2. Предефинирани функции. Приятелски функции

1. Предефинирани функции.

Езикът C++ позволява функции с едно и също име да имат различно съдържание и действие. При обръщение към функция с дадено име се зарежда тази от едноименните функции, която отговаря най-точно по брой, тип и ред на аргументите.

Задача 1. Да се дефинира отново клас triangle, като се добави още една член-функция, съименник на член-функция face(), която се отличава по наличието на аргумент и по действие – освен лицето, тя връща и периметъра на триъгълника.

```
#include <iostream.h>
#include <math.h>
class triangle {
    private:
       double a, b, c;
    public:
       triangle();
       double face();         //Член-функция face() без аргумент
       double face(double *); //Член-функция face() с един аргумент
       void show(char *);
       ~triangle()  { cout << "\n\nDestructing object triangle!\n"; }
};
triangle::triangle()          //Дефиниция на конструктора
{do
  { cout << "\n\nEnter three values for the sides of triangle:\n";
    cin >> a >> b >> c;
   }
   while(!((a>0)&&(b>0)&&(c>0)&&((a+b>c)&&((a+c)>b)&&((b+c)>a))));
}
double triangle::face()          //Дефиниция на член-функция face() без аргументи

{
   double p, s;
   p = (a+b+c)/2;
   s = sqrt(p*(p-a)*(p-b)*(p-c));
   return s;
}
double triangle::face(double *p) //Предефинирана член-функция face()
{
   double pp = a+b+c;
   *p = pp;       //Връщане на втория резултат чрез аргумент-указател
   pp /= 2;
   double s;
   s = sqrt(pp*(pp-a)*(pp-b)*(pp-c));
   return s;
}
void triangle::show(char *name) {
   cout << "Sides of the triangle " << name << ":\n";
   cout << "a=" << a << ",b=" << b << ",c=" << c;
}
void main()
{
triangle tr1;             //Обект tr1; извиква се конструкторът на класа
double p,s;
tr1.show("tr1");
cout << "\nThe face of triangle1 is s=" << tr1.face(); //Зарежда се член-функция
face() без
                                                      // аргумент
triangle tr2;             //Обект tr2; извиква се конструкторът на класа
tr2.show("tr2");
s=tr2.face(&p);           //Зарежда се член-функция face() с аргумент
cout << "\nThe face of triangle2 is s=" << s << ", and the perimeter is p=" << p;
}
```

Резултати:

```
Въведете три стойности за страните на триъгълник: 30 40 50
Страни на триъгълника tr1:
a=30, b=40, c=50
Лицето на триъгълник1 е s=600
Въведете три стойности за страните на триъгълник: 10 10 5
Страни на триъгълника tr2:
a=10, b=10, c=5
Лицето на триъгълник2 е s=24.206, а периметъра е p=25
```

2. Приятелската функция не е член-функция на класа, но тя има достъп до *private* членовете на класа. Приятелски функции се използват, когато е необходимо една функция да има достъп до *private* членовете на *два или повече* различни класа.
Декларацията на приятелската функция може да се намира както в частта *public*, така и в частта *private* и се характеризира с ключовата дума **friend**, която се поставя в началото на декларацията. Една приятелска функция се дефинира като обикновена функция, която не е член на клас.

2.1. Независима функция – приятелска за два или повече класа.
Тази функция трябва да се декларира във всеки от класовете като приятелска. Приятелските функции не са член-функции на класа и затова те не получават като първи параметър по неявен начин **this**. Поради тази причина е необходимо подаването на обект по явен начин като параметър на функцията.

```
friend int sp_greater(car c, truck t);
```

Извикването на приятелска функция се извършва чрез името на функцията и списък от фактически параметри:

```
sp_greater(c1, t1);
```

Задача 2. Следната програма създава клас **car** и клас **truck**, всеки от които съдържа *private* променлива, която определя скоростта на превозното средство. Тази програма съдържа функцията **sp_greater()**, която е приятелска функция и на **car** и на **truck**.

```
#include <iostream.h>

class truck;    //Предварителна декларация

class car {
      int passengers;
      int speed;
    public:
      car(int p, int s) { passengers = p; speed = s; }

      friend int sp_greater(car c, truck t);
};

class truck {
      int weight;
      int speed;
    public:
      truck(int w, int s) { weight = w; speed = s;}

      friend int sp_greater(car c, truck t);
};
```

```
/* Връща положителна стойност, ако скоростта на колата е по-голяма от
тази на камиона. Връща 0, ако скоростите са еднакви. Връща отрицателна
стойност, ако скоростта на камиона е по-голяма от тази на колата. */

int sp_greater(car c, truck t)
{
  return c.speed - t.speed;
}

int main()
{
int t;
car c1(6, 55), c2(2, 120);
truck t1(10000, 55), t2(20000, 72);

cout << ”Comparing c1 and t1:\n”;
t = sp_greater(c1, t1);
if(t<0) cout << ”Truck is faster.\n”;
else if(t==0) cout << ”Car and truck speed is the same.\n”;
else cout << ”Car is faster.\n”;

cout << ”\nComparing c2 and t2:\n”;
t = sp_greater(c2, t2);
if(t<0) cout << ”Truck is faster.\n”;
else if(t==0) cout << ”Car and truck speed is the same.\n”;
else cout << ”Car is faster.\n”;

return 0;
}
```

Тъй като **sp_greater()** приема параметри едновременно от класовете **car** и **truck**, то е невъзможно да се декларират и двата класа, преди **sp_greater()** да бъде включена в тях. За да се укаже на C++ компилатора, че даден идентификатор представлява име на клас, се използва предварителна декларация:

```
  class truck;
```

2.2. Член-функцията на един клас – приятелска за друг клас.

При декларирането на приятелската функция трябва да се прецизира класът, на който тя е член-функция с помощта на оператор за принадлежност(::).

Задача 3. Предния пример, написан така, че **sp_greater()** да е член-функция на **car** и приятелска функция на **truck**, ще изглежда по следния начин:

```
#include <iostream.h>

class truck;    //Предварителна декларация

class car {
      int passengers;
      int speed;
    public:
      car(int p, int s) { passengers = p; speed = s;}

      int sp_greater(truck t);
};

class truck {
      int weight;
      int speed;
```

```
    public:
      truck(int w, int s) { weight = w; speed = s;}

      friend int car::sp_greater(truck t);
};



/* Връща положителна стойност, ако скоростта на колата е по-голяма от
тази на камиона. Връща 0, ако скоростите са еднакви. Връща отрицателна
стойност, ако скоростта на камиона е по-голяма от тази на колата. */

int car::sp_greater(truck t)
{
  return speed - t.speed;   // Тъй като sp_greater() е член на car, то
  трябва да се подаде само
}                           // обектът за камион

int main()
{
int t;
car c1(6, 55), c2(2, 120);
truck t1(10000, 55), t2(20000, 72);

cout << "Comparing c1 and t1:\n";
t = c1.sp_greater(t1);      //Извиква се като член-функция на car
if(t<0) cout << "Truck is faster.\n";
else if(t==0) cout << "Car and truck speed is the same.\n";
else cout << "Car is faster.\n";

cout << "\nComparing c2 and t2:\n";
t = c2.sp_greater(t2);      //Извиква се като член-функция на car
if(t<0) cout << "Truck is faster.\n";
else if(t==0) cout << "Car and truck speed is the same.\n";
else cout << "Car is faster.\n";

return 0;
}
```

В този случай се използва оператора за принадлежност(::) в декларацията на приятелската функция в класа **truck**, за да укаже на компилатора, че функцията **sp_greater()** е член на класа **car**.

Задачи за самостоятелна работа:

Задача 1: Да се дефинира клас за описание на правоъгълник (по аналог на класа triangle) с две член-функции face(), която се отличава по наличието на аргумент и по действие – освен лицето, тя връща и периметъра на правоъгълника.
Задача 2: Разширете функционалността на задачи 2 и 3, като имплементирате друга приятелска функция, която да изчислява времето за изминаване на зададени километри (S = V * T).

Допълнителни задачи:

Задача 1: Напишете клас Stack, който изобразява стек. Напишете функцията функцията loadstack(), която връща като резултат стек, напълнен със символите от азбуката (a-z). Предефинирайте функцията така, че да приема като параметър стойност от тип int с име upper. Ако upper е 1, попълнете

стека с голями букви от азбуката. В противен случай го попълнете с малки букви.

Задача 2: Представете си ситуация, в която два класа pr1 и pr2 си поделят един принтер. След това си представете, че други части от вашата програма трябва да знаят кога принтерът се използва от обект от някой от тези два класа. Създайте функцията inuse(), която връща резултат true, ако принтерът се използва от някой от двата класа и false във всички останали случаи. Направете тази функция приятелска и за двата класа.

```
class pr1 (
        int printing;
        // …
       public:
         pr1() {printing = 0;}
         void set_print(int status) {printing = status;}
        // …
};

class pr2 (
        int printing;
        // …
       public:
         pr2() {printing = 0;}
         void set_print(int status) {printing = status;}
        // …
};
```