::show_array(void)
{
  for (int i = 0; i < index; i++) cout << data[i] << ' ';
  cout << endl;
}
template<class T, class T1> int array<T, T1>::add_value(T value)
{
  if (index == size)
  return(-1); // მასივი  სავსეა
  else
  {
    data[index] = value;
    index++;
    return(0); // წარმატებით
  }
}
void main(void)
{
  // 100 ელემენტიანი  მასივი
  array<int, long> numbers(100)7
  // 200 ელემენტიანი  მასივი
  array<float, float> values(200);
  int i;
```

```
  for (i = 0; i < 50; i++) numbers.add_value(i);
  numbers.show_array();
  cout << "რიცხვების ჯამი ტოლია " << numbers.sum () <<
endl;
  cout << "საშუალო მნიშვნელობა ტოლია " <<
numbers.average_value() << endl;
  for (i = 0; i < 100; i++) values.add_value(i * 100);
  values.show_array();
  cout << "რიცხვების ჯამი ტოლია " << values.sum() <<
endl;
  cout << "საშუალო მნიშვნელობა ტოლია " <<
values.average_value() << endl;
}
```

ობიექტების გამოცხადება კლასის შაბლონის საფუძველზე

კლასის შაბლონის გამოყენებით ობიექტების შექმნისათვის უნდა მიუთითოთ კლასის შაბლონის სახელი, რომლის შემდეგაც კუთხოვან ფრჩხილებში მიუთითეთ ტიპები, რომლითაც კომპილატორი შეცვლის ***T, T1, T2*** და ა.შ. სიმბოლოებს. ამის შემდეგ პროგრამამ უნდა მიუთითოს ობიექტის (ცვლადის) სახელი იმ პარამეტრების მნიშვნელობებით, რომელიც უნდა გადასცეთ კლასის კონსტრუქტორს, როგორც ქვემოთ არის ნაჩვენები:

template_class_name<typel, type2> object_name(parameter1, parameter2);

როდესაც კომპილატორს ხვდება ასეთი გამოცხადება, იგი ქმნის მითითებულ ტიპებზე დაფუძნებულ კლასს. მაგალითად, შემდეგი ოპერატორი იყენებს ***array*** კლასის

შაბლონს ***char*** ტიპის მასივის შესაქმნელად, რომელშიც ინახება 100 ელემენტი:

array<char, int> small_numbers(100) ;

დასკვნა:

- კლასების შაბლონები საშუალებას იძლევა თავი ავარიდოთ კოდის დუბლირებას ისეთი კლასებისათვის, რომლის ობიექტები განსხვავდებიან მათი ელემენტების მხოლოდ ტიპებით;
- კლასის შაბლონის შესაქმნელად კლასის გამოცხადებას წაუმძღვარეთ template გასაღებური სიტყვა და ტიპების სიმბოლოები, მაგალითად T და T1;
- შემდგომ კლასის ცალკეული ფუნქციის განსაზღვრას წაუმძღვარეთ template გასაღებური სიტყვა. გარდა ამისა მიუთითეთ შაბლონის ტიპი კუთხოვან ფრჩხილებში, ხოლო გამოსახულება კუთხოვან ფრჩხილებში განათავსეთ კლასის სახელსა და ხედვის არეს დაშვების ოპერატორს შორის, მაგალითად:
  class_name<T,T1>::function_name.
- შაბლონის გამოყენებით კლასის შესაქმნელად მიუთითეთ კლასის სახელი და ტიპებისათვის ჩამნაცვლებელი მნიშვნელობები, კუთხოვან ფრჩხილებში. მაგალითად, class_name<int, long> object.

ლაბორატორიული სამუშაო 14
თემა: **cin** და **cout** დამატებითი შესაძლებლობები

როგორც განვლილი მასალებიდან ჩანს, **c++** ენაზე დაწერილი ყოველი პროგრამა შეიცავს *iostream.h* სათაო ფაილს. მაგრამ აქამდე ჩვენ არ ვიცოდით რა არის ამ ფაილში. ეს ფაილი განსაზღვრას კლასებს *istream* და *ostream* (შემავალი ნაკადი და გამომავალი ნაკადი), ხოლო *cin* და *cout* არიან ამ კლასების ცვლადები (ობიექტები).

მოცემულ მასალაში თქვენ გაეცნობით როგორ ფართოვდება შეტანა-გამოტანის შესაძლებლობები ფუნქციების გამოყენებით, რომლებიც ჩაშენებულია **cin** და **cout** კლასებში.

თქვენ შეისწავლით შემდეგ ძირითად კონცეფციებს:

- **iostream.h** სათაო ფაილი შეიცავს კლასის განსაზღვრას, რომელიც თქვენ შეგიძლიათ გააანალიზოთ, რათა უკეთ გაიგოთ ნაკადური შეტანა/გამოტანა;
- **cout.width** მეთოდის გამოყენებით თქვენ პროგრამებს შეუძლიათ გამოტანის სიგანის მართვა;
- **cout.fill** მეთოდის გამოყენებით თქვენ პროგრამებს შეუძლიათ ცარიელი გამომავალი სიმბოლოების (ტაბულაცია და ხარვეზი) შეცვლა ზოგიერთი განსაზღვრული სიმბოლოთი;
- მცოცავწერტლიანი მნიშვნელობების გამომავალ ნაკადში ციფრების რაოდენობის მართვისათვის შეიძლება გამოყენებულ იქნას **cout.setprecision** მეთოდი;

– სიმბოლოების სათითაოდ შეტანა-გამოტანისათვის პროგრამას შეუძლია გამოიყენოს **cout.put** და **cin.get** ნაკადური მეთოდები;
– **cin.getline** მეთოდის გამოყენებით შეიძლება მთელი სტრიქონის გამოტანა ერთდროულად.

cout გამოყენება

როგორც უკვე იცით **cout** წარმოადგენს კლასს, რომელიც შეიცავს სხვადასხვა მეთოდს. ქვემოთ წარმოდგენილ პროგრამაში ნაჩვენებია ზოგიერთი მეთოდის გამოყენება გამოტანის ფორმატირებისთვის. *setw* მანიპულატორი პროგრამას საშუალებას აძლევს მიუთითოს სიმბოლოების მინიმალური რაოდენობა, რომელიც შეუძლია დაიკავოს გამომავალმა მნიშვნელობამ:

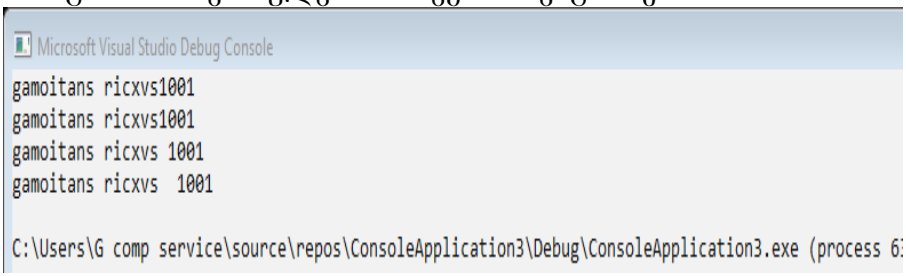
```
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
  cout << "გამოიტანს რიცხვს" << setw(3) << 1001 << endl;
  cout << "გამოიტანს რიცხვს" << setw (4) << 1001 << endl;
  cout << "გამოიტანს რიცხვს" << setw (5) << 1001 << endl;
  cout << "გამოიტანს რიცხვს" << setw(6) << 1001 << endl;
}
```

ანალოგიურად მოქმედებს *cout.width* მეთოდი, რომელიც წარმოდგენილია ქვემოთ:

```
#include <iostream>
using namespace std;
```

```
int main()
{
	system("color F0");

		int i;
		for (i = 3; i < 7; i++)
		{
			cout << "gamoitans ricxvs";
			cout.width(i);
			cout << 1001 << endl;
		}


}
```

პროგრამის შესრულებისას ეკრანზე გამოვა:



შემავსებელი სიმბოლოს გამოყენება

ზოგჯერ საჭიროა შემავსებელი სიმბოლოს გამოყენება. დავუშვათ ვაკეთებთ სარჩევს, სადაც ჩამონათვალსა და გვერდის ნომერს შორის შემავსებლად გამოყენებულია სიმბოლო წერტილი.

ინფორმაციის ცხრილი

განვიხილოთ პროგრამა, სადაც *cout.fill* ფუნქცია საშუალებას იძლევა მივუთითოთ სიმბოლო, რომელიც შეავსებს ცარიელ სივრცეს:

```
#include <iostream.h>
#include <iomanip.h>
void main(void)
{
  cout << "ინფორმაციის ცხრილი" << endl;
  cout.fill ('.');
  cout << "კომპანიის პროფილი" << setw(31) << 10 << endl;
  cout << "კომპანიის შემოსავლები და დანაკარგები" <<
setw(12) << 13 << endl;
  cout << "კომპანიის წევრები" << setw(31) << 15 << endl;
}
```

მცოცავწერტილიანი მნიშვნელობების თანრიგების მართვა

თუ პროგრამაში *cout-ს* ვიყენებთ მცოცავწერტილიანი მნიშვნელობების გამოსატანად, წინასწარ ვერ ვივარაუდებთ წერტილის შემდეგ თუ რამდენი ციფრი იქნება დუმილით გამოტანილი. **setprecision** მანიპულატორის გამოყენებით შეგვიძლია მივუთითოთ საჭირო ციფრების რაოდენობა. ქვემოთ წარმოდგენილ პროგრამაში გამოყენებულია **setprecision** მანიპულატორი წერტილიდან მარჯვნივ ციფრთა რაოდენობის მართვისათვის:

```
#include <iostream.h>
#include <iomanip.h>
void main(void)
```

```
{
  float value = 1.23456;
  int i;
  for (i = 1; i < 6; i++) cout << setprecision(i) << value << endl;
}
```

პროგრამის შესრულებისას გამოვა შემდეგი შედეგი:

1.2
1.23
1.235
1.2346
1.23456

setprecision მანიპულატორის აწყობები მოქმედებს მანამ, სანამ პროგრამა ხელახლა არ გამოიყენებს მოცემულ მანიპულატორს.

ცალკეულ ჯერზე თითო სიმბოლოს გამოტანა

ქვემოთ წარმოდგენილ პროგრამაში გამოყენებულია *cout.put* ფუნქცია შეტყობინების: ”ვსწავლობთ დაპროგრამებას c++ ენაზე” სიმბოლო-სიმბოლოდ გამოსატანად:

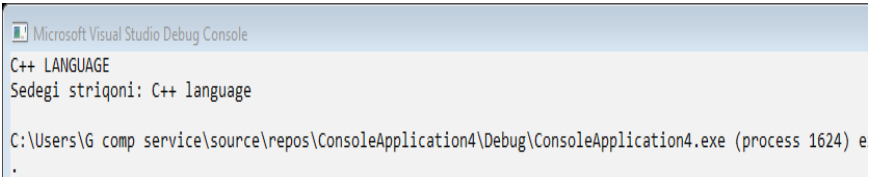
```
#include <iostream>
void main(void)
{
  char string[] = "ვსწავლობთ დაპროგრამებას C++ ენაზე!";
  int i;
  for (i = 0; string[i]; i++) cout.put(string[i]) ;
}
```

შემდეგ პროგრამაში გამოყენებულია *toupper* ფუნქცია სიმბოლოს ზედა რეგისტრში გადასაყვანად, ხოლო შემდეგ ეს სიმბოლო გამოდის *cout.put* ფუნქციის დახმარებით:

```
#include <iostream>
#include <ctype.h>
```

```
using namespace std;

int main()
{
  system("color F0");

  char string[] = "C++ language";
  int i;
  for (i = 0; string[i]; i++)
cout.put(toupper(string[i]));
  cout << endl << "Sedegi striqoni: " << string << endl;
}
```

პროგრამის გაშვების შემდეგ ეკრანზე გამოვა შედეგი:



კლავიატურიდან სტრიქონის ცალ-ცალკე სიმბოლოებად წაკითხვა

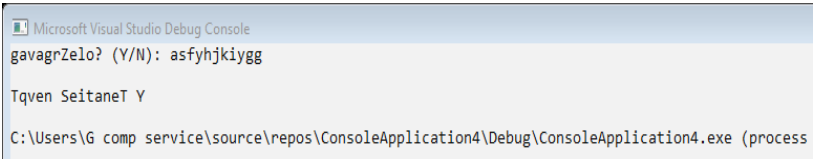
cin წარმოგვიდგენს *cin.get* ფუნქციას, რომელიც ცალკეული სიმბოლოს წაკითხვის საშუალებას იძლევა. მოცემული ფუნქციით სარგებლობისათვის სიმბოლოს ანიჭებთ ამ ფუნქციის დასაბრუნებელ ცვლადს:

letter = cin.get();

ქვემოთ წარმოდგენილია პროგრამა, რომელსაც გამოაქვს შეტყობინება, რომლის საპასუხოდ უნდა შეიტანოთ Y ან N. შემდეგ იგი იმეორებს ციკლში cin.get ფუნქციის გამოძახებას სიმბოლოს წასაკითხად, სანამ არ მიიღებს Y ან N პასუხს:

```
#include <iostream>
#include <ctype.h>
using namespace std;
```

```
int main()
{
  system("color F0");

  char letter;
  cout << "gavagrZelo? (Y/N): ";
  do
  {
        letter = cin.get();
        // zeda registrSi gardaqmna
        letter = toupper(letter);
  } while ((letter != 'Y') && (letter != 'N'));
  cout << endl << "Tqven SeitaneT " << letter << endl;
}
```



კლავიატურიდან მთელი სტრიქონის წაკითხვა

ხშირ შემთხვევაში საჭიროა, რომ პროგრამამ წაიკითხოს მთელი სიმბოლური სტრიქონი, რისთვისაც გამოიყენება *cin.getline* ფუნქცია. აღნიშნული ფუნქციის გამოყენებისას საჭიროა მიუთითოთ სიმბოლური სტრიქონი, რომელშიც განთავსდება სიმბოლოები და ასევე სტრიქონის სიგრძე, როგორც ქვემოთ არის ნაჩვენები:

cin.getline(string, 64);

როდესაც *cin.getline* კითხულობს სიმბოლოებს კლავიატურიდან, იგი არ წაიკითხავს იმაზე მეტს რასაც დაიტევს სტრიქონი. მასივის ზომის განსაზღვრისათვის

მოსახერხებელია *sizeof* ოპერატორის გამოყენება, როგორც ქვემოთ არის ნაჩვენები:

cin.getline(string, sizeof(string));

თუ თქვენ მოგვიანებით შეცვლით სტრიქონის ზომას, არ მოგიწევთ თქვენს პროგრამაში არსებულ *cin.get*-თან დაკავშირებული ცალკეული ოპერატორის ძებნა და შეცვლა. ამის ნაცვლად *sizeof* ოპერატორი გამოიყენებს სტრიქონის კორექტირებულ ზომას. ქვემოთ მოყვანილი პროგრამა გამოიყენებს cin.getline ფუნქციას ტექსტური სტრიქონის კლავიატურიდან წასაკითხად:

```
#include <iostream>
using namespace std;

int main()
{
  system("color F0");
  char string[128];
  cout << "SeitaneT teqsturi striqoni da daaWireT Enter" << endl;
  cin.getline(string, sizeof(string));
  cout << "Tqven SeitaneT: " << string << endl;
}
```

```
Microsoft Visual Studio Debug Console
SeitaneT teqsturi striqoni da daaWireT Enter
afghklk ajkjalk
Tqven SeitaneT: afghklk ajkjalk

C:\Users\G comp service\source\repos\ConsoleApplication4\Debug\ConsoleApplication4.exe (proces
.
```

ზოგჯერ საჭიროა არა მთელი სტრიქონის, არამედ რომელიმე სიმბოლომდე წაკითხვა, მსგავსი ოპერაციის შესასრულებლად საძიებო სიმბოლო უნდა იქნას გადაცემული *cin.getline* ფუნქციაში. ქვემოთ მოყვანილი

ფუნქცია სტრიქონს წაიკითხავს მანამ, სანამ არ შეხვდება სტრიქონზე გადასვლა ან სანამ არ წაიკითხავს 64 სიმბოლოს ან სანამ არ შეხვდება სიმბოლო “z”:

cin.getline(string, 64, 'z');

განვიხილოთ პროგრამა, რომელიც გამოიყენებს *cin.getline* ფუნქციას სტრიქონის წასაკითხად “z” სიმბოლოს ჩათვლით:

```
#include <iostream>
using namespace std;

int main()
{
  system("color F0");

  char string[128];
  cout << "SeitaneT striqoni da daaWireT Enter" << endl;
  cin.getline(string, sizeof(string), 'z');
  cout << "Tqven SeitaneT: " << string << endl;
}
```

```
Microsoft Visual Studio Debug Console
SeitaneT striqoni da daaWireT Enter
axchgfhgvxz
Tqven SeitaneT: axchgfhgvx

C:\Users\G comp service\source\repos\ConsoleApplication4\Debug\Conso
```

გააკეთეთ ტესტირება სხვადახვა სტრიქონისთვის. ზოგიერთი მათგანი დაიწყეთ z სიმბოლოთი, ზოგიერთი დაასრულეთ z სიმბოლოთი, ხოლო ზოგიერთი საერთოდ არ შეიცავდეს z სიმბოლოს.

დასკვნა:

- *cin* და *cout* არიან *istream* და *ostream* კლასების ობიექტები (ცვლადები), რომლებიც განსაზღვრულია *iostream.h* სათაო ფაილში. ამდენად ისინი წარმოადგენენ ფუნქციებს, რომელთაც თქვენი პროგრამები გამოიძახებენ განსაზღვრული ამოცანების ამოსახსნელად;
- cout.width საშუალებას იძლევა გამოტანის დროს მითითებულ იქნას სიმბოლოების მინიმალური რაოდენობა;
- cout.fill ფუნქცია საშუალებას იძლევა მითითებულ იქნას cout.width ან setw გამოყენებით წარმოქმნილი ცარიელი სივრცის შემავსებელი სიმბოლო;
- setprecision მანიპულატორი საშუალებას იძლევა მცოცავწერტილიანი რიცხვების გამოტანის დროს მითითებულ იქნას წერტილის შემდეგ ციფრთა რაოდენობა;
- cin.get და cout.put ფუნქციები საშუალებას იძლევა შეტანილ ან გამოტანილ იქნას ერთი სიმბოლო;
- cin.getline ფუნქცია სტრიქონის კლავიატურიდან წაკითხვის საშუალებას იძლევა.

ლაბორატორიული სამუშაო 15
თემა: შეტანა-გამოტანის ფაილური ოპერაციები

პროგრამების გართულებასთან ერთად პროგრამებმა უნდა შეინახონ და მიიღონ ინფორმაცია ფაილების გამოყენებით.

მოცემულ მასალაში განხილულია შემდეგი ძირითადი კონცეფციები:

- გამომავალი ფაილური ნაკადის გამოყენებით თქვენ შეგიძლიათ ჩაწეროთ ინფორმაცია ფაილში ჩასმის ოპერატორის (<<) დახმარებით;
- შემავალი ფაილური ნაკადის გამოყენებით თქვენ შეგიძლიათ ფაილში შენახული ინფორმაციის წაკითხვა ამოღების ოპერატორის (>>) დახმარებით;
- ფაილის გახსნისა და დახურვისათვის გამოიყენებთ ფაილური კლასების მეთოდებს;
- ფაილური მონაცემების წაკითხვისა და ჩაწერისათვის შეგიძლიათ გამოიყენოთ ჩასმის და ამოღების ოპერატორები, ასევე ფაილური კლასების ზოგიერთი მეთოდები.

შეტანა ფაილურ ნაკადში

სათაო ფაილი *iostream.h* განსაზღვრავს *cout* გამომავალ ნაკადს. ანალოგიურად, სათაო ფაილი *fstream.h* განსაზღვრავს გამომავალი ფაილური ნაკადის კლასს, რომლის სახელია *ofstream.* ამ კლასის ობიექტების გამოყენებით პროგრამებს შეუძლია შეასრულოს გამოტანა ფაილში. დასაწყისისათვის თქვენ უნდა გამოაცხადოთ *ofstream* ტიპის ობიექტი, სადაც მოთხოვნილი გამომავალი ფაილის სახელი

მიეთითება სიმბოლური სტრიქონის სახით, როგორც ქვემოთ არის ნაჩვენები:

ofstream file_object("FILENAME.TXT");

ofstream ტიპის ობიექტის გამოცხადებისას თუ თქვენ მიუთითებთ ფაილის სახელს, დისკზე შეიქმნება ახალი ფაილი მითითებული სახელის გამოყენებით ან გადაეწერება ამავე სახელის ფაილს თუ იგი უკვე არსებობს დისკზე. ქვემოთ წარმოდგენილი პროგრამა ქმნის *ofstream* ტიპის ობიექტს და შემდეგ გამოიყენებს ჩასმის ოპერატორს ტექსტის რამდენიმე სტრიქონის შესატანად BOOKINFO.TXT ფაილში:

```
#include "stdafx.h"
#include <fstream>
#include<iostream>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
            ofstream  book_file("d:\BOOKINFO.TXT");
   book_file << "vswavlobT daprogramebas  C++ enaze, " <<
"meore redaqcia" << endl;
   book_file << "Jamsa Press" << endl;
   book_file << "22.95" << endl;
   system("pause");
}
```

მოცემულ მაგალითში პროგრამა ხსნის BOOKINFO.TXT ფაილს და შემდეგ ჩაწერს სამ სტრიქონს ფაილში:

ვსწავლობთ დაპროგრამებას C++ ენაზე, მეორე რედაქცია

Jamsa Press

22.95

წაკითხვა შემავალი ფაილური ნაკადიდან

წაკითხვა ფაილიდან ხდება *ifstream* ტიპის ობიექტების გამოყენებით. ანალოგიურად თქვენ ქმნით ობიექტს, რომელსაც პარამეტრად გადასცემთ მოთხოვნილი ფაილის სახელს:

```
ifstream input_file("filename.EXT");
```

ქვემოთ წარმოდგენილი პროგრამა ხსნის BOOKINFO.TXT ფაილს, რომელიც თქვენ შექმენით წინა პროგრამაში და კითხულობს, შემდეგ კი ასახავს ფაილის პირველ-სამ ელემენტს:

```
#include "stdafx.h"
#include <fstream>
#include<iostream>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
 ifstream input_file("d:\BOOKINFO.TXT") ;
   char one[64], two[64], three[64];
   input_file >> one;
   input_file >> two;
   input_file >> three;
   cout << one << endl;
   cout << two << endl;
   cout << three << endl;
   system("pause");
     }
```

მოცემული პროგრამის გაშვების შემდეგ იგი ასახავს ფაილის პირველ-სამ სიტყვას. *cin* მსგავსად შემავალი ფაილური ნაკადები გამოიყენებენ ცარიელ სიმბოლოებს რათა განისაზღვროს სად მთავრდება ერთი მნიშვნელობა და სად იწყება ახალი. ეკრანზე გამოჩნდება შედეგი:

მთელი სტრიქონის წაკითხვა

როგორც წინა მასალაში განვიხილეთ კლავიატურიდან მთელი სტრიქონის წასაკითხად გამოიყენება *cin.getline*. მსგავსად *ifstream* ტიპის ობიექტებს შეუძლიათ *getline* გამოიყენონ ფაილური შეტანის სტრიქონების წასაკითხად. ქვემოთ წარმოდგენილი პროგრამა გამოიყენებს *getline*-ს BOOKINFO.TXT ფაილიდან სამივე სტრიქონის წასაკითხად:

```
#include "stdafx.h"
#include <fstream>
#include<iostream>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
ifstream input_file("d:\BOOKINFO.TXT");
   char one[64], two[64], three [64] ;
   input_file.getline(one, sizeof(one)) ;
   input_file.getline(two, sizeof(two));
   input_file.getline(three, sizeof(three)) ;
   cout << one << endl;
   cout << two << endl;
   cout << three << endl;
   system("pause");

    }
```

მოცემულ შემთხვევაში პროგრამა წარმატებით კითხულობს ფაილის შინაარსს, რადგან მან იცის, რომ ფაილი შეიცავს სამ სტრიქონს. ხშირ შემთხვევაში პროგრამამ არ იცის რამდენი სტრიქონია ფაილში. ასეთ

შემთხვევებში წაკითხვა გაგრძელდება მანამ, სანამ არ შეხვდება ფაილის დასასრული.

ფაილის დასასრულის განსაზღვრა

პროგრამაში ჩვეულებრივი ფაილური ოპერაცია არის ფაილის შინაარსის წაკითხვა, სანამ არ შეხვდება ფაილის დასასრული. ფაილის დასასრულის განსაზღვრისათვის პროგრამაში გამოიყენება eof ფუნქცია. ეს ფუნქცია აბრუნებს 0 მნიშვნელობას, თუ ფაილის დასასრული ჯერ კიდევ არ შეხვდა, ხოლო 1-ს თუ ფაილის დასასრული შეხვდა. *while* ციკლის გამოყენებით პროგრამას შეუძლია განუწყვეტლივ წაიკითხოს ფაილის შინაარსი, სანამ არ იპოვის ფაილის დასასრულს, როგორც ქვემოთ არის ნაჩვენები:

```
    while (! input_file.eof())
    {
  // ოპერატორები
}
```

მოცემულ შემთხვევაში პროგრამა აგრძელებს ციკლის შესრულებას, სანამ *eof* ფუნქცია აბრუნებს მცდარს. ქვემოთ წარმოდგენილი პროგრამა ფაილს კითხულობს დასასრულის მიღწევამდე:

```
#include "stdafx.h"
#include <fstream>
#include<iostream>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{   ifstream  input_file("d:\BOOKINFO.TXT");
   char line[64];
   while (! input_file.eof())
   {
      input_file.getline(line, sizeof(line));
```

```
      cout << line << endl;
   }
   system("pause");
     }
```

ანალოგიურად, შემდეგი პროგრამა კითხულობს ფაილის შინაარსს თითო ჯერზე თითო სიტყვას, სანამ არ შეხვდება ფაილის დასასრული:

```
#include "stdafx.h"
#include <fstream>
#include<iostream>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{     ifstream input_file("d:\BOOKINFO.TXT");
   char word[64] ;
   while (! input_file.eof())
   {
      input_file >> word;
        cout << word << endl; }
   system("pause");
}
```

შედეგი:

```
vswavlobT
daprogramebas
C++
enaze,
meore
redaqcia
Jamsa
Press
22.95
```

ხოლო შემდეგი პროგრამა *get* ფუნქციის გამოყენებით კითხულობს ფაილის შინაარსს თითო ჯერზე თითო სიმბოლოს, სანამ არ შეხვდება ფაილის დასასრული:

```
#include "stdafx.h"
#include <fstream>
#include<iostream>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{        ifstream input_file("d:\BOOKINFO.TXT");
   char letter;
   while (! input_file.eof())
   {
      letter = input_file.get();
      cout << letter;
   }
   system("pause");
}
```

შედეგი:

```
vswavlobT daprogramebas  C++ enaze, meore redaqcia
Jamsa Press
22.95
 Press any key to continue . . .
```

ფაილური ოპერაციების შესრულებისას შეცდომების შემოწმება

აქამდე წარმოდგენილი პროგრამები გვთავაზობდნენ, რომ შეტანა-გამოტანის ფაილური ოპერაციების დროს არ ხდება შეცდომები. მაგრამ ეს ყოველთვის ასე არ არის. მაგალითად, თუ ხსნით ფაილს შესატანად, პროგრამამ უნდა შეამოწმოს, რომ ფაილი არსებობს. ანალოგიურად, თუ პროგრამას შეაქვს მონაცემები ფაილში, აუცილებელია დარწმუნდეთ, რომ ოპერაციამ წარმატებით ჩაიარა. შეცდომებზე თვალყურის მიდევნების მიზნით შესაძლებელია ფაილური

ობიექტის fail ფუნქციის გამოყენება. თუ ფაილური ოპერაციის პროცესში შეცდომები არ იყო, ფუნქცია დააბრუნებს მცდარს(0). მხოლოდ თუ შეხვდა შეცდომა fail ფუნქცია დააბრუნებს ჭეშმარიტს. მაგალითად, თუ პროგრამა ხსნის ფაილს, მან უნდა გამოიყენოს fail ფუნქცია, რათა განისაზღვროს მოხდა თუ არა შეცდომა, როგორც ქვემოთ არის ნაჩვენები:

```
      ifstream input_file("FILENAME.DAT");
if (input_file.fail())
      {
  cerr << " FILENAME.EXT გახსნის შეცდომა" << endl;
  exit(1);
}
```

ამდენად, პროგრამები უნდა დარწმუნდნენ, რომ წაკითხვის და ჩაწერის ოპერაციებმა ჩაიარა წარმატებით. შემდეგი პროგრამა გამოიყენებს fail ფუნქციას სხვადასხვა შეცდომითი სიტუაციების შესამოწმებლად:

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{   char line[256] ;
   ifstream input_file("BOOKINFO.TXT") ;
   if (input_file.fail()) cerr << "BOOKINFO.TXT failis
gaxsnis Secdoma"<< endl;
   else
   { while ((! input_file.eof()) && (! input_file.fail()))
      {
          input_file.getline(line, sizeof(line)) ;
         if (! input_file.fail()) cout << line << endl;
      }
   }
      system("pause");}
```

ფაილის დახურვა

ფაილთან მუშობის დასრულების შემდეგ იგი უნდა დაიხუროს, რისთვისაც გამოიყენება *close* ფუნქცია, როგორც ქვემოთ არის ნაჩვენები:

```
input_file.close ();
```

პროგრამის მიერ შეტანილი ყველა ინფორმაცია ინახება ფაილში.

ფაილის გახსნის მართვა

არსებული ფაილის ბოლოში ინფორმაციის დასამატებლად საჭიროა ფაილი გაიხსნას დამატების რეჟიმში, რისთვისაც ფაილის გახსნისას უნდა მიეთითოს მეორე პარამეტრი, როგორც ქვემოთ არის ნაჩვენები:

```
ifstream output_file("FILENAME.EXT", ios::app);
```

მოცემულ შემთხვევაში *ios::app* პარამეტრი უჩვენებს ფაილის გახსნის რეჟიმს. პროგრამების სირთულის მიხედვით ფაილის გახსნის რეჟიმისათვის გამოიყენება მნიშვნელობათა შერწყმა, რომელიც წარმოდგენილია ცხრილში:

გახსნის რეჟიმი	დანიშნულება
ios::app	ხსნის ფაილს დამატების რეჟიმში, ფაილზე მიმთითებელს განათავსებს ფაილის ბოლოში
ios::ate	ფაილზე მიმთითებელს განათავსებს ფაილის ბოლოში
ios::in	მიუთითებს ფაილის გახსნას შესატანად
ios::nocreate	თუ მითითებული ფაილი არ არსებობს, არ შეიქმნას ფაილი და დააბრუნოს შეცდომა
ios::noreplace	თუ ფაილი არსებობს, გახსნის ოპერაცია უნდა შეწყდეს და დააბრუნოს შეცდომა
ios::out	მიუთითებს ფაილის გახსნას გამოსატანად
ios::trunc	ჩამოყრის(გადაწერს) შინაარსს არსებული ფაილიდან

ფაილის გახსნის შემდეგი ოპერაცია ხსნის ფაილს გამოსატანად, *ios::noreplace* რეჟიმის გამოყენებით, რათა თავიდან აიცილოს არსებულ ფაილზე გადაწერა:

```
ifstream output_file("FIlename.EXT", ios::out | ios::noreplace);
```

წაკითხვის და ჩაწერის ოპერაციების შესრულება

მოცემულ მასალაში წარმოდგენილი ყველა პროგრამა ასრულებდა ფაილურ ოპერაციებს სიმბოლურ სტრიქონებზე. არანაკლებ საჭიროა მასივების და სტრიქონების წაკითვა და ჩაწერაც. ამისათვის უნდა იქნას გამოყენებული *read* და *write* ფუნქციები. ამ ფუნქციების გამოყენებისას უნდა მიუთითოთ მონაცემთა ბუფერი, რომელშიც მონაცემები წაიკითხება ან საიდანაც ისინი ჩაიწერება, ასევე ბუფერის სიგრძე ბაიტებში, როგორც ქვემოთ არის ნაჩვენები:

```
input_file.read(buffer, sizeof(buffer)) ;
output_file.write(buffer, sizeof(buffer));
```

შემდეგი პროგრამა გამოიყენებს *write* ფუნქციას სტრუქტურის შემადგენლობის EMPLOYEE.DAT ფაილში ჩასაწერად:

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{      struct employee
   {
      char name[64];
      int age;
      float salary;
```

```
   } worker = { "Happy Jamsa", 33, 25000.0 };
   ofstream emp_file("d:\EMPLOYEE.DAT") ;
   emp_file.write((char *) &worker, sizeof(employee));
      system("pause");
}
```

write ფუნქცია ჩვეულებრივ იღებს მიმთითებელს სიმბოლურ სტრიქონზე. სიმბოლოები *(char *)* წარმოადგენს ტიპებში მოყვანის ოპერატორს, რომელიც კომპილატორს უკეთებს ინფორმირებას, რომ თქვენ გადასცემთ სხვა ტიპზე მიმთითებელს. მსგავსად, მომდევნო პროგრამა გამოიყენებს *read* მეთოდს ფაილიდან თანამშრომლის შესახებ ინფორმაციის წასაკითხად:

```
#include "stdafx.h"
#include <fstream>
#include<iostream>
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    struct employee
   {
      char name [64] ;
      int age;
      float salary;
   } worker = { "Happy Jamsa", 35, 2000.0 };
   ifstream emp_file("d:\EMPLOYEE1.DAT");
   emp_file.read((char *) &worker, sizeof(employee));
   cout << worker.name << endl;
   cout << worker.age << endl;
   cout << worker.salary << endl;
      system("pause");
}
```

დასკვნა:

- *fstream.h* სათაო ფაილი განსაზღვრავს *ifstream* და *ofstream* კლასებს, რომელთა დახმარებით პროგრამას შეუძლია შეასრულოს ფაილური შეტანა-გამოტანის ოპერაციები;
- ფაილის შესატანად ან გამოსატანად გახსნისათვის ობიექტი უნდა გამოაცხადოთ *ifstream* ან *ofstream* ტიპად, ამ ობიექტის კონსტრუქტორზე მოთხოვნილი ფაილის სახელის გადაცემით;
- მას შემდეგ რაც პროგრამაში გაიხსნება ფაილი შეტანის ან გამოტანისათვის, შესაძლებელია მონაცემების წაკითხვა ან ჩაწერა ამოღების (<<) ან ჩასმის (>>) ოპერატორების გამოყენებით;
- პროგრამას შეუძლია შეასრულოს სიმბოლოების შეტანა ან გამოტანა ფაილში ან ფაილიდან, *get* და *put* ფუნქციების გამოყენებით;
- პროგრამას შეუძლია ფაილიდან წაიკითხოს მთელი სტრიქონი *getline* ფუნქციის გამოყენებით;
- უმეტესი პროგრამები კითხულობენ ფაილის შინაარსს, ვიდრე არ შეხვდება ფაილის დასასრული. ფაილის დასასრული შეიძლება განისაზღვროს *eof* ფუნქციის დახმარებით;
- როდესაც პროგრამა ასრულებს ფაილურ ოპერაციებს, მან უნდა შეამოწმოს ყველა ოპერაციის მდგომარეობა, რათა დარწმუნდეს, რომ ოპერაციები შესრულებულია წარმატებულად. შეცდომების შესამოწმებლად შეიძლება გამოყენებულ იქნას *fail* ფუნქცია;
- როდესაც ფაილში საჭიროა მასივების ან სტრუქტურების შეტანა ან გამოტანა, შეიძლება გამოყენებულ იქნას *read* და *write* მეთოდები;
- ფაილთან მუშაობის დასრულების შემდეგ საჭიროა მისი დახურვა *close* ფუნქციის გამოყენებით.

ტესტები

თემა: კლასები და ობიექტები

შეკითხვა 1

რომელი კონსტრუქციით შეიძლება იყოს კლასი განსაზღვრული:

a) კლასის_ტიპი კლასის_სახელი {კლასის_წევრების_სია}
b) კლასის_სახელი {კლასის წევრების სია}
c) კლასის_სახელი {კლასის მონაცემები, კლასის მეთოდები}
d) კლასის_ტიპი {კლასის მონაცემები, კლასის ფუნქციები}

შეკითხვა 2

კლასის ტიპის აღსანიშნად რომელი მომსახურე სიტყვა გამოიყენება:

a) class
b) object
c) union
d) type

შეკითხვა 3

ჩამოთვლილთაგან რომელია მონაცემთა აბსტრაქტული ტიპი?:

a) int
b) double
c) string
d) Class

შეკითხვა 4

კლასის ტიპი ქვემოთ ჩამოთვლილთაგან რომელს მიეკუთვნება:

a) მომსახურე სიტყვას
b) იდენტიფიკატორს
c) სპეციფიკატორს
d) კლასიფიკატორს

შეკითხვა 5

კლასის სახელი ქვემოთ ჩამოთვლილთაგან რომელს მიეკუთვნება:

a) იდენტიფიკატორს
b) სპეციფიკატორს
c) კლასიფიკატორს
d) მომსახურე სიტყვას

შეკითხვა 6

კლასის წევრების სიაში შეიძლება იყოს განსაზღვრა და აღწერა:

a) ტიპიზებული მონაცემების
b) კლასის ფუნქციების
c) კლასის ობიექტების
d) კლასის კლასიფიკატორების

შეკითხვა 7

რომელი კონსტრუქცია გამოიყენება კლასის ობიექტის ასაღწერად:

a) კლასის_სახელი ობიექტის_სახელი
b) კლასის_სახელი ფუნქციის_სახელი
c) ობიექტის_სახელი კლასის_სახელი
d) ობიექტის_სახელი მეთოდის_სახელი

შეკითხვა 8

ქვემოთჩამოთვლილთაგან რომელი გამოიყენება ობიექტის მონაცემებზე მიმართვისთვის:

a) ობიექტის_სახელი . მონაცემის_სახელი
b) მონაცემის_სახელი.კლასის_სახელი
c) მონაცემის_სახელი.ობიექტის_სახელი
d) მონაცემის_სახელი.ფუნქციის_სახელი

შეკითხვა 9

ფრაგმენტში მოცემულია complex კლასის ობიექტები:

complex x1, x2;

რომელია სწორი მიმართვა ობიექტის მონაცემებზე:

a) x1.re = 1.24;
b) x2.im = 2.3;
c) x1_re=1.24;
d) x2_im=2.3;

შეკითხვა 10

ფრაგმენტში მოცემულია complex კლასის ობიექტები:

```
complex x1,x2;
```

რომელია სწორი მიმართვა ობიექტის ფუნქციებზე:

```
a) x2.set(5.1,1.7);
b) x1.print();
c) x1*set(5.1,1.7)
d) x2*print();
```

შეკითხვა 11

ქვემოთჩამოთვლილთაგან რომელი გამოიყენება ობიექტის კომპონენტზე მიმართვისთვის:

a) მიმთითებელი_ობიექტზე->კომპონენტის_სახელი
b) მიმთითებელი_ობიექტზე->კლასის_სახელი
c) მიმთითებელი ობიექტზე->ობიექტის_სახელი
d) სამივე ზემოთ ჩამოთვლილი

შეკითხვა 12

ფრაგმენტში მოცემულია complex კლასის ობიექტზე მიმთითებელი:

complex *point = &x1;

რომელია სწორი მიმართვა ობიექტის მონაცემებზე:

```
a) point ->re = 1.24;
```

b) `point ->im = 2.3;`

c) point.re=1.24;

d) point.im=2.3;

შეკითხვა 13

ფრაგმენტში მოცემულია complex კლასის ობიექტზე მიმთითებელი:

```
complex *point = &x1;
```

რომელია სწორი მიმართვა ობიექტის მონაცემებზე:

a) `point ->print();`

b) point.print();

c) point_print():

d) point&print():

შეკითხვა 14

კომპონენტების ხედვის (ხილვადობის) თვალსაზრისით ჩამოთვლილთაგან წვდომის რომელი სპეციფიკატორები გამოიყენება:

a) public

b) private

c) class

d) struct

შეკითხვა 15

რომელია საერთო წვდომის სპეციფიკატორი:

a) public

b) private

c) protected

d) სამივე

შეკითხვა 16

რომელია კერძო წვდომის სპეციფიკატორი:

a) private

b) public
c) protected
d) სამივე

თემა: კლასის ფუნქციები

შეკითხვა 17

რომელი ტერმინი გამოიყენება კლასში განსაზღვრული ფუნქციისთვის?:

a) ცვლადი წევრი
b) ფუნქცია წევრი
c) კლასის ფუნქცია
d) კლასიკური ფუნქცია

შეკითხვა 18

ქვემოთჩამოთვლილთაგან რომელი გამოიყენება ობიექტის ფუნქციაზე მიმართვისთვის:

a) ობიექტის_სახელი . ფუნქციის_სახელი
b) ფუნქციის_სახელი.კლასის_სახელი
c) ფუნქციის_სახელი.ობიექტის_სახელი
d) ფუნქციის_სახელი.მეთოდის_სახელი

შეკითხვა 19

რომელი განსაზღვრა რთავს ნებას კომპილატორს ჩასვას არგუმენტი ფუნქციის გამოძახებისას, თუ იგი არ არის განსაზღვრული?:

a) გამოძახება მნიშვნელობით
b) გამოძახება ბმულით
c) არგუმენტი ჩუმათობის პრინციპით
d) მიმთითებლის მოთხოვნა

შეკითხვა 20

რომელია ფუნქციის მუდმივად გამოცხადების სწორი ხერხი?:

a) const int ShowData(void) { /* statements */ }
b) int const ShowData(void) { /* statements */ }
c) int ShowData(void) const { /* statements */ }
d) არცერთი პასუხი არ არის სწორი

შეკითხვა 21

რომელია სწორი გამონათქვამი ჩაშენებულ ფუნქციებთან მიმართებით?:

a) აჩქარებს შესრულებას
b) ანელებს შესრულებას
c) ზრდის კოდის ზომას
d) არცერთი არ არის სწორი

შეკითხვა 22

ქვემოთჩამოთვლილთაგან რომელს აქვს წვდომა კლასის კერძო მონაცემებზე?:

a) პროგრამის ნებისმიერ ფუნქციას
b) პროგრამის ყველა გლობალურ ფუნქციას
c) მოცემული კლასის ნებისმიერ ფუნქცია წევრს
d) მოცემული კლასის მხოლოდ საზოგადო ფუნქცია წევრებს

თემა: კლასის კომპონენტებზე ღია წვდომა

შეკითხვა 23

რა მიიღება შემდეგი პროგრამის შესრულების შედეგად?

```
class Bix
{
    public:
      int x;
};
```

```
int _tmain(int argc, _TCHAR* argv[])
{ Bix *p = new Bix();

    (*p).x = 10;
    cout<< (*p).x << "," << p->x << "," ;
    p->x = 20;
    cout<< (*p).x << "," << p->x ;
    return 0;
}
```

შეკითხვა 24

რა მიიღება შემდეგი პროგრამის შესრულების შედეგად?

```
class soldier
{
public:
  int x,y;

  void move (int dx, int dy)
  {
    x += dx;
    y += dy;
    cout<<x<<",";
    cout<<y;
  }
};
int _tmain(int argc, _TCHAR* argv[])
{ soldier a;
    a.x=3; a.y=4;
   a.move(1,2);
    return 0;
}
```

შეკითხვა 25

რა მიიღება შემდეგი პროგრამის შესრულების შედეგად?

```
class  figure
{
public:
  int s1, s2;

  void element (int a, int b)
  {s1=2*(a+b);
   s2=a*b;
        cout<<s1<<","<<s2;

 }
};
int _tmain(int argc, _TCHAR* argv[])
{ figure d;
       d.element(4,6);
    system("pause");}
```

შეკითხვა 26

რას გამოიტანს პროგრამა შესრულების შედეგად?:

```
class IndiaBix
{
    public:
    void GetData(char *s, int x, int y )
    {
        int i = 0;
        for (i = x-1; y>0; i++)
        {
            cout<< s[i];
            y--;
        }
    }
};
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    IndiaBix objBix;
    objBix.GetData((char*)"welcome!",          1,
3);system("pause");
    return 0;
}
```

შეკითხვა 27

რა მიიღება პროგრამის შესრულების შედეგად?

```
class soldier
{
public:
  int x,y;

  void move (int dx, int dy)
  {
    x -= dx;
    y -= dy;
    cout<<x<<",";
    cout<<y;
  }
};
int _tmain(int argc, _TCHAR* argv[])
 { soldier a;
    a.x=3; a.y=4;
   a.move(1,2);
    system("pause");}
```

შეკითხვა 28

რას გამოიტანს პროგრამა შესრულების შედეგად?:

```
class inform
```

```
{
    public:
    void GetData(char *s, int x, int y )
    {
        int i = 0;
        for (i = x-1; y>0; i++)
        {
            cout<< s[i];
            y--;
        }
    }
};

int _tmain(int argc, _TCHAR* argv[])
 {
inform  objOne;
    objOne.GetData((char*)"student!", 1, 4);
      system("pause");return 0;
}
```

თემა: კლასის დახურული და ღია კომპონენტები

შეკითხვა 29

მოცემულ ფრაგმენტში კლასის რომელი ტიპის წვდომას მიეკუთვნება a ცვლადი?:

```
class  abc {
int a;
public:   void F(){ a++; }
};

int _tmain(int argc, _TCHAR* argv[])
 {abc  dd;
dd.F();
```

……

}

a. დახურული
b. ღია
c. დაცული
d. სამივე პასუხი სწორია

შეკითხვა 30

მოცემულ ფრაგმენტში კლასის რომელი ტიპის წვდომას მიეკუთვნება `void F()` ფუნქცია?:

```
class  abc {
int a;
public:   void F(){ a++; }
};

int _tmain(int argc, _TCHAR* argv[])
 {abc  dd;
dd.F();
```

……

}

a. დახურული
b. ღია
c. დაცული
d. სამივე პასუხი სწორია

შეკითხვა 31

მოცემული ფრაგმენტიდან გამომდინარე რომელია სწორი პასუხი?

```
class soldier
{
public:
  int attack();
};
```

```
int soldier::attack()
{
  return 0;
}
```

a) მეთოდი განსაზღვრულია კლასის შიგნით
b) მეთოდი განსაზღვრულია კლასის გარეთ
c) მეთოდი არ არის განსაზღვრული
d) არცერთი პასუხი არ არის სწორი

შეკითხვა 32

მოცემული პროგრამის მიხედვით ქვემოთჩამოთვლილ-თაგან რომელია სწორი პასუხი?:

```
class soldier
{
private:
  int x, y;
public:
  void move (int dx, int dy)
  {
    x += dx;
    y += dy;
    cout<<x<<endl;
    cout<<y<<endl;
  }
};
int _tmain(int argc, _TCHAR* argv[])
  { soldier a;
   a.x=3;
   a.y=4;
   a.move(1,2);
    system("pause");}
```

a. 4 6
b. კომპილატორი მოგვცემს შეცდომას
c. 6 4
d. 24 12

შეკითხვა 33

მოცემულია პროგრამის ფრაგმენტი

```
class some_class
{ public:
         int some_variable;
        void initialize_private(int, float);
    void show_data(void);
 private:
int key_value;
float key_number; }
.....
int _tmain(int argc, _TCHAR* argv[])
{some_class object;
.....
```

აირჩიეთ ქვემოთჩამოთვლილთაგან რომელია სწორი მიმართვა კლასის ელემენტებზე:

a. object.key_value;
b. object.some_variable = 1001;
c. object.show_data()
d. object.key_number;

შეკითხვა 34

მოცემულია პროგრამის ფრაგმენტი

```
class some_class

{ public:
         int some_variable;
        void initialize_private(int, float);
```

```
    void show_data(void);
 private:
int key_value;
float key_number; }
.............
int _tmain(int argc, _TCHAR* argv[])
{some_class object;
.....................
```

აირჩიეთ ქვემოთჩამოთვლილთაგან რომელია არასწორი მიმართვა კლასის ელემენტებზე:

a. object.key_value;
b. object.some_variable = 1001;
c. object.show_data()
d. object.key_number;

შეკითხვა 35

მოცემულია პროგრამის ფრაგმენტი

```
class employee
{
public:
   int assign_values(char *, long, float);
   void show_employee(void);
   int change_salary(float);
   long get_id(void);
private:
   char name [64];
   long employee_id;
   float salary;
};
.........................
int _tmain(int argc, _TCHAR* argv[])
{employee  worker;
.....................
```

აირჩიეთ ქვემოთჩამოთვლილთაგან რომელი ბრძანებაა სწორი:

a) worker.employee_id=1100;
b) worker.show_employee();
c) worker.slary=12000;
d) worker.name=’name’;

შეკითხვა 36

მოცემულია პროგრამის ფრაგმენტი

```
class employee
{
public:
   int assign_values(char *, long, float);
   void show_employee(void);
   int change_salary(float);
   long get_id(void);
private:
   char name [64];
   long employee_id;
   float salary;
};
……….
int _tmain(int argc, _TCHAR* argv[])
{employee  worker;
……….
```

აირჩიეთ ქვემოთჩამოთვლილთაგან რომელი ბრძანება არ არის სწორი?:

a) worker.show_employee();
b) worker.employee_id=1100;
c) worker.slary=12000;
d) worker.name=’name’;

შეკითხვა 37

რა შედეგს მოგვცემს შემდეგი პროგრამა?:

```
class IndiaBix
{
    int val;
    public:
    void SetValue(char *str1, char *str2)
    {
        val = strcspn(str1, str2);
    }
    void ShowValue()
    {
        cout<< val;
    }
};

int _tmain(int argc, _TCHAR* argv[])
 {
    IndiaBix objBix;
    objBix.SetValue((char*)"India",
(char*)"Bix");
    objBix.ShowValue(); system("pause");
    return 0;
}
```

შეკითხვა 38

რა შედეგს მოგვცემს შემდეგი პროგრამა?:

```
class soldier
{
private:
  int x, y;
public:
  void move (int dx, int dy)
```

```
  { x=3,y=4;
    x += dx;
    y += dy;
    cout<<x<<",";
    cout<<y;
  }
};
int _tmain(int argc, _TCHAR* argv[])
 { soldier a;
   a.move(1,2); system("pause");}
```

შეკითხვა 39

რა შედეგს მოგვცემს შემდეგი პროგრამა?:

```
class one
{
private:
  int a, b;
public:
  void more (int x, int y)
  { a=4,b=7;
    if (a>b) {a*=x;b*=y;} else {a+=x; b+=y;}
    cout<<a<<",";
    cout<<b;
  }
};
int _tmain(int argc, _TCHAR* argv[])
 { one a;
   a.more(1,2); system("pause");}
```

შეკითხვა 40

რა შედეგს მოგვცემს შემდეგი პროგრამა?:

```
class MyClass
{
  private:
```

```
  int a, b;
public:
  void per(int x, int y)
  { int a=5, b=6, p;
  p=a*x+b*y;
    cout<<p;
      }
};
int _tmain(int argc, _TCHAR* argv[])
 { MyClass  a;
   a.per(3,4); }
```

შეკითხვა 41

რა შედეგს მოგვცემს შემდეგი პროგრამა?:

```
class MyClass
{
  int a, b;
public:
  void teri(int x, int y)
  { int a=4, b=6,p;
  p=x+y+a*b;
    cout<<p;
      }
};
int _tmain(int argc, _TCHAR* argv[])
 { MyClass  m;
   m.teri(5,8); system("pause");}
```

თემა: ფუნქციების გადატვირთვა

შეკითხვა 42

ფუნქციების გადატვირთვა ნიშნავს რამდენიმე ფუნქციის განსაზღვრას, რომელთაც აქვთ:

a) ერთნაირი სახელები და განსხვავებული პარამეტრები
b) სხვადასხვა სახელები და სხვადასხვა პარამეტრები

c) სხვადასხვა სახელები და ერთნაირი პარამეტრები
d) ერთნაირი სახელები და ერთნაირი პარამეტრები

შეკითხვა 43

გადატვირთული ფუნქციების პარამეტრების ნაკრები შეიძლება განსხვავდებოდნენ:

a) მხოლოდ ტიპებით
b) მხოლოდ რაოდენობით
c) განლაგების მიმდევრობით, რაოდენობით, ტიპებით
d) მხოლოდ განლაგების მიმდევრობით

შეკითხვა 44

რომელია სწორი გამონათქვამი ფუნქციების გადატვირთვასთან მიმართებით?:

a) არგუმენტების ტიპები განსხვავებულია
b) არგუმენტების რაოდენობა განსხვავებულია
c) არგუმენტების რაოდენობა იგივეა
d) არგუმენტების ტიპები იგივეა

შეკითხვა 45

მოცემული ფრაგმენტის მიხედვით, ჩამოთვლილთაგან რომელია სწორი განსაზღვრა `function` ფუნქციასთან მიმართებით?:

```
int function (int x)
{
  return 0;
}

int function (float x)
{
```

```
    return 0;
}
```

a) გადატვირთული ფუნქცია
b) ერთნაირი ფუნქცია
c) სხვადასხვა პარამეტრიანი ფუნქცია
d) განსხვავებული ფუნქცია

შეკითხვა 46

მოცემული ფრაგმენტის მიხედვით, ჩამოთვლილთაგან რომელია სწორი განსაზღვრა `areaRectangle` ფუნქციასთან მიმართებით?:

```
float areaRectangle(float, float)
{
 return a * b;
}
float areaRectangle(float a_m, float a_sm, float b_m,
float b_sm)
{
  return (a_m * 100 + a_sm) * (b_m * 100 + b_sm);
}
```

a) გადატვირთული ფუნქცია, რომლის პარამეტრების ნაკრები განსხვავდება ტიპებით
b) გადატვირთული ფუნქცია, რომლის პარამეტრების ნაკრები არ განსხვავდება რაოდენობით
c) გადაუტვირთავი ფუნქცია, რომლის პარამეტრები არ განხვავდება რაოდენობით
d) გადატვირთული ფუნქცია, რომლის პარამეტრების ნაკრები განსხვავდება რაოდენობით

შეკითხვა 47

რა შედეგს მოგვცემს მოცემული ფრაგმენტი?:

```
int function (int x)
```

```
{
  return 0;
}

int function (int x)
{
  return 1;
}
```

a) კომპილატორი მოგვცემს შეცდომას
b) დააბრუნებს x მნიშვნელობას
c) დააბრუნებს 0 მნიშვნელობას
d) დააბრუნებს 1 მნიშვნელობას

შეკითხვა 48

მოცემული ფრაგმენტის მიხედვით როგორ შეიძლება დავახასიათოთ max ფუნქცია?:

```
int max(int  num1, int  num2)
 { if (num1 > num2)
 return  num1;
return  num2; }

double max(double num1, double num2)
 { if (num1 > num2)
 return num1;
return num2; }
```

a) გადატვირთული ფუნქცია, რომლის პარამეტრების ნაკრები განსხვავდება რაოდენობით
b) გადატვირთული ფუნქცია, რომლის პარამეტრების ნაკრები განსხვავდება ტიპებით
c) გადაუტვირთავი ფუნქცია, რომლის პარამეტრები განხვავდება რაოდენობით
d) გადატვირთული ფუნქცია, რომლის პარამეტრების ნაკრები განსხვავდება მიმდევრობით

შეკითხვა 49

მოცემული ფრაგმენტის მიხედვით რომელია სწორი დახასიათება `repchar` ფუნქციასთან მიმართებით?:

```
void repchar() {
    for (int j=0; j<45; j++)
        cout << '*';
    cout << endl;
}

void repchar(char ch) {
    for (int j=0; j<45; j++)
        cout << ch;
    cout << endl;
}

void repchar(char ch, int n) {
    for (int j=0; j<n; j++)
        cout << ch;
    cout << endl;
}
```

a) გადატვირთული ფუნქცია არგუმენტების ერთნაირი რაოდენობით

b) ერთსახელიანი ფუნქცია არგუმენტების ერთნაირი რაოდენობით

c) გადაუტვირთავი ფუნქცია არგუმენტების ერთნაირი რაოდენობით

d) გადატვირთული ფუნქცია არგუმენტების განსხვავებული რაოდენობით

შეკითხვა 50

მოცემული ფრამგენტის მიხედვით, როგორ გამოვიძახოთ გადატვირთული ფუნქცია, რომელიც დაბეჭდავს 45 *-ს?:

```
void repchar() {
    for (int j=0; j<45; j++)
        cout << '*';
    cout << endl;
}

void repchar(char ch) {
    for (int j=0; j<45; j++)
        cout << ch;
    cout << endl;
}

void repchar(char ch, int n) {
    for (int j=0; j<n; j++)
        cout << ch;
    cout << endl;
}
```

a) `repchar();`
b) `repchar('=');`
c) `repchar('+',30);`
d) repchar(45, '*');

შეკითხვა 51

მოცემული ფრაგმენტის მიხედვით `int imax = max(1, 10);` ბრძანება გადატვირთული ფუნქციის რომელ ვარიანტს გამოიძახებს?:

```
int max(int  num1, int  num2)
 { if (num1 > num2)
 return  num1;
return  num2; }

double max(double num1, double num2)
 { if (num1 > num2)
 return num1;
return num2; }
```

a) double max(double num1, double num2)
b) ორივეს ერთდროულად
c) int max(int num1, int num2)
d) არცერთს არ გამოიძახებს

შეკითხვა 52

მოცემული ფრაგმენტის მიხედვით `double dmax = max(1.0, 20.0);` ბრძანება გადატვირთული ფუნქციის რომელ ვარიანტს გამოიძახებს?:

```
int max(int  num1, int  num2)
 { if (num1 > num2)
 return  num1;
return  num2; }

double max(double num1, double num2)
 { if (num1 > num2)
 return num1;
return num2; }
```

a) int max(int num1, int num2)
b) double max(double num1, double num2)
c) ორივეს ერთდროულად
d) არცერთს არ გამოიძახებს

თემა: კონსტრუქტორი

შეკითხვა 53

ჩამოთვლილთაგან აირჩიეთ სწორი განმარტება:

a) კონსტრუქტორი არის კლასის მეთოდი, რომელიც ამარტივებს კლასის მონაცემ-ელემენტების ინიციალიზებას;

b) კონსტრუქტორი არის კლასის მეთოდი, რომელიც გამოიყენება კლასის მხოლოდ ღია მონაცემებზე წვდომისათვის;
c) კონსტრუქტორი არის მეთოდი, რომელიც გამოიყენება კლასის მხოლოდ დახურულ მონაცემებზე წვდომისათვის;
d) კონსტრუქტორი არის მეთოდი, რომელიც გამოიყენება კლასის მხოლოდ დაცულ მონაცემებზე წვდომისათვის.

შეკითხვა 54

კონსტრუქტორს აქვს ისეთივე სახელი, როგორც:

a) კლასს
b) ფუნქციას
c) მეთოდს
d) სათაო ფაილს

შეკითხვა 55

კონსტრუქტორს არა აქვს:

a) სახელი
b) დასაბრუნებელი მნიშვნელობა
c) პარამეტრები
d) მუდმივები

შეკითხვა 56

ობიექტის შექმნისას კონსტრუქტორი გამოიძახება:

a) სპეციალური ბრძანებით
b) ავტომატურად
c) სპეციალური ფუნქციით
d) სპეციალური მეთოდით

შეკითხვა 57

ობიექტის მიერ დაკავებული მეხსიერების გასათავისუფლებლად გამოიყენება:

a) დეაქტივატორი
b) სპეციფიკატორი
c) დესტრუქტორი
d) იმიტატორი

შეკითხვა 58

დესტრუქტორს აქვს იგივე სახელი, როგორც:

a) ფუნქციას
b) მეთოდს
c) კლასს
d) კონსტრუქტორს

შეკითხვა 59

რომელი სიმბოლო იწერება დესტრუქტორის წინ?:

a) &
b) ~
c) #
d) *

შეკითხვა 60

დესტრუქტორს არა აქვს:

a) დასაბრუნებელი მნიშვნელობა
b) სახელი
c) პარამეტრები
d) ~ ნიშანი

შეკითხვა 61

ჩამოთვლილთაგან რომელია სწორი გამონათქვამი?:

a) კონსტრუქტორი გამოიძახება ობიექტის დეკლარაციისას
b) კონსტრუქტორი გამოიძახება ობიექტის შესრულებისას
c) კონსტრუქტორი გამოიძახება კლასის დეკლარაციისას
d) კონსტრუქტორი გამოიძახება კლასის შესრულებისას

შეკითხვა 62

ჩამოთვლილთაგან რომელს ასრულებს კონსტრუქტორი?:

a) ახალი კლასის შენებას
b) ახალი ობიექტის შენებას
c) ახალი ფუნქციის შენებას
d) ობიექტის ინიციალიზებას

შეკითხვა 63

მოცემულ ფრაგმენტში რომელია კონსტრუქტორი?:

```
class queue {
int q[100];
int sloc, rloc;
public:
queue ();
void qput(int i);
int qget ();
}
```

a) queue ();
b) void qput(int i);
c) int q[100];
d) int qget ();

შეკითხვა 64

ზოგადად როგორი სახე აქვს ობიექტის ინიციალიზებას კონსტრუქტორის მეშვეობით?:

a) object(valuel, value2, value3);
b) class_name object(valuel, value2, value3);
c) void object(valuel, value2, value3);
d) ~ object(valuel, value2, value3);

შეკითხვა 65

რომელი კონსტრუქტორია გამოყენებული მოცემულ ფრაგმენტში?:

```
class Account {
public:
   Account();
private:
   char *_name;
   unsigned int _acct_nmbr;
   double _balance;
};
```

a) Account() კონსტრუქტორი დუმილით
b) კონსტრუქტორი პარამეტრიანი
c) double _balance ორმაგი სიზუსტის
d) unsigned int _acct_nmbr უნიშნო მთელი

შეკითხვა 66

აირჩიეთ კოსტრუქტორი გამოყენების სწორი ვარიანტი მოცემული ფრაგმენტისთვის:

```
class employee
{
public:
   employee(char *, long, float);
```

```
        void show_employee(void);
        int change_salary(float) ;
        long get_id(void);
    private:
        char name [15] ;
        long employee_id;
        float salary;
};
```

a) employee worker("Janet", 010220.22, 20303.0);
b) employee worker(0102202, 20303.0);
c) employee worker("Janet", 0102202, 20303.0);
d) employee worker(0102202.22, 20303.0);

შეკითხვა 67

მოცემული ფრაგმენტიდან გამომდინარე ჩამოთვლილთაგან რომელია სწორი გამონათქვამი?:

```
class MyClass
{
public:
    MyClass();

private:
    int x;
};

MyClass :: MyClass() : x(100)
{
}
int _tmain(int argc, _TCHAR* argv[])
{ MyClass m;

 }
```

a) შეიქმნება m ობიექტი და არ გამოიძახება კონსტრუქტორი დუმილით

b) შეიქმნება `m` ობიექტი და გამოიძახება კონსტრუქტორი დუმილით

c) შეიქმნება `MyClass`-ის `m` ობიექტი და დასრულდება პროგრამა

d) არცერთი არ არის სწორი გამონათქვამი

შეკითხვა 68

რა შედეგს მოგვცემს შემდეგი პროგრამა?:

```
class MyClass
{
public:
    MyClass();

private:
    int x;
};

MyClass :: MyClass() : x(100)
{x*=2; cout<<x;
}

int _tmain(int argc, _TCHAR* argv[])
{  MyClass m;
 }
```

a) 200

b) 100

c) 2

d) 102

შეკითხვა 69

რა შედეგს გამოიტანს შემდეგი პროგრამა?“

```
class BixTeam
{
```

```
    int x, y;
    public:
    BixTeam(int xx)
    {
        x = ++xx;
    }
    void Display()
    {
        cout<< --x << ",";
    }
};

int _tmain(int argc, _TCHAR* argv[])
 {
    BixTeam objBT(45);
    objBT.Display();
    int *p = (int*)&objBT;
    *p = 23;
    objBT.Display();system("pause");
    return 0;
}
```

შეკითხვა 70

რას გამოიტანს პროგრამა შესრულების შედეგად?:

```
class BixData
{
    int x, y, z;
    public:
    BixData(int xx, int yy, int zz)
    {
        x = ++xx;
        y = ++yy;
        z = ++zz;
    }
    void Show()
    {
```

```
        cout<< x++ << "," << y++ << "," << z++;
    }
};

int _tmain(int argc, _TCHAR* argv[])
 {
    BixData objData(1, 2, 3);
    objData.Show();system("pause");
    return 0;
}
```

შეკითხვა 71

რა მიიღება პროგრამის შესრულების შედეგად?:

```
class Bix
{
    int x, y;
    public:
    Bix(int xx, int yy)
    {
        x = xx;
        y = yy;
    }
    void show()
    {
        cout<< x * y << endl;
    }
};

int _tmain(int argc, _TCHAR* argv[])
 {
    Bix obj(10, 20);
       obj.show();system("pause");
    return 0; }
```

შეკითხვა 72

რას გამოიტანს პროგრამა შესრულების შედეგად?:

```
class IndiaBix
{
    int a;
    int b;
    public:
    IndiaBix(int c, int d)
    {       a=c; b=d;
    }
    int BixFunction()
    {   int s;
  s=(a+b)*2;
        return s;
    }
};
int _tmain(int argc, _TCHAR* argv[])
 {

    IndiaBix objBix(4,6);
    cout<< objBix.BixFunction();system("pause");
    return 0;

}
```

შეკითხვა 73

რას გამოიტანს მოცემული პროგრამა?:

```
class AB
{
    private:
    int a;
    int b;
     public:
    AB(int c, int d)
    {a=3;b=4;
        a+= c;
        b+= d;
        cout << a<<",";
```

```
        cout << b;
  } };

int _tmain(int argc, _TCHAR* argv[])
 {

AB obj1(100, 200);  system("pause");

}
```

შეკითხვა 74

რას გამოიტანს შემდეგი პროგრამა?:

```
class SomeData
{
private:
  int someNum1;
  double someNum2;
  char someSymb[18];
public:
  SomeData()
  {
        someNum1 = 111;
        someNum2 = 222.22;
        strcpy_s(someSymb, "string");
              }

  void showSomeData()
  {
        cout << someNum1<<",";
        cout << someNum2<<",";
        cout <<someSymb;
  }
};
int _tmain(int argc, _TCHAR* argv[])
{
  SomeData obj1;
  obj1.showSomeData();system("pause");
```

```
}
```

შეკითხვა 75

მოცემულია პროგრამის ფრაგმენტი:

```
class Example
{
     int x, y;
   public:
     Example(int a, int b);
};
Example :: Example(int a, int b)
{
     x = a;
     y = b;
}
```

რა მნიშვნელობები მიენიჭება x და y ცვლადებს `Example e(0,50)` ბრძანების პირობებში?:

თემა: კონსტრუქტორის გადატვირთვა

შეკითხვა 76

ჩამოთვლილთაგან რომელია სწორი განმარტება?:

a) კონსტრუქტორი კონკრეტული ეგზემპლარისთვის გამოიძახება მხოლოდ ერთჯერ - მისი შექმნისას;
b) კონსტრუქტორი კონკრეტული ეგზემპლარისთვის გამოიძახება მრავალჯერ;
c) კონსტრუქტორი კონკრეტული ეგზემპლარისთვის არ გამოიძახება ერთჯერადად;
d) არცერთი არ არის სწორი

შეკითხვა 77

რამდენი კონსტრუქტორი შეიძლება იყოს განსაზღვრულ კლასში?:

a) მხოლოდ ერთი
b) ერთი ან ორი
c) რამდენიმე - საჭიროების მიხედვით
d) არცერთი

შეკითხვა 78

კლასში გვაქვს რამდენიმე კონსტრუქტორი, რა შეიძლება ვუწოდოთ მას?:

a) გადატვირთული კონსტრუქტორი
b) გადარქმეული კონსტრუქტორი
c) ერთსახელიანი კონსტრუქტორი
d) მრავალფუნქციური კონსტრუქტორი

შეკითხვა 79

რით განსხვავდებიან კლასის კონსტრუქტორები ერთმანეთისგან?:

a) სახელებით
b) დასაბრუნებელი ტიპით
c) დასაბრუნებელი მნიშვნელობით
d) პარამეტრებით

შეკითხვა 80

კლასში გვაქვს სამი კონსტრუქტორი: ერთი მათგანი არ მიიღებს პარამეტრებს, მეორე მიიღებს - ორ პარამეტრს, ხოლო მესამე - სამ პარამეტრს, რა ჰქვია ასეთ კონსტრუქტორებს?:

a) გადატვირთული კონსტრუქტორები
b) ცარიელი კონსტრუქტორები
c) გადარქმეული კონსტრუქტორები

d) მულტიფუნქციური კონსტრუქტორები

შეკითხვა 81

გადატვირთული კონსტრუქტორის შემთხვევაში კომპილატორი როგორ განსაზღვრავს რომელ კონსტრუქტორს მიმართავს ობიექტი?:

a) სახელის მიხედვით
b) დასაბრუნებელი ტიპის მიხედვით
c) გადასაცემი პარამეტრების მიხედვით
d) დასაბრუნებელი მნიშვნელობის მიხედვით

შეკითხვა 82

ჩამოთვლილთაგან რომელია სწორი განმარტება?:

a) დესტრუქტორი არ გადაიტვირთება, რადგან მას არა აქვს პარამეტრები;
b) დესტრუქტორი არ გადაიტვირთება, რადგან მას არა აქვს დასაბრუნებელი ტიპი;
c) დესტრუქტორი შეიძლება გადაიტვირთოს
d) დესტრუქტორი გადაიტვირთება, მიუხედავად იმისა, რომ არა აქვს პარამეტრები

შეკითხვა 83

რომელი გადატვირთული კონსტრუქტორია გამოყენებული მოცემულ ფრაგმენტში?:

```
class _2Data
{
  double data1;
  double data2;
public:
  _2Data();
  _2Data(double initData1, double initData2);
  void showData();
```

```
};
 _2Data::_2Data(double initData1, double initData2)
{
  data1 = initData1;
  data2 = initData2;
  }
 _2Data::_2Data()
{
  data1 = 0;
  data2 = 0;
  }
  ..................
```

a) _2Data() და _2Data(double initData1, double initData2)

b) data1 = initData1 და data2 = initData2

c) void showData()

d) double data1 და double data2

შეკითხვა 84

მოცემული ფრაგმენტის მიხედვით ჩამოთვლილთაგან რომელია სწორი?:

```
class _2Data
{
  double data1;
  double data2;
public:
  _2Data();
  _2Data(double initData1, double initData2);
  void showData();
};

_2Data::_2Data(double initData1, double initData2)
{
  data1 = initData1;
  data2 = initData2;
}
```

```
_2Data::_2Data()
{
  data1 = 0;
  data2 = 0;

}
.....................................

int _tmain(int argc, _TCHAR* argv[])
{
  _2Data Obj1;

  _2Data Obj2(222, 333);

  }


.............................
```

a) Obj1 მიმართავს _2Data() კონსტრუქტორს;
b) Obj2 მიმართავს _2Data(double initData1, double initData2) კონსტრუქტორს;
c) Obj1 მიმართავს ორპარამეტრიან კონსტრუქტორს;
d) Obj2 მიმართავს უპარამეტრო კონსტრუქტორს;

შეკითხვა 85

რას გამოიტანს პროგრამა შესრულების შედეგად?:

```
class _2Data
{
  double data1;
  double data2;
public:
  _2Data();
  _2Data(double initData1, double initData2);
  void showData();
};
```

```
_2Data::_2Data(double initData1, double initData2)
{
  data1 = initData1;
  data2 = initData2;
}

_2Data::_2Data()
{
  data1 = 0;
  data2 = 0;
}

void _2Data::showData()
{
  cout << "data1 = " << data1<<" ";
  cout << "data2 = " << data2<<" " ;
}


int _tmain(int argc, _TCHAR* argv[])
  {_2Data Obj1;
  Obj1.showData();

  _2Data Obj2(222, 333);
  Obj2.showData();system("pause");
}
```

a) data1=0 data2=0

b) data1=222 data2=333

c) data1=0 data2=0 data1=222 data2=333

d) "data1 = " data1; "data2 = " data2

შეკითხვა 86

მოცემული ფრაგმენტის მიხედვით რომელ კონსტრუქტორს მიმართავს X one (10)

ობიექტი?:

```
Class X
{
int integer_part;
double double_part;
public:
X(int i){integer_part =i;}
X(double d){double_part=d;}
}
```

a) x(int i){integer_part =i;}
b) x(double d){double_part=d;}
c) ორივეს შეიძლება მიმართოს
d) არცერთს არ მიმართავს

შეკითხვა 87

მოცემული ფრაგმენტის მიხედვით რომელ კონსტრუქტორს მიმართავს X two (3.14);

ობიექტი?:

```
Class X
{
int integer_part;
double double_part;
public:
X(int i){integer_part =i;}
X(double d){double_part=d;}
}
```

a) x(int i){integer_part =i;}
b) x(double d){double_part=d;}
c) ორივეს შეიძლება მიმართოს
d) არცერთს არ მიმართავს

თემა: ოპერატორების გადატვირთვა

შეკითხვა 88

რაში მდგომარეობს ოპერატორების გადატვირთვა?:

a) ოპერატორის აზრის შეცვლაში განსაზღვრულ კლასთან მიმართებით
b) ოპერატორის აზრის შეცვლა მთელი პროგრამისთვის
c) ოპერატორის აზრის შეცვლა პროგრამის დასაწყისში
d) ოპერატორის აზრის შეცვლა პროგრამის დასრულებისას

შეკითხვა 89

რა შემთხვევაში არის საჭირო ოპერატორების გადატვირთვა?:

a) პროგრამის კომპილაციის გასამარტივებლად
b) პროგრამის წაკითხვადობის გასაუმჯობესებლად
c) პროგრამის გააზრების გასამარტივებლად
d) პროგრამის შესრულების გასაუმჯობესებლად

შეკითხვა 90

რომელი გასაღებური სიტყვა გამოიყენება ოპერატორების გადასატვირთად?:

a) operator
b) union
c) protected
d) completed

შეკითხვა 91

რომელი გასაღებური სიტყვა არ გამოიყენება ოპერატორების გადასატვირთად?:

a) operator

b) union
c) protected
d) completed

შეკითხვა 92

თუ პროგრამა გადატვირთავს ოპერატორს რომელიმე კლასისთვის, მაშინ ამ ოპერატორის აზრი იცვლება:

a) პროგრამის მთელი ნაწილისთვის
b) მხოლოდ განსაზღვრული კლასისთვის
c) ყველა კლასისთვის
d) არცერთი კლასისთვის

შეკითხვა 93

ჩამოთვლილთაგან რომელი ოპერატორი არ გადაიტვირთება?:

a) *(ელემენტზე მიმთითებელი)
a) :: (ხედვის არეს ნებართვა)
b) (მინუსი)
c) + (პლიუსი)

შეკითხვა 94

ჩამოთვლილთაგან რომელი ოპერატორი გადაიტვირთება?:

a) *(ელემენტზე მიმთითებელი)
b) :: (ხედვის არეს ნებართვა)
c) (მინუსი)
d) + (პლიუსი)

შეკითხვა 95

მოცემულ ფრაგმენტში რომელია გადატვირთული ოპერატორი?:

```
class string
{
public:
   string(char *);
   void operator +(char *);
   void operator -(char);
   void show_string(void);
private:
   char data[256] ;
};
```

a) string(char *);
b) void operator +(char *);
c) void operator -(char);
d) void show_string(void);

შეკითხვა 96

მოცემულია პროგრამის ფრაგმენტი:

```
class string
{
public:
   string(char *);
   void operator +(char *);
   void show_string(void);
private:
   char data[256] ;
};
....
string::string(char *str)
{
   strcpy(data, str);
}
void string::operator +(char *str)
```

```
{
   strcat(data, str);
}
.......
int _tmain(int argc, _TCHAR* argv[])
{
   string title( "vscavlobT programirebas");
.....................
```

ქვემოთჩამოთვლილთაგან რომელია გადატვირთული ოპერატორის გამოყენების სწორი ბრძანება?:

a)title.show_string();
b) title + " me vscavlob!";
c) title.show_string() ;
d)title+show_string();

შეკითხვა 97

მოცემულია პროგრამის ფრაგმენტი:

```
class string
{
public:
   string(char *);
   void operator -(char);
   void show_string(void);
private:
   char data[256] ;
};
string::string(char *str)
{
   strcpy(data, str);
}
void string::operator -(char letter)
{
   char temp[256] ;
   int i, j;
```

```
        for (i = 0, j = 0; data[i]; i++) if (data[i]!= letter)
temp[j++] = data[i];
        temp[j] = NULL;
        strcpy(data, temp);
}
.....................
      int _tmain(int argc, _TCHAR* argv[])
      {string lesson("operatorebis gadatvirTva");
      ……..
```

ქვემოთჩამოთვლილთაგან რომელია გადატვირთული ოპერატორის გამოყენების სწორი ბრძანება?:

a) title.show_string() ;
b) lesson.show_string();
c) lesson - 'a';
d) lesson-show_string();

შეკითხვა 98

რას გამოიტანს შემდეგი პროგრამა?:

```
class string1
{
public:
   string1(char *);
   void operator +(char *);
   void show_string(void);
private:
   char data[256] ;
};
string1::string1(char *str)
{
   strcpy(data, str);
}
void string1::operator +(char *str)
{
   strcat(data, str);
}
void string1::show_string(void)
```

```
{
   cout << data << endl;
}

int _tmain(int argc, _TCHAR* argv[])
 {    string1 title( "automated control");
   title + " systems";
   title.show_string();
}
```

a) automated

b) automated control

c) automated control systems

d) control systems

შეკითხვა 99

რას გამოიტანს შემდეგი პროგრამა?:

```
class string1
{
public:
   string1(char *);
   void operator -(char);
   void show_string(void);
private:
   char data[256] ;
};
string1::string1(char *str)
{
   strcpy(data, str);
}
void string1::operator-(char letter)
{
   char temp[256] ;
   int i, j;
   for (i = 0, j = 0; data[i]; i++) if (data[i]!= letter)
temp[j++] = data[i];
   temp[j] = NULL;
```

```
   strcpy(data, temp);
}
void string1::show_string(void)
{
   cout << data << endl;
}


int _tmain(int argc, _TCHAR* argv[])
 {    string lesson("management");
   lesson - 'a';
   lesson.show_string();
system("pause") ;

}
```

a) management
b) mngement
c) manag
d) ment

შეკითხვა 100

რას გამოიტანს შემდეგი პროგრამა?:

```
class atype {
int a[3];
public:
atype (int i, int j, int k) {
a[0] = i;
a[1] = j;
a[2] = k;
}
int operator[] (int i) { return a[i]; }
};

int _tmain(int argc, _TCHAR* argv[])
 {
```

```
  atype ob(1, 2, 3);
  cout << ob[1]; system("pause") ;

  return 0;
  }
```

a) 1
b) 2
c) 3
d) 1 2 3

შეკითხვა 101

რას გამოიტანს შემდეგი პროგრამა?:

```
  class atype {
  int a[3];
  public:
  atype (int i, int j, int k) {
  a[0] = i;
  a[1] = j;
  a[2] = k;
  }
  int operator[] (int i) { return a[i]; }
  };

  int _tmain(int argc, _TCHAR* argv[])
    {
  atype ob(1, 2, 3);
  cout << ob[1]<<" "<<ob[2]; system("pause") ;

  return 0;
  }
```

a) 1 2
b) 2 3
c) 3 2
d) 1 2 3

თემა: კლასის სტატიკური წევრები

შეკითხვა 102

რომელი გასაღებური სიტყვით აღიწერება კლასის სტატიკური წევრი?:

a) static
b) private
c) public
d) protected

შეკითხვა 103

რომელი შეიძლება იყოს კლასის ერთობლივად გამოსაყენებელი მონაცემი?

a) გლობალური მონაცემი
b) სტატიკური მონაცემი წევრი
c) კლასის წევრი
d) ფუნქცია წევრი

შეკითხვა 104

პროგრამის რომელ ნაწილში აღიწერება სტატიკური ცვლადი?:

a) კლასის შიგნით
b) კლასის გარეთ
c) მთავარ ფუნქციაში
d) მთავარი ფუნქციის გარეთ

შეკითხვა 105

რომელი განსაზღვრაა მართებული სტატიკური ფუნქცია-წევრისთვის?

a) მას შეუძლია წვდომა მხოლოდ თავისივე კლასის სტატიკურ ცვლადებზე
b) მისი გამოყენება შეიძლება ობიექტის გარეშე კლასის სახელთან ერთად
c) მას შეუძლია წვდომა პროგრამის ნებისმიერ ელემენტზე მისი გამოყენება შეუძლებელია ობიექტის გარეშე

შეკითხვა 106

თუ პროგრამას არა აქვს ობიექტი, მაგრამ საჭიროა ელემენტის გამოყენება, როგორ უნდა გამოცხადდეს ეს გამოსაყენებელი ელემენტი?

a) Private და static
b) Protected და static
c) Public და static
d) სამივე სწორია

შეკითხვა 107

თუ ობიექტი არ არის შექმნილი როგორი სახით შეიძლება იქნას გამოძახებული სტატიკური ფუნქცია?

a) კლასის_სახელი::ფუნქციის_სახელი();
b) ფუნქციის_სახელი();
c) ფუნქციის_სახელი()::კლასის_სახელი
d) კლასის_სახელი:მონაცემის_სახელი

შეკითხვა 108

თუ კლასის ელემენტი გამოცხადებულია static სახით, მაშინ ასეთი ელემენტი შეიძლება გამოყენებულ იქნას:

a) მოცემული კლასის ყველა ობიექტის მიერ
b) მოცემული კლასის ზოგიერთი ობიექტის მიერ
c) მოცემული კლასის არცერთი ობიექტის მიერ
d) მოცემული კლასის შერჩევითი ობიექტის მიერ

შეკითხვა 109

კლასის სტატიკური ფუნქცია-ელემენტი შეიძლება გამოძახებულ იქნას მაშინაც კი, როდესაც:

a) არ არის შექმნილი მოცემული კლასის არცერთი ობიექტი
b) მიმართვა სტატიკურ ფუნქცია-ელემენტზე შეუძლებელია
c) მიმართვა სტატიკურ ფუნქცია-ელემენტზე საჯაროა
d) მიმართვა სტატიკურ ფუნქცია-ელემენტზე კერძოა

შეკითხვა 110

ჩამოთვლილთაგან რომელია სწორი განმარტება სტატიკურ ცვლადთან მიმართებით?

a) სტატიკური ცვლადი ინარჩუნებს თავის მნიშვნელობას იმ ბლოკიდან გამოსვლის შემდეგაც, სადაც იგი არის განსაზღვრული
b) სტატიკური ცვლადი ინიციალიზდება მხოლოდ ერთხელ და და ინახება პროგრამის დასრულებამდე
c) სტატიკური ცვლადი არ ინარჩუნებს თავის მნიშვნელობას ბლოკიდან გამოსვლის შემდეგ
d) სტატიკური ცვლადი ინიციალიზდება ყოველ ჯერზე მასზე მიმართვის დროს

შეკითხვა 111

რა დაიბეჭდება მოცემული ფრაგმენტის შესრულების შემდეგ?:

```
void incrementAndPrint()
{
    static int s_value = 1;
    ++s_value;
 std::cout << s_value << ",";}

int _tmain(int argc, _TCHAR* argv[])
{ incrementAndPrint();
    incrementAndPrint();
    incrementAndPrint();
   system("pause");
}
```

შეკითხვა 112

რა დაიბეჭდება მოცემული ფრაგმენტის შესრულების შემდეგ?

```
class CBase
{
```

```
 public:
      void BaseTest()
 {
  static int i=0;
  i++;
  cout<<i<<",";
 }
};

int _tmain(int argc, _TCHAR* argv[])
{CBase b1;
CBase b2;

b1.BaseTest();
b2.BaseTest();
system("pause");
 return 0;
}
```

შეკითხვა 113

რა დაიბეჭდება მოცემული ფრაგმენტის შესრულების შემდეგ?

```
.....................

void demo()
{
    static int count = 0;
    cout << count << ",";
       count++;
}

int _tmain(int argc, _TCHAR* argv[])
{  for (int i=0; i<5; i++)
        demo();
   system("pause");
 return 0;
}
```

შეკითხვა 114

რა დაიბეჭდება მოცემული ფრაგმენტის შესრულების შემდეგ?

```
class IndiaBix
{
    static int x;
    public:
    static void SetData(int xx)
    {
        x = xx;
    }
    static void Display()
    {
        cout<< x ;
    }
};
int IndiaBix::x = 0;
int _tmain(int argc, _TCHAR* argv[])
{    IndiaBix::SetData(44);
    IndiaBix::Display();
        return 0;
}
```

შეკითხვა 115

რა არის მოცემულ პროგრამაში შეცდომის მიზეზი?

```
class IndiaBix
{
    static int x;
    public:
    static void SetData(int xx)
    {
        x = xx;
    }
   void Display()
    {
        cout<< x ;
    }
```

```
};
int IndiaBix::x = 0;
int _tmain(int argc, _TCHAR* argv[])
{    IndiaBix::SetData(33);
    IndiaBix::Display();
       return 0;
}
```

a) `Display()` ფუნქცია არ არის სტატიკური

b) `Display()` ფუნქცის გამოძახება არასწორია

c) `SetData(33)` ფუნქცია ვერ მოახდენს მნიშვნელობის გადაცემას

d) `SetData(33)` ფუნქცია პარამეტრიანია

შეკითხვა 116

რა არის მოცემულ პროგრამაში შეცდომის მიზეზი?

```
class IndiaBix
{
    static int x;
    public:
    static void SetData(int xx)
    {
        this->x = xx;
    }
    static void Display()
    {
        cout<< x ;
    }
};
int IndiaBix::x = 0;
int _tmain(int argc, _TCHAR* argv[])
{        IndiaBix::SetData(22);
    IndiaBix::Display();
   system("pause");
       return 0;
}
```

a) `SetData(22)` ფუნქცია ვერ გადასცემს მნიშვნელობას
b) `Display()` ფუნქცია არ შეიძლება იყოს უპარამეტრო
c) `This` შეიძლება გამოყენებულ იქნას მხოლოდ არასტატიკური ფუნქცია-წევრისთვის
d) `IndiaBix::Display()` გამოძახება არასწორია

შეკითხვა 117

რა დაიბეჭდება მოცემული პროგრამის შესრულების შემდეგ?:

```
class IndiaBix
{
    static int count;
    public:
    static void First(void)
    {
        count = 10;
    }
    static void Second(int x)
    {
        count = count + x;
    }
    static void Display(void)
    {
        cout<< count << endl;
    }
};
int IndiaBix::count = 0;

int _tmain(int argc, _TCHAR* argv[])
{           IndiaBix :: First();
    IndiaBix :: Second(5);
    IndiaBix :: Display();
       return 0;
}
```

a) 15
b) 0

c) 5

d) 10

შეკითხვა 118

რა დაიბეჭდება მოცემული პროგრამის შესრულების შემდეგ?:

```
class IndiaBix
{
    static int x;
    public:
    IndiaBix()
    {
        if(x == 1)
            exit(0);
        else
            x++;
    }
    void Display()
    {
        cout<< x << ",";
    }
};
int IndiaBix::x = 0;

int _tmain(int argc, _TCHAR* argv[])
{   IndiaBix objBix1;
    objBix1.Display();
    IndiaBix objBix2;
    objBix2.Display();
       }
```

a) 1

b) 1,2

c) 0,1

d) 1,1

შეკითხვა 119

რა დაიბეჭდება მოცემული ფრაგმენტის შესრულების შემდეგ?:

```
class X {
private:
  int i;
public:
  static int si;
  void set_i(int arg) { i = arg; }
  static void set_si(int arg) { cout<<arg<<","; }
  void print_i() {
    cout << i <<",";
     }
};
int X::si = 77;

int _tmain(int argc, _TCHAR* argv[])
{    X xobj;
  xobj.set_i(11);
  xobj.print_i();
 X::set_si(13);
 cout<< X::si;
       return 0;
}
```

a) 11,13,77
b) 77,11,13
c) 11,13
d) 11,77

შეკითხვა 120

რა დაიბეჭდება მოცემული ფრაგმენტის შესრულების შემდეგ?:

```
class GfG
{
```

```
   public:
     static int i;
          };
int GfG::i = 1;
int _tmain(int argc, _TCHAR* argv[])
{ GfG obj1;
  GfG obj2;
 obj1.i =2;
 obj2.i = 3;
    cout << obj1.i<<","<<obj2.i;
  system("pause");
 return 0;
}
```

a) 3, 3
b) 1, 2, 3
c) 2, 3
d) 2, 2

შეკითხვა 121

რა დაიბეჭდება მოცემული ფრაგმენტის შესრულების შემდეგ?:

```
class GfG
{
   public:
     static int i;

static void f() {i++;cout<<i<<",";}
          };
int GfG::i = 1;
int _tmain(int argc, _TCHAR* argv[])
{    GfG obj;
     cout << obj.i<<",";
 GfG::f();
```

```
    cout<<obj.i;
    system("pause");
   return 0;
  }
```

a) 1,2,2
b) 1,1,1
c) 1,2,3
d) 2,2,2

თემა: მარტივი მემკვიდროებითობა

შეკითხვა 122

თუ პროგრამა იყენებს მემკვიდრეობითობას, მაშინ ახალი კლასის წარმოქმნისთვის საჭიროა?

a) საბაზო კლასი
b) მემკვიდრე კლასი
c) არ არის საჭირო საბაზო კლასი
d) არ არის საჭირო მემკვიდრე კლასი

შეკითხვა 123

მემკვიდროებითობა არის ობიექტზე ორინტირებული დაპროგრამების?

a) უმნიშვნელო კონცეფცია
b) ფუნდამენტური კონცეფცია
c) ყველა ზემოთ ჩამოთვლილი პასუხი სწორია
d) არცერთი პასუხი არ არის სწორი

შეკითხვა 124

შესაძლებელია თუ არა საბაზო კლასის კერძო ელემენტებზე პირდაპირი წვდომა მემკვიდრე კლასიდან?

a) შესაძლებელია
b) არ არის შესაძლებელი

c) ნაწილობრივ შესაძლებელია
d) პერიოდულად შესაძლებელია

შეკითხვა 125

წვდომის რომელი მოდიფიკატორი გამოიყენება მემკვიდრე კლასიდან წვდომის თვალსაზრისით?

a) public
b) private
c) protected
d) static

შეკითხვა 126

ჩამოთვლილთაგან რომელი დაცული წვდომის მოდიფიკატორი?

a) protected
b) public
c) private
d) static

შეკითხვა 127

როგორი სახე შეიძლება ჰქონდეს ზოგადად წარმოებული კლასის კონსტრუქტორს?

a) საბაზო კონსტრუქტორი1 (არგუმენტები სია) : საბაზო კონსტრუქტორი2 (არგუმენტების სია)
b) წარმოებული კონსტრუქტორი (არგუმენტები სია), საბაზო კონსტრუქტორი (არგუმენტების სია)
c) წარმოებული კონსტრუქტორი (არგუმენტები სია) : საბაზო კონსტრუქტორი (არგუმენტების სია)
d) საბაზო კონსტრუქტორი1 (არგუმენტები სია) , საბაზო კონსტრუქტორი2 (არგუმენტების სია)

შეკითხვა 128

როდესაც საბაზო კლასს აქვს კონსტრუქტორი არგუმენტებით, წარმოებულ კლასს რა სახით შეუძლია გამოიყენოს ეს კონსტრუქტორი?

a) საჭირო არგუმენტების საბაზო კლასზე გადაცემის გზით
b) არგუმენტებს თვითონვე მიანიჭებს მნიშვნელობებს
c) საჭირო არგუმენტების წარმოებულ კლასზე გადაცემის გზით
d) არგუმენტებს მიანიჭებს მუდმივ მნიშვნელობებს

შეკითხვა 129

ჩამოთვილთაგან რომელი გასაღებური სიტყვა გამოიყენება კლასის წევრებზე წვდომის თვალსაზრისით?

a) Default
b) Break
c) Protected
d) Asm

შეკითხვა 130

შესაძლებელია თუ არა, რომ საბაზო და მემკვიდრე კლასების ელემენტებს ჰქონდეთ ერთიდაიგივე სახელი?

a) შესაძლებელია
b) არ არის შესაძლებელი
c) ყველა პასუხი სწორია
d) არცერთი პასუხი არ არის სწორი

შეკითხვა 131

კლასთა მემკვიდრეობითობა არის?

a) არსებული საბაზო კლასიდან ახალი კლასის წარმოქმნის უნარი

b) კლასის თვისება
c) კლასის ობიექტის თვისება
d) ობიექტების სიმრავლის თვისება

შეკითხვა 132

კლასთა მემკვიდრეობითობა საშუალებას იძლევა შეიქმნას წარმოებული კლასი:

a) საბაზო კლასიდან
b) მეგობარი კლასიდან
c) წარმოებული კლასიდან
d) კომპლექსური კლასიდან

შეკითხვა 133

საბაზო კლასის რომელი ელემენტებია მისაწვდომი მემკვიდრე კლასში?

a) public
b) protected
c) private
d) contect

შეკითხვა 134

ჩამოთვლილთაგან რომელია სწორი განსაზღვრა?

a) საბაზო კლასზე მიმთითებელს არ შეუძლია მიეთითოს წარმოებულ კლასზე
b) წარმოებულ კლასზე მიმთითებელი არ შეიძლება შეიქმნას
c) წარმოებულ კლასზე მიმთითებელს არ შეუძლია მიეთითოს საბაზო კლასზე
d) საბაზო კლასზე მიმთითებელი არ შეიძლება შეიქმნას

შეკითხვა 135

აირჩიეთ სწორი განმარტება:

a) წარმოებული კლასის შექმნისას მემკვიდრეობითობის ტიპი შეიძლება იყოს მხოლოდ საჯარო
b) წარმოებული კლასის შექმნისას მემკვიდრეობითობის ტიპი შეილძება იყოს მხოლოდ კერძო
c) წარმოებული კლასის შექმნისას მემკვიდრეობითობის ტიპი შეიძლება იყოს როგორც საჯარო, ასევე კერძო და დაცული
d) წარმოებული კლასის შექმნისას მემკვიდრეობითობის ტიპი შეილძება იყოს მხოლოდ დაცული

შეკითხვა 136

რომელი განსაზღვრაა სწორი იმ შემთხვევისთვის, როდესაც კლასი მიღებულია დახურული (კერძო) მემკვიდრეობითობით?

a) საბაზო კლასის საჯარო წევრები ხდება წარმოებული კლასის დაცული წევრები
b) საბაზო კლასის საჯარო წევრები ხდება წარმოებული კლასის კერძო წევრები
c) საბაზო კლასის კერძო წევრები ხდება წარმოებული კლასის კერძო წევრები
d) საბაზო კლასის საჯარო წევრები ხდება წარმოებული კლასის საჯარო წევრები

შეკითხვა 137

რომელი განსაზღვრაა სწორი იმ შემთხვევისთვის, როდესაც კლასი მიღებულია დაცული მემკვიდრეობითობით?

a) საბაზო კლასის დაცული წევრები წარმოებულ კლასში ხდებიან კერძო წევრები
b) საბაზო კლასის საჯარო წევრები წარმოებულ კლასში ხდებიან საჯარო წევრები

c) საბაზო კლასის კერძო წევრები წარმოებულ კლასში ხდებიან კერძო წევრები
d) საბაზო კლასის საჯარო წევრები წარმოებულ კლასში ხდებიან დაცული წევრები

შეკითხვა 138

მოცემული კოდის მიხედვით რომელი კლასი არის საბაზო?

```
class manager : public employee
```

a) manager
b) employee
c) არცერთი კლასი არ არის საბაზო
d) ორივე კლასი საბაზოა

შეკითხვა 139

მოცემული კოდის მიხედვით რომელი კლასი არის წარმოებული?

```
class boss : public worker
```

a) boss
b) worker
c) არცერთი კლასი არ არის მემკვიდრე
d) ორივე კლასი მემკვიდრეა

შეკითხვა 140

რომელი განმარტებაა მართებული მოცემული ფრაგმენტის მიხედვით?

```
class manager : public employee
```

a) საბაზო კლასი არის manager, ხოლო მემკვიდრე კლასია employee
b) საბაზო კლასი არის employee, ხოლო მემკვიდრე კლასია manager
c) ორივე კლასი საბაზოა
d) ორივე კლასი მემკვიდრეა

შეკითხვა 141

რომელი განმარტებაა მართებული კოდის მოცემული ფრაგმენტის მიხედვით?

```
class manager : public employee
```

a) manager წარმოებული კლასი მიღებულია employee საბაზო კლასიდან ღია მემკვიდრეობითობის უპირატესობით
b) employee წარმოებული კლასი მიღებულია manager საბაზო კლასიდან ღია მემკვიდრეობითობის უპირატესობით
c) employee წარმოებული კლასი მიღებულია manager საბაზო კლასიდან დახურული მემკვიდრეობითობის უპირატესობით
d) employee წარმოებული კლასი მიღებულია manager საბაზო კლასიდან დაცული მემკვიდრეობითობის უპირატესობით

შეკითხვა 142

მოცემული კოდის მიხედვით რა სახელი შეიძლება ერქვას საბაზო კლასის კონსტრუქტორს?

```
class boss : public worker
```

a) boss
b) worker
c) Employee
d) Manager

შეკითხვა 143

მოცემული კოდის მიხედვით რა სახელი შეიძლება ერქვას წარმოებული კლასის კონსტრუქტორს?

```
class boss : public worker
```

a) boss
b) worker
c) Employee
d) Manager

შეკითხვა 144

მოცემული კოდის მიხედვით რა სახელი შეიძლება ერქვას საბაზო კლასის დესტრუქტროს?

```
class manager : public employee
```

a) ~manager
b) ~employee
c) ~Employee
d) ~Manager

შეკითხვა 145

მოცემული კოდის მიხედვით რა სახელი შეიძლება ერქვას მემკვიდრე კლასის დესტრუქტროს?

```
class manager : public employee
```

a) ~manager
b) ~employee
c) ~Employee
d) ~Manager

შეკითხვა 146

ჩამოთვლილთაგან რომელი განმარტება შეესაბამება მოცემულ ფრაგმენტს?

manager::manager(char *name, char *position, char *company_car, float salary, float bonus, int stock_options) : employee(name, position, salary)

a) manager კლასის კონსტრუქტორი იძახებს employee კლასის კონსტრუქტორს

b) employee კლასის კონსტრუქტორი იძახებს manager კლასის კონსტრუქტორს
c) manager კლასის კონსტრუქტორი არ იძახებს employee კლასის კონსტრუქტორს
d) manager კლასის კონსტრუქტორი იძახებს manager კლასის კონსტრუქტორს

შეკითხვა 147

მოცემული ფრაგმენტის მიხედვით რომელი მინიჭებებია სწორი წარმოებულ კლასში?

```
class some {
public:
  int a_;
protected:
  int b_;
private:
  int c_;
};
……..
class derived : public some {
  derived() {
    a_ = 0;
    b_ = 0;
    c_ = 0;
  }
};
```

a) a_ = 0
b) b_ = 0
c) c_ = 0
d) სამივე სწორია

შეკითხვა 148

მოცემული ფრაგმენტის მიხედვით რომელი მინიჭება მოგვცემს შეცდომას კომპილაციისას?

```
class some {
public:
  int a_;
protected:
  int b_;
private:
  int c_;
};
……..
class derived : public some {
  derived() {
    a_ = 0;
    b_ = 0;
    c_ = 0;
  }
};
```

a) a_ = 0
b) b_ = 0
c) c_ = 0
d) სამივე მინიჭება

შეკითხვა 149

რა შედეგს გამოიტანს მოცემული პროგრამა?

```
class BixBase
{
    int x, y;
    public:
    BixBase(int xx = 10, int yy = 10)
    {
        x = xx;
        y = yy;
```

```
    }
    void Show()
    {
        cout<< x * y << endl;
    }
};
class BixDerived : public BixBase
{
    private:
        BixBase objBase;
    public:
    BixDerived(int xx, int yy) : BixBase(xx, yy)
    {
        objBase.Show();
    }
};
int _tmain(int argc, _TCHAR* argv[])

{    BixDerived objDev(10, 20);}
```

a) 100
b) 200
c) 100,200
d) შეცდომას კომპილაციისას

შეკითხვა 150

რა შედეგს გამოიტანს მოცემული პროგრამა?

```
class A
{
    public:
    void BixFunction(void)
    {
        cout<< "Class A" << ",";
    }
};
class B: public A
{
```

```
    public:
    void BixFunction(void)
    {
        cout<< "Class B" << endl;
    }
};
int _tmain(int argc, _TCHAR* argv[])
{   A *ptr;
    B objB;
    ptr = &objB;
    ptr->BixFunction();
    objB. BixFunction();
 system("pause");

}
```

a) Class A
b) Class B
c) Class C
d) Class A, Class B

შეკითხვა 151

რა შედეგს გამოიტანს მოცემული პროგრამა?

```
class A
{
    public:
    void BixFunction(void)
    {
        cout<< "Class A" << " ";
    }
};
class B: public A
{
    public:
    void BixFunction(void)
    {
        cout<< "Class B" << endl;
```

```
    }
};
class C : public B
{
    public:
    void BixFunction(void)
    {
        cout<< "Class C" << endl;
    }
};
int _tmain(int argc, _TCHAR* argv[])
{  A *ptr;
    B objB;
    ptr = &objB;
    ptr->BixFunction();
    ptr = new C();
    ptr->BixFunction();
       }
```

a) Class A Class A
b) Class A
c) Class B
d) Class C

შეკითხვა 152

რა შედეგს გამოიტანს მოცემული პროგრამა?

```
class Based
{
    public:
    void OutFunction(void)
    {
        cout<< "Class_1" <<endl;
    }
};
class Derived1: public Based
{
    public:
```

```
        void OutFunction(void)
        {
            cout<< "Class_2"<< ",";
        }
    };
    class Derived2 : public Derived1
    {
        public:
        void OutFunction(void)
        {
            cout<< "Class_3" << ",";
        }
    };
    int _tmain(int argc, _TCHAR* argv[])
    {  Based B; Derived1 D1; Derived2 D2;
    D2.OutFunction();
    D1.OutFunction();
    B.OutFunction();
           system("pause");
     }
```

a) Class_3, Class_2, Class_1
b) Class_1, Class_2, Class_3
c) Class_3, Class_2,
d) Class_2, Class_1

შეკითხვა 153

რა დაიბეჭდება მოცემული ფრაგმენტის შესრულების შემდეგ?:

```
class Shape {
  public:
    void setWidth(int w) {
      width = w;
    }
    void setHeight(int h) {
      height = h;
    }
```

```
  protected:
     int width;
     int height;
};
class Rectangle: public Shape {
  public:
     int getArea() {
       return (width * height);
     }
};
  int _tmain(int argc, _TCHAR* argv[])
{
  Rectangle Rect;
  Rect.setWidth(15);
  Rect.setHeight(5);
         cout<< Rect. getArea();

         ……..}
```

შეკითხვა 154

რა დაიბეჭდება პროგრამის ამ ფრაგმენტის შესრულების შემდეგ?:

```
class  PRODUCT{
   public:
      void pr1(int x) {
         x1 = x;
      }
      void pr2(int y) {
         y1 = y;
      }
  protected:
      int x1;
      int y1;
};
class SS: public PRODUCT {
   public:
      int Pr() {
```

```
            return x1*y1;
        }
};
int _tmain(int argc, _TCHAR* argv[])
{
  SS p;
   p.pr1(5);
   p.pr2(5);
  cout<< p.Pr();
   ……….
}
```

თემა: Protected დაცული მონაცემები

შეკითხვა 155

რომელია დაცული წვდომის სპეციფიკატორი:

a) private
b) public
c) protected
d) სამივე

შეკითხვა 156

ჩამოთვლილთაგან რომელია სწორი გამონათქვამი?:

a) თუ ელემენტი არის დაცული, წარმოებული კლასის ობიექტები მას მიმართავენ როგორც საჯაროს
b) თუ ელემენტი არის დაცული, წარმოებული კლასის ობიექტები მას მიმართავენ ოპერატორ წერტილის გამოყენებით
c) თუ ელემენტი არის დაცული, წარმოებული კლასის ობიექტები მას მიმართავენ როგორც კერძოს
d) თუ ელემენტი არის დაცული, წარმოებული კლასის ობიექტები მასზე მიმართვას ვერ განახორციელებენ

შეკითხვა 157

რისთვის გამოიყენება protected ჰდე?:

a) რათა წარმოებული კლასის ობიექტებს არ ჰქონდეთ პირდაპირი წვდომა საბაზო კლასის ელემენტებზე;
b) რათა საბაზო კლასის ელემენტებმა წვდომა ვერ განახორციელონ წარმოებულ კლასზე
c) რათა წარმოებული კლასის ობიექტებს საბაზო კლასის ელემენტებზე პირდაპირი წვდომის ნება დაერთოს
d) რათა საბაზო კლასის ელემენტებმა წვდომა განახორციელონ წარმოებულ კლასზე

შეკითხვა 158

წარმოებულ კლასს შეუძლია მიმართოს საბაზო კლასის დაცულ ელემენტებს, როგორც:

a) კერძოს
b) საჯაროს
c) წარმოებულს
d) ნაწილობრივს

შეკითხვა 159

წარმოებული კლასის გარდა პროგრამის სხვა ნაწილისთვის საბაზო კლასის დაცული ელემენტები:

a) კერძოს ექვივალენტურია
b) საჯაროს ექვივალენტურია
c) კერძოს არაექვივალენტურია
d) სამივე პასუხი მცდარია

შეკითხვა 160

რაზე მიუთითებს protected წვდომის მოდიფიკატორი?:

a) პროგრამაში გამოყენებული იქნება კლასთა მემკვიდრეობითობა;

b) პროგრამა იქნება სინტაქსურად გამართული;
c) პროგრამა იქნება სემანტიკურად გამართული;
d) პროგრამის ტესტირება იქნება შესაძლებელი.

შეკითხვა 161

მოცემულია პროგრამის ფრაგმენტი. void update() ფუნქციაში რომელი მინიჭებებია სწორი?:

```
class a
{
  private: int x;
  public: int y;
  protected: int z;
};
class b:public a
{
   public:
     void update()
       {
        x=100;
        y=100;
        z=100;
       cout<<y<<" "<<z;       }
};
```

a) სამივე: x=100; y=100; z=100;
b) მხოლოდ x=100;
c) y=100; z=100;
d) მხოლოდ y=100;

შეკითხვა 162

მოცემულია პროგრამის ფრაგმენტი, რომელშიც სინტაქსური შეცდომაა. ქვემოთჩამოთვლილთაგან რომელია სინტაქსური შეცდომის მიზეზი?:

```
class a
```

```
{
  private: int x;
  public: int y;
  protected: int z;
};

class b:public a
{
    public:
       void update()
          {
            x=100;
            y=100;
            z=100;
           cout<<y<<" "<<z;          }
};
```

a) x=100, რადგან მას აქვს private ჭდე;
b) y=100, რადგან მას აქვს public ჭდე;
c) z=100, რადგან მას აქვს protected ჭდე;
d) სამივე შეცდომაა

შეკითხვა 163

მოცემულ ფრაგმენტში შეცდომაა. რა ცვლილება უნდა შევიტანოთ, რათა სინტაქსურად იყოს გამართული?:

```
class A
{private:
int i;
public:
void showi() {cout<<"i="<<i<<"\n";}
};

class B : private A
{
public:
void seti_B(int tmp) {i=tmp; showi();}
```

```
};
int _tmain(int argc, _TCHAR* argv[])
{B obB;
obB.seti_B(777);
system("pause");
  return 0;
}
```

a) i მონაცემს მივანიჭოთ public ჭდე
b) i მონაცემს მივანიჭოთ protected ჭდე
c) i მონაცემს შევცვალოთ სხვა ცვლადით
d) პროგრამის გასწორება შეუძლებელია

შეკითხვა 164

რა დაიბეჭდება შემდეგი ფრაგმენტის შესრულების შედეგად:

```
class one
{protected:
int x;
public:
int y;
};
class two : public one
{
public:
void set_obj(int tmp1, int tmp2) { x=tmp1; y=tmp2;
cout<<x+y;}
};
int _tmain(int argc, _TCHAR* argv[])
{two obj;
obj.set_obj(19,11);
   }
```

a) 19, 11
b) 19
c) 11
d) 30

შეკითხვა 165

რა დაიბეჭდება შემდეგი ფრაგმენტის შესრულების შედეგად:

```
class one
{
 int x;
protected:
int y;
};
class two : public one
{
public:
void set_obj(int tmp1, int tmp2) { x=tmp1; y=tmp2;
cout<<x*y;}
};

int _tmain(int argc, _TCHAR* argv[])
{two obj;
obj.set_obj(21,2);
```

a) 42
b) 21,2
c) 2, 21
d) პროგრამაში შეცდომაა

შეკითხვა 166

რა დაიბეჭდება შემდეგი ფრაგმენტის შესრულების შედეგად:

```
class Box1 {
 protected:
int width;
public:
      int length;

};
 class Box2: public Box1
  {public:
void get_box(int tmp1, int tmp2){width=tmp1;
length=tmp2;
```

```
cout<<"product of box:"<<tmp1*tmp2;}
 };
int _tmain(int argc, _TCHAR* argv[])
{Box2 boxa;
    boxa.get_box(6,7);
     return 0;
 }
```

a) product of box:42
b) length of box: 6
c) width of box:7
d) temp1*temp2

შეკითხვა 167

მოცემულ ფრაგმენტში შეცდომაა. აირჩიეთ რა არის შეცდომის სავარაუდო მიზეზი?:

```
class Box1 {
 private:
int width;
public:
      int length;

};
 class Box2: public Box1
  {public:
void  get_box(int  tmp1,  int  tmp2){width=tmp1;
length=tmp2;
cout<<"product of box:"<<tmp1*tmp2;}
 };
int _tmain(int argc, _TCHAR* argv[])
{Box2 boxa;
    boxa.get_box(6,7);
     return 0;
 }
```

a) წარმოებული კლასიდან `length` მონაცემზე პირდაპირი წვდომა ვერ შესრულდება;
b) `boxa` ობიექტი პირდაპირ წვდომას ვერ განახორციელებს `get_box` ფუნქციაზე;

c) `width` მონაცემი წარმოებული კლასის ხედვის არეში არ იმყოფება;
d) წარმოებული კლასიდან შეუძლებელია შედეგის გამოტანა.

თემა: მრავლობითი მემკვიდრეობითობა

შეკითხვა 168

რა შემთხვევაში მიიღება მრავლობითი მემკვიდრეობითობა?:
a) როდესაც კლასი შეიქმნება რამდენიმე საბაზო კლასისგან;
b) როდესაც კლასი შეიქმნება ერთი საბაზო კლასისგან;
c) როდესაც კლასი შეიქმნება ღია მემკვიდრეობით;
d) როდესაც კლასი შეიქმნება დაცული მემკვიდრეობითობით.

შეკითხვა 169

რომელ ატრიბუტებს მიიღებს მრავლობითი მემკვიდრეობითობის დროს წარმოებული კლასი?:
a) მხოლოდ ერთი კლასის ატრიბუტებს;
b) ორი ან მეტი კლასის ატრიბუტებს;
c) მხოლოდ ერთი კლასის დაცულ ატრიბუტებს;
d) მხოლოდ ერთი კლასის დახურულ ატრიბუტებს.

შეკითხვა 170

წარმოებული კლასის კონსტრუქტორი მრავლობითი მემკვიდრეობითობის დროს რას გამოიძახებს?:
a) ყველა საბაზო კლასის კონსტრუქტორს;
b) მხოლოდ ერთი საბაზო კლასის კონსტრუქტორს;
c) მხოლოდ ერთი კლასის კონსტრუქტორს;
d) საბაზო კლასის კონსტრუქტორს არ გამოიძახებს.

შეკითხვა 171

მრავლობითი მემკვიდრეობითობის დროს როგორი მიმდევრობით ხდება წარმოებულ კლასში საბაზო კლასების კონსტრუქტორების გამოძახება?:
a) ნებისმიერი მიმდევრობით;
b) მიმდევრობას არავითარი მნიშვნელობა არა აქვს;

c) იმავე მიმდევრობით, როგორც ნაჩვენებია სიაში წარმოებული კლასის გამოცხადებისას;
d) სამივე პასუხი მცდარია.

შეკითხვა 172

როგორ შეიძლება გამოცხადდეს წარმოებული კლასი რამდენიმე საბაზო კლასიდან?:

a) წარმოებული კლასის სახელისა და ორი წერტილის შემდეგ უნდა მიეთითოს მძიმეებით გამოყოფილი საბაზო კლასის სახელები;
b) წარმოებული კლასის სახელისა და ორი წერტილის შემდეგ უნდა მიეთითოს რომელიმე საბაზო კლასის სახელი;
c) წარმოებული კლასის სახელის შემდეგ არ არის აუცილებელი საბაზო კლასის სახელების მითითება;
d) სამივე პასუხი მცდარია.

შეკითხვა 173

მრავლობითი მემკვიდრეობითობის დროს, როდესაც გამოიყენება საბაზო კლასების სია, როგორი მიმდევრობით გამოიძახება მათი დესტრუქტორები?:

a) სიის მიხედვით, მარცხნიდან მარჯვნივ;
b) უკუმიმდევრობით, მარჯვნიდან მარცხნივ;
c) ნებისმიერი მიმდევრობით;
d) საერთოდ არ გამოიძახება.

შეკითხვა 174

რა შეიძლება ითქვას პროცესზე, როდესაც წარმოებული კლასიდან ვქმნით ახალ კლასს?

a) მიიღება კლასთა კომპლექტი;
b) მიიღება კლასთა ჯგუფი;
c) მიიღება კლასთა დაჯგუფება;
d) მიიღება კლასთა იერარქია;

შეკითხვა 175

რა შეიძლება ითქვას შემთხვევაზე, როდესაც კლასი შეიქმნება რამდენიმე საბაზო კლასისგან?

a) მიიღება მრავლობითი მემკვიდრეობითობა;
b) მიიღება მარტივი მემკვიდრეობითობა;
c) მიიღება მეგობარი კლასი;
d) მიიღება ანალოგიური კლასი.

შეკითხვა 176

ჩამოთვლილთაგან რომელია მისაწვდომი კლასთა იერარქიაში?

a) Private მონაცემ-წევრები;
b) Protected მონაცემ-წევრები;
c) ფუნქცია-წევრები;
d) ყველა ჩამოთვლილი.

შეკითხვა 177

ჩამოთვლილთაგან რომელი არ არის მემკვიდრეობითობის ტიპი?

a) მრავლობითი;
b) მრავალდონიანი;
c) განაწილებული;
d) იერარქიული

შეკითხვა 178

c++ საშუალებას იძლევა მემკვიდრეობა მიღებულ იქნას:

a) მხოლოდ ერთი კლასიდან;
b) რამდენიმე კლასიდან;
c) ორი ან მეტი კლასიდან;
d) არცერთი კლასიდან.

შეკითხვა 180

თუ რამდენიმე საბაზო კლასში განსაზღვრულია წევრები ერთნაირი სახელებით, როგორ უნდა მივმართოთ ასეთ წევრებს?

a) წევრის წინ მივუთითოთ კლასის სახელი და ნებართვის ოპერატორი ::
b) წევრის წინ მივუთითოთ კლასის სახელი;
c) მივმართოთ მხოლოდ წევრის სახელით;
d) წევრზე მიმართვა შეუძლებელია.

შეკითხვა 181

რომელი ტიპის მემკვიდრეობითობაა მოცემული *class A : public X, public Y* მაგალითში?

a) მრავალდონიანი
b) მრავლობითი
c) ჰიბრიდული
d) იერარქიული

შეკითხვა 182

მოცემული ფრაგმენტიდან გამომდინარე რომელი გამონათქვამია სწორი class C -სთან მიმართებით?

```
class A
{public:
    A() {}
.....
};
class B
{public
    B() {}
.....
};

class C: public A, public B
{
    C(){}
```

```
.............
    };
```

a) საჯარო მემკვიდრეობითობით მიღებული წარმოებული კლასია;
b) მიღებულია მრავლობითი მემკვიდრეობითობით;
c) მიღებულია ერთი კლასის ბაზაზე;
d) არ არის წარმოებული კლასი.

შეკითხვა 183

მოცემულია ორი კლასი: computer_screen და mother_board, მათგან მრავლობითი მემკვიდრეობითობით რომელი ბრძანებით შეიძლება მივიღოთ computer კლასი?

```
class computer_screen
{ public:
.............
};

class mother_board
{ public:
.............};
```

a) class computer : public computer_screen, public mother_board;
b) class computer : computer_screen, mother_board;
c) class computer : public computer_screen;
d) class computer : public mother_board.

შეკითხვა 184

მოცემულია ორი კლასი: cat და rabbit, მათგან მრავლობითი მემკვიდრეობითობით რომელი ბრძანებით შეიძლება მივიღოთ cabbit კლასი?

```
class   cat
{ public:
.............
```

```
};

class  rabbit
{ public:
..............};
```

a) class cabbit : public cat, public rabbit;
b) class cabbit : cat, rabbit;
c) class cabbit : public cat;
d) class cabbit : public rabbit.

შეკითხვა 185

მოცემული ფრაგმენტის მიხედვით ჩამოთვლილთაგან რომელი განსაზღვრა შეესაბამება `class` A-ს?

```
class X
{...};
class Y
{...};
class Z
{...};
class A : public X, public Y, public Z
{...};
```

a) წარმოებული კლასი მიღებულია სამი საბაზო კლასიდან ღია მემკვიდრეობით;
b) წარმოებული კლასი ჩამოთვლილთაგან ერთ-ერთის მემკვიდრეა;
c) წარმოებული კლასი X კლასის საჯარო მემკვიდრეა;
d) წარმოებული კლასი Y კლასის საჯარო მემკვიდრეა;

შეკითხვა 186

მოცემული ფრაგმენტის მიხედვით A კლასში გვინდა გამოვითვალოთ სამივე კლასის key ელემენტების ჯამი. ჩამოთვლილთაგან რომელია სწორი ბრძანება?

```
class X
```

```
{int key;};
class Y
{int key;};
class Z
{int key;};
class A : public X, public Y, public Z
{int key;
.....................
};
```

a) key=X::key+Y::key+Z::key;

b) A::key=key+key+key;

c) key=X+Y+Z;

d) key=X.key+Y.key+Z.key;

შეკითხვა 187

ფრაგმენტში მოცემულია ორი საბაზო კლასი: base1, base2 რომელთაგან გვინდა შევქმნათ my_class წარმოებული კლასი. ჩამოთვლილთაგან აირჩიეთ რომელია სწორი ბრძანება?

```
class base1
     {
     public:
     void run(){}
     };

     class base2
     {
     public:
     void run() {}
        };
```

a) class my_class : public base1, public base2 { }

b) class my_class : private base1, private base2 { }

c) class my_class : protected base1, protectes base2 { }

d) არცერთი არ არის სწორი

შეკითხვა 188

მოცემული ფრაგმენტიდან გამომდინარე რომელი გამონათქვამია მართებული?

```
class Bird {};
class Swimmer {};
class Penguin : public Bird, public Swimmer {};
```

a) Penguin კლასი შექმნილია Bird და Swimmer კლასებიდან ღია (საჯარო) მემკვიდრეობითობის პრივილეგიით;
b) Penguin კლასი შექმნილია Bird და Swimmer კლასებიდან კერძო მემკვიდრეობითობის პრივილეგიით;
c) Penguin კლასი შექმნილია Bird და Swimmer კლასებიდან დაცული მემკვიდრეობითობის პრივილეგიით;
d) Penguin კლასი შექმნილია Bird და Swimmer კლასებიდან შერეული მემკვიდრეობითობის პრივილეგიით

შეკითხვა 189

მოცემული ფრაგმენტიდან გამომდინარე რომელი გამონათქვამია მართებული?

```
class Bird {};
class Swimmer {};
class Penguin : private Bird, private Swimmer {};
```

a) Penguin კლასი შექმნილია Bird და Swimmer კლასებიდან ღია(საჯარო) მემკვიდრეობითობის პრივილეგიით;
b) Penguin კლასი შექმნილია Bird და Swimmer კლასებიდან კერძო მემკვიდრეობითობის პრივილეგიით;
c) Penguin კლასი შექმნილია Bird და Swimmer კლასებიდან დაცული მემკვიდრეობითობის პრივილეგიით;
d) Penguin კლასი შექმნილია Bird და Swimmer კლასებიდან შერეული მემკვიდრეობითობის პრივილეგიით

შეკითხვა 190

მოცემული ფრაგმენტიდან გამომდინარე რომელი გამონათქვამია მართებული?

```
class Bird {};
class Swimmer {};
class Penguin : protected Bird, protected Swimmer {};
```

a) Penguin კლასი შექმნილია Bird და Swimmer კლასებიდან ღია(საჯარო) მემკვიდრეობითობის პრივილეგიით;
b) Penguin კლასი შექმნილია Bird და Swimmer კლასებიდან კერძო მემკვიდრეობითობის პრივილეგიით;
c) Penguin კლასი შექმნილია Bird და Swimmer კლასებიდან დაცული მემკვიდრეობითობის პრივილეგიით;
d) Penguin კლასი შექმნილია Bird და Swimmer კლასებიდან შერეული მემკვიდრეობითობის პრივილეგიით

შეკითხვა 191

მოცემულია ორი საბაზო კლასი Bird და Swimmer, ჩამოთვლილთაგან რომელი გამოაცხადებს მათგან მიღებულ Penguin კლასს დაცული მემკვიდრეობითობის პრივილეგიით?

a) class Penguin : protected Bird, protected Swimmer {};
b) class Penguin : private Bird, private Swimmer {};
c) class Penguin : public Bird, public Swimmer {};
d) class Penguin : private Bird, public Swimmer {};

შეკითხვა 192

ფრაგმენტში მოცემულია საბაზო კლასი, აირჩიეთ რომელია კონსტრუქტორი დუმილით?

```
class Unit
{
protected:
        int health;
public:
```

```
            void showHealth() { cout << "Unit health:
" << health << endl; };
            Unit(): health(10) { }
            Unit(int a): health(a) { }
    };
```

a) Unit(int a): health(a) { }
b) Unit(): health(10) { }
c) health(10)
d) health(a)

შეკითხვა 193

ფრაგმენტში მოცემულია საბაზო კლასი, აირჩიეთ რომელია კონსტრუქტორი პარამეტრით?

```
    class Unit
    {
    protected:
            int health;
    public:
            void showHealth() { cout << "Unit health:
" << health << endl; };
            Unit(): health(10) { }
            Unit(int a): health(a) { }
    };
```

a) Unit(): health(10) { }
b) Unit(int a): health(a) { }
c) health(10)
d) health(a)

შეკითხვა 194

მოცემული ფრაგმენტის მიხედვით რომელი შეიძლება იყოს computer კლასის კონსტრუქტორი?

```
    class computer_screen
    {
    public:
```

```
        computer_screen(char *, int, int);
    private:
        ..............
    };
    class mother_board
    {
    public:
        mother_board( int, int);
    private:
        ..............
    };
    class computer : public computer_screen, public
mother_board
    {
       private:
        char name[64];
        int hard_disk;
        ......................
    };
```

a) computer(char *, int);

b) computer(char *, int, char *, int, int, int, int);

c) computer(char *, int, char *, int);

d) computer(int, int, int, int)

შეკითხვა 195

რას გამოიტანს შემდეგი პროგრამა?

```
class BaseClass1 {
public:
   BaseClass1() {
      cout << "BaseClass1 constructor." << endl;
   }
};
class BaseClass2 {
public:
   BaseClass2() {
```

```
        cout << "BaseClass2 constructor." << endl;
    }
};
class DerivedClass : public BaseClass1, public BaseClass2  {
public:
    DerivedClass() {
        cout << "DerivedClass constructor." << endl;
    }
};

int _tmain(int argc, _TCHAR* argv[])
{ DerivedClass dc;
}
```

a) BaseClass1 constructor
b) BaseClass2 constructor
c) DerivedClass constructor
d) არცერთს არ გამოიტანს

შეკითხვა 196

რას გამოიტანს მოცემული პროგრამა?

```
class base1
{ public:
int x;
};
class base2
{public:
int y;
};
class derived: public base1, public base2
{ private:
int z, s;
public:
void show_der ()
{z=2; int p=x*y*z;
cout<<p<<endl;}
```

```
};

int _tmain(int argc, _TCHAR* argv[])
{   derived d;
d.x=6;
d.y=8;
d.show_der();
}
```

a) 96
b) 48
c) 12
d) 16

შეკითხვა 197

რას გამოიტანს შემდეგი პროგრამა?

```
class base1
{ private:
int x;
};
class base2
{private:
int y;
};
class derived: public base1, public base2
{ public:
int z,s,p;
private:
void show_der ()
{p=x*y*z;
cout<<p<<endl;}
};
int _tmain(int argc, _TCHAR* argv[])
{   derived d;
d.x=6;
d.y=8;
d.show_der();
```

```
}
```

a) 96
b) 18
c) 12
d) შეცდომას კომპილაციისას

შეკითხვა 198

რას გამოიტანს მოცემული პროგრამა?

```
class base1
{ protected:
int a;
};
class base2
{protected:
int b;
};

class derived: public base1, public base2
{ private:
int p;
public:
void show_der ()
{a=3; b=4;
p=2*(a+b);
cout<<p++<<endl;}
};

int _tmain(int argc, _TCHAR* argv[])
{   derived m;
m.show_der();
}
```

შეკითხვა 199

რას გამოიტანს შემდეგი პროგრამა?

```
class D_class1
{
```

```
public:
D_class1() {cout << "D_class1 created\n";}
};
class D_class2
{
public:
D_class2() {cout << "D_class2 created\n";}
};
class D_class: public D_class1, public D_class2
 {
public:
D_class() {cout << "D_class created\n";}
};

int _tmain(int argc, _TCHAR* argv[])
{  D_class d;
}
```

a) D_class1 created
b) D_class2 created
c) D_class created
d) არცერთს არ გამოიტანს

შეკითხვა 200

როგორი მიმდევრობით გამოიძახებს პროგრამა კლასის კონსტრუქტორებს?

```
class D_class1
{
public:
D_class1() {cout << "D_class1 created\n";}
~D_class1() {cout << "  D_class1 destroyed\n "; }
};
class D_class2
{
public:
D_class2() {cout << "D_class2 created\n";}
```

```
~D_class2() {cout << "  D_class2 destroyed\n "; }
};
class D_class: public D_class1, public D_class2
 {
public:
D_class() {cout << "D_class created\n";}
~D_class() {cout << "  D_class destroyed\n "; }
};

int _tmain(int argc, _TCHAR* argv[])
{  D_class d;
return 0;
}
```

a) D_class1() D_class2() D_class()
b) D_class2() D_class1() D_class()
c) D_class() D_class1() D_class2()
d) სამივე მიმდევრობით

თემა: ვირტუალური ფუნქცია, პოლიმორფიზმი

შეკითხვა 201

პროგრამის რომელ ნაწილში განისაზღვრება ვირტუალური ფუნქცია?

a) წარმოებულ კლასში
b) მეგობარ კლასში
c) საბაზო კლასში
d) დახურულ კლასში

შეკითხვა 202

ნებისმიერ წარმოებულ კლასს შეუძლია:

a) შექმნას ვირტუალური ფუნქცია
b) დაასახელოს ვირტუალური ფუნქცია
c) გამოიყენოს ან გადატვირთოს ვირტუალური ფუნქცია

d) გაანადგუროს ვირტუალური ფუნქცია

შეკითხვა 203

როგორ შეიძლება ვირტუალური ფუნქციის გამოძახება?

a) საბაზო კლასის მიმთითებლის მეშვეობით
b) მონაცემ-წევრის მეშვეობით
c) საჯარო მონაცემების მეშვეობით
d) დახურული მონაცემების მეშვეობით

შეკითხვა 204

როგორ განისაზღვრება ვირტუალური ფუნქციის რომელი ეგზემპლარი უნდა იქნას გამოძახებული?

a) არჩევანი დინამიურად დამოკიდებულია ობიექტზე, რომელზეც იგზავნება მიმთითებელი
b) არჩევანი განისაზღვრება ვირტუალური ფუნქციის მიერ
c) არჩევანი განისაზღვრება ვირტუალური ფუნქციის მიერ პროგრამის შესრულების ბოლო ეტაპზე
d) არჩევანი განისაზღვრება ვირტუალური ფუნქციის მიერ პროგრამის დასაწყისში

შეკითხვა 205

რომელი ფრაზით ხასიათდება პოლიმორფიზმი c++ ენაში?

a) მრავალი ინტერფეისი -ერთი მეთოდი
b) მრავალი ინტერფეისი - მრავალი მეთოდი
c) ერთი ინტერფეისი - მრავალი მეთოდი
d) ფაქტობრივი ინტერფეისი - ფაქტობრივი მეთოდი

შეკითხვა 206

პოლიმორფიზმი არის თვისება, რომელიც საშუალებას იძლევა:

a) ერთიდაიგივე სახელი გამოყენებულ იქნას მხოლოდ ერთი მსგავსი ამოცანის ამოხსნისთვის
b) სხვადასხვა სახელი გამოყენებულ იქნას მრავალი მსგავსი ამოცანის ამოხსნისთვის
c) ერთიდაიგივე სახელი გამოყენებულ იქნას რამდენიმე მსგავსი, მაგრამ ტექნიკურად განსხვავებული ამოცანისთვის;
d) სხვადასხვა სახელი გამოყენებულ იქნას მხოლოდ ერთი ამოცანის ამოხსნისთვის

შეკითხვა 207

რომელ ფუნქციას იყენებს პროგრამა პოლიმორფული ობიექტის შექმნისთვის?

a) Virtual ფუნქციას
b) Static ფუნქციები
c) მეგობარ ფუნქციას
d) წარმოებულ ფუნქციას

შეკითხვა 208

რომელი გასაღებური სიტყვა მიუთითებს, რომ ფუნქცია ვირტუალურია?

a) Virtual
b) Static
c) Void
d) Const

შეკითხვა 209

როგორ განვასხვავოთ წმინდად ვირტუალური ფუნქცია?

a) შეიცავს მრავალ ოპერატორს

b) არ შეიცავს ოპერატორებს და მინიჭებული აქვს 0 მნიშვნელობა
c) შეიცავს ერტადერტ ოპერატორს
d) შეიცავს ერტზე მეტ ოპერატორს

შეკითხვა 210

რა შედეგს გამოიტანს მოცემული პროგრამა?

```
class X {
protected:
  int i;
public:
  void seti(int c) {i = c;}
  virtual void print() {cout << i<<",";}
};
class Y : public X {
public:
  void print() {cout << i<<endl;}
};

int _tmain(int argc, _TCHAR* argv[])
{  X x;
  X *px=&x;
  Y y;
  x.seti(10);
  y.seti(15);
  px->print();
```

```
  px=&y;
  px->print();
 return 0;
}
```

შეკითხვა 211

რას შედეგს გამოიტანს მოცემული პროგრამა?

```
class figure {
protected:
  double x, y,r;
public:
  figure(double a=0, double b=0) {x = a; y = b;}
  virtual void area()   {cout<< x*y<<",";}
 };
class circle : public figure {
public:
circle(double a)  {r=a;};
  void area() {cout<<3.14*r*r;}
};

int _tmain(int argc, _TCHAR* argv[])
{
   figure f(3,4);
   circle cir(2);
  figure *p=&f;
  p-> area() ;
```

```
  p=&cir;
  p->area();
}
```

შეკითხვა 212

რას შედეგს გამოიტანს მოცემული პროგრამა?

```
class A{
public:
    virtual int f() = 0;
};
class B : public A{
public:
    int f(){
        cout << "3" <<",";
        return 0;
    }
};
class C : public A{
public:
    int f(){
        cout << "2" << endl;
        return 0;
    }
};
 int _tmain(int argc, _TCHAR* argv[])
{  B b;
```

```
   C c;
   A * a;
   a = &b;
  a->f();
  a=&c;
  a->f();
}
```

შეკითხვა 213

რას შედეგს გამოიტანს მოცემული პროგრამა?

```
class A
{
public:
    A() { f(); }
    virtual void f()
    {
        std::cout << "A::f";
    }
};
class B : public A
{
public:
    void f()
    {
        std::cout << "B::f";
    }
```

```
};
 int _tmain(int argc, _TCHAR* argv[])
{  A * a = new B();
}
```

შეკითხვა 214

რას შედეგს გამოიტანს მოცემული პროგრამა?

```
class A
{
public:
    A() { f(); }
    virtual void f()
    {
        std::cout << "A::f"<<",";
    }
};
class B : public A
{
public:
    void f()
    {
        std::cout << "B::f";
    }
};
int _tmain(int argc, _TCHAR* argv[])
{  A * a = new B();
```

```
   a->f();
}
```

შეკითხვა 215

რას შედეგს გამოიტანს მოცემული პროგრამა?

```
class Base {
public:
virtual void who() {
cout << "base,";
}
};
class first_d: public Base {
public:
void who() {
cout << "1_der,";
}
};
class seconded: public Base {
public:
void who() {
cout << "2_der"<<endl;
}
};
int _tmain(int argc, _TCHAR* argv[])
{
Base base_obj;
```

```
Base *p;
first_d first_obj;
seconded second_obj;
p = &base_obj;
p->who();
p = &first_obj;
p->who();
p = &second_obj;
p->who();
}
```

შეკითხვა 216

რას შედეგს გამოიტანს მოცემული პროგრამა?

```
class Base {
public:
virtual void who() {
cout << "Base,";
}
};
class first_d: public Base {
public:
void who() {
cout << "First,";
}
};
class second_d: public Base {
```

```
};
int _tmain(int argc, _TCHAR* argv[])
{ Base base_obj;
Base *p;
first_d first_obj;
second_d second_obj;
p = &base_obj;
p->who();
p = &first_obj;
p->who();
p = &second_obj;
p->who();
}
```

თემა: მეგობარი ფუნქციები

შეკითხვა 217

ჩამოთვლილთაგან რომელია სწორი განმარტება მეგობარ ფუნქციასთან მიმართებით?

a) friend ფუნქცია არ არის კლასის წევრი, მაგრამ მას აქვს წვდომა კლასის დახურულ და დაცულ წევრებთან

a) friend ფუნქცია არის კლასის წევრი, მას აქვს წვდომა კლასის დახურულ და დაცულ წევრებთან

b) friend ფუნქცია არის კლასის წევრი, მაგრამ მას არა აქვს წვდომა კლასის დახურულ და დაცულ წევრებთან

c) friend ფუნქცია არ არის კლასის წევრი, მაგრამ მას აქვს წვდომა კლასის მხოლოდ დაცულ წევრებთან

შეკითხვა 218

ერთ-ერთი მიზეზი თუ რატომ უშვებს C++ ფუნქცია-მეგობრების არსებობას დაკავშირებულია სიტუაციასთან, როდესაც:

a) ორმა კლასმა უნდა გამოიყენოს ერთიდაიგივე ფუნქცია
b) ორმა კლასმა არ უნდა გამოიყენოს ერთიდაიგივე ფუნქცია
c) ორმა კლასმა უნდა გამოიყენოს სხვადასხვა ფუნქცია
d) ორმა კლასმა უნდა განსაზღვროს სხვადასხვა ფუნქცია

შეკითხვა 219

ჩამოთვლილთაგან რომელი არ არის კლასის წევრი?

a) სტატიკური ფუნქია
b) მეგობარი ფუნქცია
c) ფუნქცია მუდმივები
d) ვირტუალური ფუნქცია

შეკითხვა 220

ჩამოთვლილთაგან რომელი არ შეიძლება იყოს მეგობარი?

a) ფუნქცია
b) კლასი
c) ობიექტი
d) მონაცემ-წევრი

შეკითხვა 221

რომელია სწორი განმარტება მოცემული გამოცხადების მიხედვით?

```
    class cl {
public:
friend void frd();
    …….
};
```

a) ფუნქცია frd() გამოცხადებულია cl კლასის მეგობრად
b) ფუნქცია frd() გამოცხადებულია კლასის მეგობრად, რომელიც ჯერ არ არის გახსნილი
c) ფუნქცია frd() გამოცხადებულია კლასის მეგობრად, რომელიც შემდგომში უნდა გაიხსნას
d) ფუნქცია frd() არის cl კლასის ვირტუალური მეგობარი

შეკითხვა 222

რომელია სწორი განმარტება მეგობარ ფუნქციებთან მიმართებით?

a) `friend` ფუნქცია გამოცხადდება კლასის დახმარებით, რომელიც წარუდგენს წვდომას
b) `friend` ფუნქცია გამოცხადდება კლასის დახმარებით, რომელიც წვდომას არ წარადგენს
c) `friend` ფუნქცია არ გამოცხადდება კლასის დახმარებით
d) `friend` ფუნქცია არ გამოცხადდება სხვა კლასის დახმარებით

შეკითხვა 223

კლასის რომელ ნაწილში შეიძლება `friend` ფუნქციის გამოცხადება?

a) კლასის ნებისმიერ ნაწილში, რადგან მასზე არ მოქმედებს წვდომების მართვის გასაღებური სიტყვები
b) კლასის ღია (საჯარო) ნაწილში
c) კლასის დახურულ (კერძო) ნაწილში
d) კლასის დაცულ ნაწილში

შეკითხვა 224

ჩამოთვლილთაგან რომელი გამონათქვამია მართებული?

a) კლასის ფუნქცია-წევრი შეიძლება გამოცხადებულ იქნას სხვა კლასში როგორც მეგობარი
b) კლასის ფუნქცია-წევრი არ შეიძლება გამოცხადებულ იქნას სხვა კლასში
c) კლასის ფუნქცია-წევრი ვერ გამოცხადდება სხვა კლასში როგორც მეგობარი
d) კლასის ყველა ფუნქცია წევრი არის მეგობარი და იგი არ საჭიროებს სხვა კლასში გამოცხადებას

შეკითხვა 225

მოცემული პროგრამიდან გამომდინარე, რომელი განმარტებაა მართებული?

```
class Point
{
    friend void ChangePrivate( Point & );
public:
    Point( void ) : m_i(0) {}
    void PrintPrivate( void ){cout << m_i << endl; }

private:
    int m_i;
};
  void ChangePrivate ( Point &i ) { i.m_i++; }
int _tmain(int argc, _TCHAR* argv[])
{  Point sPoint;
   sPoint.PrintPrivate();
   ChangePrivate(sPoint);
   sPoint.PrintPrivate();
  system("pause");
 return 0;
```

}

a) ChangePrivate ფუნქციას აქვს წვდომა Point კლასის დახურულ მონაცემ-წევრზე
b) კლასის მეგობარ ფუნქციას აქვს წვდომას ამ კლასის დახურულ მონაცემ-წევრზე
c) კლასის მეგობარ ფუნქციას არა აქვს წვდომა ამ კლასის დახურულ მონაცემ-წევრზე
d) კლასის მეგობარი ფუნქცია არასწორად არის განსაზღვრული

შეკითხვა 226

რას გამოიტანს მოცემული პროგრამა?

```
class Point
{
    friend void ChangePrivate( Point & );
public:
    Point( void ) : m_i(0) {}
    void PrintPrivate( void ){cout << m_i << ",";
}
private:
    int m_i;
};
  void ChangePrivate ( Point &i ) { i.m_i++; }
int _tmain(int argc, _TCHAR* argv[])
{  Point sPoint;
   sPoint.PrintPrivate();
   ChangePrivate(sPoint);
   sPoint.PrintPrivate();
       return 0;}
```

შეკითხვა 227

რა შეიძლება იყოს მოცემულ ფრაგმენტში შეცდომის მიზეზი?

```
class B;
class A {
public:
   int Func1( B& b );

private:
   int Func2( B& b );
};

class B {
private:
   int _b;
   friend int A::Func1( B& );
};
int A::Func1( B& b ) { return b._b; }
int A::Func2( B& b ) { return b._b; }
.....
```

a) დახურულ _b წევრზე წვდომა არ არის დაშვებული A კლასის Func1 ფუნციისთვის
b) დახურულ _b წევრზე წვდომა არ არის დაშვებულია A კლასის Func2 ფუნციისთვის
c) A კლასის Func1 არ შეიძლება გამოცხადდეს B კლასის მეგობრად
d) A კლასის Func2 არ შეიძლება გამოცხადდეს B კლასის მეგობრად

შეკითხვა 228

მოცემული ფრაგმენტიდან გამომდინარე, რომელი განმარტებაა მართებული?

```
class line;
class box {
int color;
int upx, upy;
int lowx, lowy;
public:
friend int same_color (line l, box b);
void set_color(int c);
void define_box (int x1, int y1, int x2, int y2);
void show_box();
};
class line {
int color;
int startx, starty;
int len;
public:
friend int same_color (line I, box b);
void set_color(int c);
void define_line (int x, int y, int I);
void show_line();
};
```

.....

a) same_color() ფუნქცია არცერთი კლასის წევრი არ არის, მაგრამ ორივე კლასის მეგობარია
b) same_color() ფუნქცია box კლასის წევრია
c) same_color() ფუნქცია line კლასის წევრია
d) same_color() ფუნქცია ორივე კლასის წევრია

შეკითხვა 229

რას გამოიტანს მოცემული პროგრამა?

```
class b;
class a {
```

```
    friend int sum (a,b);
private:
int i;
public:
a() {i=12; }
};
class b {
friend int sum (a,b);
private:
 int y;
public:
b()
{y=13;}
};
int sum (a first, b second)
{return (first.i+second.y);
}
int _tmain(int argc, _TCHAR* argv[])
{ a first;
b second;
cout<<sum(first, second)<<endl;
 return 0;
}
```

თემა: მეგობარი კლასები

შეკითხვა 230

რა პრივილეგიებს იძლევა ისეთი შემთხვევა, როდესაც ერთ კლასში ცხადდება სხვა კლასი, როგორც მეგობარი?

a) კლასი მეგობარ კლასს აძლევს თავის დახურულ და დაცულ მონაცემებზე პირდაპირი წვდომის საშუალებას
b) კლასი მეგობარ კლასს აძლევს თავის მხოლოდ დახურულ მონაცემებზე წვდომის საშუალებას

c) კლასი მეგობარ კლასს აძლევს თავის მხოლოდ დაცულ მონაცემებზე წვდომის საშუალებას
d) კლასი მეგობარ კლასს არ აძლევს თავის მონაცემებზე წვდომის საშუალებას

შეკითხვა 231

ჩამოთვლილთაგან რომელია სწორი გამონათქვამი?

a) კლასში მეგობარი კლასი ვერ გამოცხადდება
b) კლასში შეიძლება გამოცხადეს სხვა კლასი, როგორც მეგობარი
c) კლასში შეიძლება გამოცხადდეს სხვა კლასები, როგორც ამ კლასის მეგობრები
d) კლასში შეიძლება გამოცხადდეს მხოლოდ მეგობარი ფუნქციები

შეკითხვა 232

ჩამოთვლილთაგან რომელი ჭდე არ თამაშობს არანაირ როლს ხილვადობის თვალსაზრისით კლასთა მემკვიდრეობითობის შემთხვევაში?

a) friend
b) public
c) private
d) protected

შეკითხვა 233

რა შემთხვევაშია მიზანშეწონილი მეგობარი კლასების გამოცხადება?

a) როდესაც ორი კლასი არ არის დაკავშირებული მემკვიდრეობითობით და საჭიროა კლასების მხოლოდ დახურულ ელემენტებზე წვდომა
b) როდესაც ორი კლასი არ არის დაკავშირებული მემკვიდრეობითობით და საჭიროა კლასების დახურულ და დაცულ ელემენტებზე წვდომა
c) როდესაც ორი კლასი არ არის დაკავშირებული მემკვიდრეობითობით და საჭიროა კლასების მხოლოდ დაცულ ელემენტებზე წვდომა
d) როდესაც კლასები მემკვიდრეობითობით არის დაკავშირებული და არ საჭიროებს კლასების დახურულ და დაცულ ელემენტებზე წვდომას

შეკითხვა 234

პროგრამის რომელ ნაწილში უნდა გამოცხადდეს friend კლასი?

a) იმ კლასის შიგნით, რომლის მეგობარიც უნდა იყოს friend ჭდით გამოცხადებული კლასი
b) იმ კლასის გარეთ, რომლის მეგობარიც უნდა იყოს friend ჭდით გამოცხადებული კლასი
c) პროგრამის ნებისმიერ ნაწილში
d) მთავარ ფუნქციაში

შეკითხვა 235

თუ წარმოებული კლასს გამოვაცხადებთ საბაზო კლასის მეგობრად, წვდომებთან დაკავშირებით რა შედეგებს მივიღებთ?

a) საბაზო კლასის დაცული წევრების გამოყენება შეუძლებელი იქნება
b) კლასის დახურულ წევრებზე წვდომა შეუძლებელი იქნება

c) საბაზო კლასის დაცული წევრების გამოყენება და მეგობრული წვდომის წარდგენა კლასის დახურულ წევრებზე ანალოგიურ შედეგებს მოგვცემს

d) შეუძლებელია წარმოებული კლასის გამოცხადება საბაზოს მეგობრად

შეკითხვა 237

ჩამოთვლილთაგან რომელი გამონათქვამია მართებული?

a) კლასში შეიძლება გამოცხადდეს მხოლოდ ერთი მეგობარი და მის წინ დაიწეროს გასაღებური სიტყვა friend

b) კლასში უნდა იქნას ჩამოთვლილი ყველა მისი მეგობარი და მათ წინ დაიწეროს გასაღებური სიტყვა friend

c) კლასში შეიძლება გამოცხადდეს რამდენიმე მეგობარი და მხოლოდ ერთთან უნდა დაიწეროს გასაღებური სიტყვა friend

d) კლასში უნდა გამოცხადდეს მხოლოდ ერთი მეგობარი და მის წინ friend სიტყვის დაწერა არ არის სავალდებულო

შეკითხვა 238

მოცემული ფრაგმენტიდან გამომდინარე რომელი განმარტებაა მართებული?

```
     class AClass {
  friend class BClass;
private:
  double value;
    public:
  AClass() { value = 3.14159; }
};
```

a) `BClass` კლასი `AClass` კლასის მეგობარია

b) `value` მისაწვდომია `AClass` და `BClass` კლასების წევრებისთვის

c) `value` მისაწვდომია მხოლოდ `AClass` კლასის წევრებისთვის

d) `value` არ არის მისაწვდომი არცერთი კლასის წევრებისთვის

შეკითხვა 239

რა შედეგს გამოიტანს შემდეგი პროგრამა?

```
class YourClass {
friend class YourOtherClass;
public:
   YourClass() : topSecret(0){}
   void printMember() { cout << topSecret <<",";
}
private:
   int topSecret;
};

class YourOtherClass {
public:
   void change( YourClass& yc, int x
){yc.topSecret = x;}
};
int _tmain(int argc, _TCHAR* argv[])
{   YourClass yc1;
   YourOtherClass yoc1;
   yc1.printMember();
   yoc1.change( yc1, 5 );
   yc1.printMember();
 return 0;}
```

a) 0,5

b) 0,0
c) 5,5
d) 5,0

შეკითხვა 240

რა შედეგს გამოიტანს შემდეგი პროგრამა?

```
class Pal {
  friend class Buddy;
 private:
 int x;
 protected:
  void doublex(void) {x *= x;}
 public:
 Pal() {x = 100;}
Pal(int n) {x = n;}
   };
class Buddy {
private:
  Pal palObject;
 public:
 void ShowValues(void);
 };

int _tmain(int argc, _TCHAR* argv[])
{   Buddy aBuddy;
   aBuddy.ShowValues();
 return 0;
}
void Buddy::ShowValues(void)
{
  Pal aPal(1234);
  cout << "palObject.x=" << palObject.x<<",";
```

```
    palObject.doublex();
    cout << "palObject.x ="<< palObject.x<<",";
    cout << "Pal.x =" << aPal.x <<'\n';}
```

a) palObject.x=100, palObject.x=1234, Pal.x=10000
b) palObject.x=1234, palObject.x=10000, Pal.x=100
c) palObject.x=100, palObject.x =10000, Pal.x =1234
d) palObject.x=100, palObject.x=10000, Pal.x=100

შეკითხვა 241

რა შედეგს გამოიტანს შემდეგი პროგრამა?

```
class MyClass
{
 friend class SecondClass;
 public:
       MyClass() : Secret(7){}
       void printMember()
       {
             cout << Secret << ",";
       }
 private:
   int Secret;
};
class SecondClass
{
 public:
       void change( MyClass& yourclass, int x )
       {
             yourclass.Secret = x;
       }
};
 int _tmain(int argc, _TCHAR* argv[])
{  MyClass my_class;
 SecondClass sec_class;
```

```
my_class.printMember();
sec_class.change( my_class, 12 );
my_class.printMember();
}
```

a) 7,12
b) 12,7
c) 7, 7
d) 12,12

შეკითხვა 242

რა შედეგს გამოიტანს შემდეგი პროგრამა?

```
class person;
class dog {
  friend class person;
private:
int health;
};
class person {
public:
  void damage (dog& d) {d.health=100;
d.health-=20;
cout<<d.health<<endl;
}
};
 int _tmain(int argc, _TCHAR* argv[])
{ dog skuby;
person  pers1;
pers1.damage(skuby);
system("pause");
return 0;
}
```

a) 100
b) 20

c) 80

d) -20

შეკითხვა 243

მოცემული ფრაგმენტის მიხედვით რომელი მინიჭება შეიძლება გამოვიყენოთ change ფუნქციაში?

```
class A{
private:
int m_a;
friend class B;
};
class B{
public:
void change( A *aptr, int a ){……}
  ……
  };
```

a) aptr->m_a = a

b) aptr.m_a = a

c) *aptr = a

d) არცერთი მინიჭება

შეკითხვა 244

მოცემული ფრაგმენტის მიხედვით რომელია განსაზღვრაა მართებული?

```
class Y
{
friend class X;
friend class Z;
};
class Z {…};
class X {…};
```

```
class E {...};
.....
```

a) Z და X კლასები არ არიან Y კლასის მეგობრები
b) Z და X კლასებია არიან Y კლასის მეგობრები
c) E კლასი არ არის Y კლასის მეგობარი
d) E კლასი არის Y კლასის მეგობარი

შეკითხვა 245

მოცემული ფრაგმენტიდან გამომდინარე რომელია სწორი მიმართვა კლასის მონაცემ-ელემენტზე?

```
class B {
    friend class A;
private:
    int i;
};
class A {
public:
    A(B b) {
        b.i = 0;
    }
};
class C {
public:
    C(B b) {
        b.i = 0;
    }
};
.....
```

a) class C {.....b.i = 0; }
b) ორივე სწორია
c) class A {.....b.i = 0; }
d) არცერთი არ არის სწორი

შეკითხვა 246

რა შედეგს მოგვცემს მოცემული პროგრამა?

```
class Square;
class Rectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (Square a);
};
class Square {
  friend class Rectangle;
  private:
    int side;
  public:
    Square (int a) : side(a) {}
};
void Rectangle::convert (Square a) {
  width = a.side;
  height = a.side;
}
 int _tmain(int argc, _TCHAR* argv[])
{ Rectangle rect;
  Square sqr (4);
  rect.convert(sqr);
  cout << rect.area();
return 0;
}
```

ლიტერატურა

1. Роберт Лафоре `Object-Oriented Programming in c++", изд. Питер, Классика Computer Science, ISBN 978-5-4237-0038-6, 0-672-32308-7; 2011г.
2. Харви Дейтел, Пол Дейтел `Как программировать на c++~, стр.1037.
3. Грэхем И. Объектно-ориентированные методы: Принципы и практика: пер. с англ. Изд. 3-е. М: Вильямс, 2004. 880 с.
4. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2003. 368 с.
5. Элиенс А. Принципы объектно-ориентированной разработки программ. М.: Вильямс, 2002. 496 с.
6. Кендалл Скотт. UML. Основные концепции. Пер. с англ. М.: Издательский дом «Вильямс», 2002.144 с.: ил.
7. Учебный курс «Основы объектно-ориентированного программирования». http://www.intuit.ru/department/se/oopbases/