**24.04.2024 года Объекты Функции**

**Объекты как параметры внешних функций и функций класса. Передача массива объектов функции. Дружественные функции. Возвращение объекта из функции. Встраиваемые функции – члены класса.**

**Объекты как параметры внешних функций и функций класса**

Для того чтобы передать объект в функцию, следует задать параметр функции, тип которого класс, и при вызове функции использовать объект этого класса в качестве аргумента.

**Передача объекта функции по значению**. Функция работает с копией объекта. Поэтому изменение полей копии объекта внутри функции не влияет на сам *объект-оригинал*. При этом ***конструктор создаваемой копии не вызывается***, так как конструктор используется для инициализации полей объекта, а копия создаётся для уже существующего (а значит – ***проинициализированного***) объекта.

При завершении работы функции созданная копия объекта удаляется и вызывается ***неявный деструктор*** копии объекта.

При передаче в функцию объекта *по значению* создаётся копия объекта. **Конструктор**копии не вызывается*.* **Деструктор** копии вызывается.

**Передача объекта функции по указателю.** В функцию передаётся адрес объекта. При этом функция может изменить значение полей объекта. Для доступа к полям объекта в функции используется *операция* *стрелка* **(->)** или *операция* *точка* **(.),** если разыменовать указатель.

**Передача объекта функции по ссылке.** Изменения объекта внутри функции влияют на исходный объект и сохраняются при завершении работы функции. Для доступа к полям объекта в функции используется *операция* *точка* **(.)**.

**// Пример 1.** Передача объекта **внешней функции** по значению, указателю,

// ссылке.

#include <iostream>

using namespace std;

class **Samp**{

int num;

**public:**

Samp(int x){num = x;} // конструктор

void set(int x){num = x;} // функция установки **num**

int get(){return num;} // функция возврата **num**

};

void **funValue**(Samp zob); // прототип **funValue()**,

void **funPointer**(Samp\* p); // прототип  **funPointer()**

void **funLink**(Samp &ref); // прототип **funLink()**

int **main()**{

Samp ob(10); // создание объекта **ob**

funValue(ob); // передача **ob** по **значению**

cout<<"ob.num = "<<ob.get()<<endl; // ob.num = **10**

funPointer(&ob); // передача адреса **ob**

cout<<"ob.num = "<<ob.get()<<endl; // ob.num = **60**

funLink(ob); // передача **ob** по **ссылке**

cout<<"ob.num = "<<ob.get()<<endl; // ob.num **= 70**

}

void **funValue**(Samp zob){ // функция **funValue()**,

zob.set(50);

cout<<"zob.num = "<<zob.get()<<endl; // **zob.num** = **50**

}

void **funPointer**(Samp\* p){ // функция **funPointer()**

p->set(60); // или (\***p).**set(60)

cout<<"p->num = "<<p->get()<<endl; // p->num = 60

}

void **funLink**(Samp &ref){ // функция **funLink()**

ref.set(70);

cout<<"ref.num = "<<ref.get()<<endl; // **ref. num** = **70**

}

// **Пример 2.** Определить класс для работы с рациональной дробью.

// Для нахождение суммы двух дробей использутся **внешняя** функция

// **sumFracExt()** и функция **класса** **sumFracCl()**.

#include <iostream>

using namespace std;

class **Frac**{

float a, b;

**public:**

Frac(float n, float m){a = n; b = m;} // конструктор с параметрами

void show(){cout<<"a /b = "<<a<<**'**/**'**<<b<<endl;} // функция **show()**

float **sumFracCl**(Frac o2); // функция **sumFracCl()**

float geta(){return a;} // функция **geta()**

float getb(){return b;} // функция **getb()**

};

float **sumFracExt**(Frac o1, Frac o2); // прототип **sumFracExt()**

int **main**(){

float x,y;

cout<<" Enter x, y for **ob1**: "; cin>>x>>y; // **1 2**

Frac ob1(x, y); // создание **ob1**

ob1.show();

cout<<" Enter x, y for **ob2:** "; cin>>x>>y; // **3 4**

Frac ob2(x, y); // создание **ob2**

ob2.show();

float rez = **sumFracExt**(ob1, ob2); // передаются 2 объекта

cout<<"rez = "<<rez<<endl; // **rez = 1.25**

float rez1 = **ob1**.**sumFracCl**(ob2); // **ob1** вызывает функцию

// **ob2** передаётся как аргумент

cout<<"rez1 = "<<rez1<<endl; // **rez1** = **1.25**

}

float **sumFracExt**(Frac o1, Frac o2){ // **sumFracExt() -** внешняя

float z = (o1.geta()\*o2.getb() + o1.getb()\*o2.geta()) / (o1.getb()\*o2.getb());

return z;

}

float Frac:: **sumFracCl**(Frac o2){ // **sumFrac() –** член класса

float z = (a \* o2.b + b \* o2.a ) / (b \* o2.b);

// float z = (this->a \* o2.b + this->b \* o2.a) / (this->b \* o2.b);

return z;

}

В **Примере 2** во внешнюю функцию **sumFracExt()** передаются **2** объекта по значению. Для доступа к закрытым полям класса **a** и **b** используются функции доступа – **geta()** и **getb().**

В функцию класса **sumFracCl()** передаётся только один объект (**ob2**). **Почему?** Потому что текущий объект (**ob1**) станет известен функции через указатель **this**, т. к. он вызывает функцию.

**Передача массива объектов функции**

Функции можно передавать адреc массива объектов аналогичным образом, как это мы делали для числовых массивов, массивов структур.

// **Пример 3**. Определить класс **Student:** фамилия, дата рождения, курс.

// Создать динамический ***массив объектов*** данного класса.

// Вывести список всех студентов, студентов заданного факультета.

#include <iostream>

#include <cstring>

#include <iomanip>

using namespace std;

const int m = 10;

struct **Date**{ // структура **Date** - дата рождения

int mon;

int year;

};

class **Student**{ // класс **Student**

char fam[m];

**Date** t;

char fac[m];

**public:**

Student();

~Student(){ cout << "~Student()" << endl; }

char \* getfac(){return fac;};

void show();

};

void **spisAll**(Student\* spis, int n); // прототип **spisAll()**

void **spisFac**(Student\* spis, int n); // прототип **spisFac()**

int **main(){**

int n;

cout<<"Enter a number of students: "; cin>>n;

Student\* spis = new Student[n]; //динамический массив объектов

**spisAll**(spis, n); //вызов **spisAll**()

**spisFac**(spis, n); //вызов **spisFac**()

delete [] spis;

}

Student::Student(){ // конструктор

cout << "Input fam: "; cin >> fam;

cout << "Input month of birthday: "; cin >> t.mon;

cout << "Input year of birthday: "; cin >> t.year;

cout << "Input facultet: "; cin >> fac;

}

void Student::**show()**{ // функция **show()**

cout<<"| "<<setw(9)<<fam<<" | "<<setw(4)<<t.mon<<" | ";

cout<<setw(5)<<t.year<<" | "<<setw(4)<<fac<<" | "<<setw(2)<<endl;

}

void shapka(){ // функция **shapka()**

cout<<"|----------------------------------------|"<<endl;

cout<<"| Family | month | year | fac |"<<endl;

cout<<"|----------------------------------------|"<<endl;

}

void **spisAll**(Student\* spis, int n){ // функция **spisAll**()

shapka();

for(int i = 0; i < n; i++)

spis[i].show();

}

void **spisFac**(Student\* spis, int n){ // функция **spisFac**()

char fac[m];

cout<<" Students the specified faculty:"<<endl;

cout << "Input faculty: "; cin >> fac;

shapka();

for(int i = 0; i < n; i++)

if(**strcmp**(spis[i].getfac(), fac) == 0) spis[i].show();

}

В классе **Student** определен деструктор, который просто уведомляет о его вызове.

**Результат:**

Input a number of students: 3

**Input family: aaaa** Input month of birthday: **3**

Input year of birthday: **2000** Input faculty: **mmf и т. д.**

**vivod studentov zadannogo faculty**

**Input faculty**: **mmf**

|-------------------------------------------|

| Family | month | year | fac |

|-------------------------------------------|

| aaaa | 3 | 2000 | mmf |

**~**Student() **~**Student()**~** Student()

**Дружественные функции**

В общем случае ***только*** функции класса имеют доступ к **закрытым членам** этого класса. Однако в языке **С++** существует возможность ***разрешить доступ к закрытым членам*** класса функциям, которые не являются членами класса, объявив эти функции ***дружественными*** по отношению к рассматриваемому классу. Для этого необходимо включить в класс ***прототип*** функции с ключевым словом **friend**.

// **Пример 4.** Использование **friend**-функции класса.

#include <iostream>

using namespace std;

class **Samp**{

int num1, num2;

**public:**

Samp(int n, int m); // конструктор с параметрами

void show(); // функция вывода s**how()**

**friend** int **sumAB**(Samp obf); // **дружественная** функция **sumAB()**

};

int **main()**{

int n1, n2;

cout<<"Enter n1, n2: "; cin>>n1>>n2; // **5 6**

Samp **ob**(n1, n2);

int sum = **sumAB(ob)**;

ob.show(); // num1 = **5**  num2 = **6**

cout<<" sum = "<<sum<<endl; // sum = **11**

}

Samp::Samp(int n, int m){ // определение конструктора

num1 = n; num2 = m;

}

void Samp::**show(){**

cout<<"num1 = "<<num1<<" num2 = "<<num2<<endl;

}

int **sumAB**(Samp obf){ // определение **sumAB()**

int sum = obf.num1 + obf.num2;

return sum;

}

В **Примере 4** объект **ob б**ыл передан как аргумент в **дружественную** функцию **sumAB(),** поэтому в функции разрешён доступ к закрытым членам класса **Samp,** но только через **объект.** Как видно из примера, функция **sumAB()** вызывается ***обычным образом***, её **нельзя** вызвать, используя имя объекта и *операцию точка* **(.)** или *стрелка* **(->)**, так как она ***не является членом*** класса.

**Дружественная функция** – это функция, которая ***не является членом* *класса***, но имеет доступ ***ко всем членам класса****,* но только через объект

В следующем примере показано, что функция может быть дружественной более, чем к одному классу.

**// Пример 5.** Функция **fun()** дружественна классу **A** и классу **B**.

#include <iostream>

using namespace std;

class **B**; // объявление класса **B**

class **A{** // определение класса **A**

int data;

**public:**

A(){data = 3;}

**friend** int fun(A obx, B oby); // дружественная функция **fun()**

};

class **B{** // определение класса **B**

int data;

**public:**

B(){data = 7;}

**friend** int fun(A oba, B obb); // дружественная функция **fun()**

};

int **main(){**

A oba; // oba.data = **3**

B obb; // obb.data = **7**

cout<<" oba.data + obb.data = "<<**fun(oba, obb)**<<endl; // **10**

**}**

int **fun(**A obx, B oby){ // функция **fun()**

return(obx.data + oby.data);

}

Следует обратить внимание на то, что в примере функция **fun()** определена ***только один раз*,** а также используется *опережающее объявление* для класса **B** – **class B**; в связи с тем, что параметром функции **fun()** являетсяобъект класса **B** до определения класса.

Следующий пример показывает, что функция также может быть ***членом одного класса* и *дружественной другому классу*.**

**// Пример 6**. Функция **fun()** член класса **A** и дружественна классу **B**.

#include <iostream>

using namespace std;

class **B**;

class **A**{

int data;

**public:**

A(){data = 3;} // конструктор

int **fun**(B oby); // функция **fun()** – **член** класса **A**

};

class **B**{

int data;

**public:**

B(){data = 7;} // конструктор

**friend** int **A::fun(B oby)**; // функция **fun()** – **друг** класса **B**

};

int **main()**{

A oba; B obb;

// вызов функции **fun()** через операцию **(.)** и объект класса **A**

cout<<" oba.data + obb.data = **"<< oba.fun(obb**)<<endl; // **10**

}

int A::fun(B oby){

return(data + oby.data);

}

// **Пример 7.** Определить класс для работы с рациональной дробью.

// Для нахождение суммы двух дробей используется **дружественная** функция

// класса.

#include <iostream>

using namespace std;

class **Frac**{

float a, b;

**public:**

Frac(float n, float m){a = n; b = m;} // конструктор с параметрами

void show(){cout<<"a /b = "<<a<<**'**/**'**<<b<<endl;}

**friend** float **sumFrac**(Frac o1, Frac o2);

};

int **main**(){

float x,y;

cout<<"Enter a and b for ob1: "; cin>>x>>y; // **1 2**

Frac **ob1**(x, y); ob1.show();

cout<<"Enter a and b for ob2: "; cin>>x>>y; // **3 4**

Frac **ob2**(x, y); ob2.show();

float rez = **sumFrac**(ob1, ob2); // вызов функции как внешней

cout<<"rez = "<<rez<<endl; // **rez = 1.25**

}

float **sumFrac**(Frac o1, Frac o2){ // **sumFrac() – друг** класса

float z = (o1.a \* o2.b + o1.b \* o2.a ) / (o1.b \* o2.b);

return z;

}

**Правила использования дружественных функций**:

* функция не является членом класса, но имеет доступ ко всем (в том числе и к ***закрытым*)** членам класса, ***используя объект***, объявленный внутри функции или переданный ей как аргумент;
* функция ***вызывается*** как обычная функция;
* прототип функции может быть объявлен как в открытой, так и в закрытой части класса, но функция будет ***всё равно открытой***;
* функция может быть дружественной более чем к одному классу;
* функция может быть членом одного класса и дружественной другому;
* дружественной функции указатель **this** **не передаётся**.

**Возвращение объекта из функции**

Объекты можно не только передавать в функцию как аргументы, но и возвращать их из функции.

// **Пример 9.** Определить класс для работы с рациональной дробью.

// Создать функцию для сложения двух объектов-дробей. Функция возвращает

// **объект**. Используется **friend**-функция.

#include <iostream>

using namespace std;

class **Frac**{

double a, b;

**public:**

Frac(){}; // конструктор

Frac(double n, double m); // конструктор

void show() {cout<<a<<"/"<<b<<endl;} // функция **show()**

**friend** Frac **sumFrac**(Frac ob1, Frac ob2); // друж. ф. **sumFrac()**

};

Frac::Frac(double n, double m){

a = n; b = m;

}

int **main()**{

double x, y;

cout<<"Enter a and b for **ob1** : "; cin>>x>>y; // **1 2**

Frac **ob1**(x, y); ob1.show();

cout<<"Enter a and b for **ob2** : "; cin>>x>>y; // **3 4**

Frac **ob2**(x, y); ob2.show();

Frac **ob3** = sumFrac(ob1, ob2);

cout<<"sum = "; ob3.show(); // **sum = 10 / 8**

}

**Frac sumFrac**(Frac obx, Frac oby){ // функция возвращает объект

Frac **rab**; // временный объект

rab.a = (obx.a \* oby.b + obx.b \* oby.a);

rab.b = (obx.b \* oby.b);

return **rab**;

}

**Встраиваемые функции – члены класса**

Язык программирования **С++** позволяет задавать функции, которые *встраиваются* в код программы в местах их вызовов, т. е. использование их не связано с механизмами вызова функции, следовательно, работают они гораздо быстрее обычных. Недостатком встраиваемых функций является то, что ***код программы увеличивается***..

Любая функция может стать встраиваемой, включая ***функции-члены*** класса. **Первый** **способ** состоит в использовании спецификатора **inline** перед определением функции. **Второй** **способ** состоит в определении функции ***внутри*** ***определения класса***, если функция достаточно короткая.