**Тема. Массивы объектов. Указатели. Ссылки.**

Массивы объектов. Указатели и объекты, массивы объектов. Адресная арифметика с объектами. Указатель this*.* Ссылка на объект. Динамические объекты, массивы объектов.

**Массивы объектов.** Объекты – это переменные, и они имеют те же возможности и свойства, что и переменные любых других типов. Поэтому вполне допустимо создавать ***массивы объектов***. Синтаксис объявления массива объектов и доступ к его элементам совершенно аналогичны обычным массивам.

**// Пример 1.** Создание и использование одномерного массива объектов.

#include <iostream>

using namespace std;

class **Samp{**

int x;

**public:**

voidsetx(int n){x = n;}

int getx(){return x;}

**};**

int **main(){**

Samp ob[4]; // массив объектов из **4**-х элементов

for(int i = 0; i < 4; i++) // присваивание полям элементов массива

ob[i].setx(i); // значений от **0** до **3**

for(int i = 0; i < 4; i++) // возврат значений полей объектов

cout<< ob[i].getx()<<' '; // и вывод их на экран – **0 1 2 3**

**}**

***Явная инициализация массива объектов.*** Если в классе есть конструктор с параметрами, то массив объектов можно инициализировать явно.

**// Пример 2.** Явная инициализация одномерного массива объектов.

#include <iostream>

using namespace std;

class **Samp{**

int x;

**public:**

Samp(int n){x = n;}

int getx(){return x;}

**};**

int **main(){**

Samp ob[4] = { 1, 2, 3, 4 }; // вызов конструктора 4 раза

for(int i = 0; i < 4; i++)

cout<< ob[i].getx()<<' '; // **1 2 3 4**

}

// **Пример 3**. Определить класс для нахождения суммы цифр натурального

// числа **n**. Сформировать динамический массив объектов данного класса и

// определить функции для работы с созданным массивом для **k** чисел. **Вариант 1.**

#include <iostream>

#include <iomanip>

using namespace std;

class **Cifra**{

int n;

int sum;

**public:**

Cifra(){}; // конструктор по умолчанию

Cifra(int n); // конструктор с параметром

void summa(); // функция получения суммы цифр числа

void show(); // функция вывода

};

Cifra::Cifra(int a){n = a;}

void Cifra::summa(){ // определение функции summa()

sum = 0;

int rab = n;

while(rab){

sum += rab % 10; rab /= 10;

}

}

void Cifra::show(){

cout<<"n = "<<setw(7)<<n<<setw(10)<<" summa cifr = "<<sum<<endl;

}

int **main(){**

int k, chislo;

cout<<"Vvedi kolichestvo chisel: "; cin>>k;

Cifra \* p = new Cifra[k]; // создание динам. массива объектов.

for(int i = 0; i < k; i++){

cout<<"Vvedi chislo: "; cin>>chislo;

Cifra ob(chislo); // создание одного объекта

p[i] = ob;

}

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

p[i].summa();

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

p[i].show();

}

**Результат:**

Vvedi kolichestvo chisel**: 3**

Vvedi chislo: 12345

Vvedi chislo: 5432

Vvedi chislo: 123

n = 12345 summa cifr = 15

n = 5432 summa cifr = 14

n = 123 summa cifr = 6

// **Пример 4**. Определить класс для нахождения суммы цифр натурального

// числа **n**. Сформировать динамический массив объектов данного класса и

// определить функции для работы с созданным массивом для **k** чисел. **Вариант 2.**

#include <iostream>

#include <iomanip>

using namespace std;

class **Cifra**{

int n;

int sum;

**public:**

Cifra(); // конструктор по умолчанию

void summa(); // функция получения суммы цифр числа

void show(); // функция вывода

};

Cifra::Cifra(){ // используется конструктор

cout<<"Vvedi chislo: "; cin>>n; // для инициализации массива

}

void Cifra::summa(){ // определение функции summa()

sum = 0;

int rab = n;

while(rab){

sum += rab % 10; rab /= 10;

}

}

void Cifra::show(){

cout<<"n = "<<setw(7)<<n<<setw(10)<<" summa cifr = "<<sum<<endl;

}

int **main(){**

int k;

cout<<"Vvedi kolichestvo chisel: "; cin>>k;

Cifra \* p = new Cifra[k]; // создание динам. массива объектов.

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

p[i].summa();

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

p[i].show();

}

**Результат:**

Vvedi kolichestvo chisel**: 2**

Vvedi chislo: 12345

Vvedi chislo: 5432

n = 12345 summa cifr = 15

n = 5432 summa cifr = 14

// **Пример 5**. Определить класс для нахождения суммы цифр натурального числа **n**.

// Сформировать динамический массив объектов данного класса и определить

// функции для работы с массивом для чисел из диапазона от **m1** до **m2**. **Вариант 3.**

#include <iostream>

#include <iomanip>

using namespace std;

class **Cifra**{ // конструктора в классе нет

int n;

int sum;

**public:**

void set(int a){n = a;}

void summa(); // функция получения суммы цифр числа

void show(); // функция вывода

};

void Cifra::summa(){ // определение функции summa()

sum = 0;

int rab = n;

while(rab){

sum += rab % 10; rab /= 10;

}

}

void Cifra::show(){

cout<<"n = "<<setw(5)<<n<<setw(10)<<" summa cifr = "<<sum<<endl;

}

int **main(){**

int m1, m2, k;

cout<<"vvedi diapazon chisel m1, m2: "; cin>>m1>>m2;

k = m2 - m1 + 1;

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

Cifra \* p = new Cifra[k]; // создание динам. массива объектов.

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

p[i].set(m1++);

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

p[i].summa();

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

p[i].show();

}

**Результат:**

vvedi diapazon chisel m1, m2: **123 126**

k = 4

n = 123 summa cifr = 6

n = 124 summa cifr = 7

n = 125 summa cifr = 8

n = 126 summa cifr = 9

**Указатели и объекты. *Указатель на объект*** – это *переменная*, которая хранит адрес объекта в памяти компьютера. Для объявления указателя на объект используется тот же синтаксис, что и в случае объявления *указателя на переменную* любого типа:

**Samp** \* p; // объявление указателя на тип **Samp**

Помимо объявления, указатели требуют ***инициализации***, так как в противном случае они содержат *непредсказуемые* значения.

Доступ к отдельному **public**-полю объекта можно получить, используя *операцию* *точки* **(.),** связывающую поле с именем объекта **– прямой доступ,** или через указатель на этот объект, используя *операцию**стрелки* **(–>),** т.е. **косвенный доступ**.

// **Пример 6**. Использование указателя на объект.

#include <iostream>

using namespace std;

class **Samp**{

int x; // **private**- переменная класса

**public**:

int y; // **public**-переменная класса

void setx(int n){x = n;}

void show();

};

void Samp::show(){

cout<<" x = "<<x<<" y = "<<y<<endl;

}

int **main()**{

Samp ob, \*p; // создание объекта **ob** и указателя **p**

ob.setx(1); // вызов функции **setx()**

ob.y = 1;

ob.show(); // x = 1 y = 1

p = &ob; // инициализация указателя

p->setx(5); // для доступа используется

p->y = 5; // операция **(->)**

p->show(); // x = 5 y = 5

(\*p).y = 55; // ***разыменование*** указателя, для доступа

// используется операция **(.)**

p->show(); // x = 5 y = 55

}

Выражение **(\*p)** разыменовывает указатель, разрешая доступ к содержимому памяти по адресу, хранящемуся в **p**.

**Указатели и массивы объектов.** После создания массива объектов некоторого класса указателю может быть присвоен адрес массива, при этом тип указателя должен совпадать с типом массива.

**// Пример 7.** Использование указателя на массив объектов.

#include <iostream>

using namespace std;

class **Samp{**

int x;

**public:**

Samp(int n){x = n;} // конструктор

int getx(){return x;} // возвращает **x**

**};**

int **main(){**

Samp ob[4] = { 1, 2, 3, 4 }; // вызов конструктора **4** раза

Samp \* p = ob; // **p** присваиваем адрес массива **ob**

for(int i = 0; i < 4; i++) // чтение массива объектов **ob**

cout<< p[i].getx()<<' '; // **1 2 3 4**

cout<<endl;

for(int i = 0; i < 4; i++)

cout<<ob[i].getx()<<' '; // **1 2 3 4**

cout<<endl;

}

**Адресная арифметика с объектами**. Под *адресной арифметикой* (*арифметика указателей*) понимают действия над указателями, связанные с использованием *адресов памяти объектов, массивов объектов*. Арифметика указателей с объектами выполняется ***относительно объекта***.

**// Пример 8.** Операции с указателями на объекты и массивы объектов.

#include <iostream>

using namespace std;

class **Samp{**

int x;

**public:**

Samp(){}; // конструктор по умолчанию

Samp(int n){x = n;} // конструктор с параметром

void set(int n){x = n;} // функция установки значения

void show(){ // функция вывода

cout<<" x = "<<x;

}

**};**

int **main(){**

Samp ob(5); // создание объекта, **ob**

Samp \*p, \*q; // **p, q** – указатели

p = &ob; // инициализация **p**

q = p; // **q** присваивается адрес,

// хранящийся в **p**

cout<<"\n p = "<<p<<" q = "<<q<<endl;

p->show(); q->show(); **//** **5** **5**

if(p == q) cout<<"\nYes"; else cout<<"\nNo";

Samp ob1(6); // создание объекта, **ob1**

Samp \* t = &ob1;

cout<<" \np = "<<p<<" t = "<<t;

if(p == t) cout<<"\nYes"; else cout<<"\nNo"<<endl;

Samp mob[4]; // массив объектов **mob**

**p = mob;** // **p** присваивается адрес массива **mob**

for(int i = 0; i < 4; i++){

p->set(i + 1); // **1 2 3 4**

p++;

}

**p = mob;** // **p** присваивается адрес массива **mob**

for(int i = 0; i < 4; i++)

p[i].show(); // **1 2 3 4**

}

**Результат:**

p = 004FFD10 q = 004FFD10

x = 5 x = 5

**Yes**

p = 004FFD10 t = 004FFCEC

**No**

x = 1 x = 2 x = 3 x = 4

В примере **8** вместо **p[i].show()** можно использовать следующие записи:

(p+i)->show();или (\*(p+i)).show();

**Указатель this**. Функции каждого объекта класса имеют доступ к некому указателю **this,** который*ссылается на сам объект***.** Указатель**thi*s*** *неявно передаётся каждой функции при её вызове*. Таким образом, любая функция может узнать адрес своего ***родного*** объекта.

**// Пример 9.** Указатель **this.**

#include <iostream>

using namespace std;

class **Adres**{

char str[10]; // массив из **10** байтов

int y;

**public:**

void showAdres(){cout<<"adres object = "<<**this**<<endl;}

};

int **main()**{

Adres ob1, ob2;

ob1. showAdres(); // вывод адреса объекта ob1

ob2. showAdres(); // вывод адреса объекта ob2

}

В функции **main()** создаются 2 объекта типа **Adres**. Затем выводятся их адреса с помощью функции **showAdres()**, которая просто выводит значение указателя **this**. При этом следует отметить, что **\*this** представляет сам объект.

***Доступ к полям объекта через указатель*** **this**. Когда происходит вызов функции, значением указателя **this** становится адрес объекта, для которого эта функция вызвана.

Указатель **this** можно интерпретировать, как любой другой указатель на объект, соответственно, его можно использовать для получения доступа к *полям-данным* объекта, на который он ссылается, причём используется *операция* *стрелки* **(->).**

**// Пример 10.** Использование указателя **this** для ссылки на поля объекта.

#include <iostream>

using namespace std;

class **Point**{

int x;

int y;

**public:**

Point(int x, int y){**this**->x = x; **this**->y = y;}

void show(){cout<<"Coords: x = "<< this->x<<"\t y = "<<this->y <<endl;

cout<<"Coords: x = "<<x<<"\t y = "<<y <<endl;}

};

int **main()**{

Point p1(20, 50);

p1.show();

}

**Результат:**

Coords: x = 20 y = 50

Coords: x = 20 y = 50

**Ссылка на объект. Э**то ещё одно имя объекта. Если объявлен объект **ob** класса **Samp:**

Samp **ob**; то можно определить *ссылку* на объект как

Samp **& ref** = ob;

и тогда **ob** и **ref** обозначают один и тот же объект. Это означает, что если, **public**-*переменной* **x**объекта **ob** присвоить, например, значение **5**, выполнив оператор **ob.x = 5;** то и **ref.x** будет равно **5**. Фактически, *ссылка – это как бы адрес объекта* (поэтому при определении ссылки используется символ **&** – знак *операции адреса*), и в этом смысле она сходна с указателем на объект, однако у ссылок есть свои особенности.

**Динамические объекты.** Для создания ***динамического объекта*** (как и для других типов данных) используют *операцию* **new** и указатель. Например:

Samp \* p; p = **new** Samp;

**или**

Samp \* p = **new** Samp;

delete p;

При создании динамических объектов возможна их **инициализация**, если в классе есть конструктор с параметрами:

Samp \* p = **new** Samp(5, 6); // **x = 5 y = 6**

Объекты, созданные с помощью *операции* **new**, называются ***динамическими объектами***, потому что они создаются и уничтожаются в процессе работы программы. Для доступа к членам объекта применяется *операция стрелки* (**->)** или *операция разыменования* указателя **(\*)**.

**// Пример 11.** Создание динамического объекта. Доступ к полям объекта.

#include <iostream>

using namespace std;

class **Samp**{

int x; // **private**- переменная **х**

**public**:

int y; // **public-** переменная **y**

void setx(int n){x = n;} // функция установки **setx()**

void show(){ // функция вывода **show()**

cout<<" x = "<<x<<" y = "<<y<<endl;

}

};

int **main()**{

Samp \*p = new Samp; // создание динамического. Объекта

p->y = 7;

p->setx(1); // вызов функции **setx()**

p->show(); // **x = 1 y = 7**

(\*p).y = 9;

(\*p).setx(5); // используется операция ***разыменования***

(\*p).show(); // **x = 5 y = 9**

delete **p;**

**}**

**Динамические массивы объектов.** В некоторых случаях в момент написания программы нам неизвестно количество объектов, которое необходимо создать. Тогда мы можем создать массив во время выполнения программы, используя *операцию* **new** и указатель:

Samp\* p = **new** Samp[размер массива];

delete [] p;

**// Пример 12.** Создание одномерного динамического массива объектов.

#include <iostream>

using namespace std;

class **Samp{ //** класс **Samp**

int x;

**public:**

Samp(){x = 0;} // конструктор по умолчанию

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

int get(){return x;} // возврат **x**

**};**

int **main(){**

int size; // size – размер динам. массива

cout<<"Vvedi size: "; cin>>size; // **4**

Samp \* p = new Samp[size]; // динам. массив объектов

for(int i = 0; i < size; i++){

cout<<p[i].get()<<' '; **// 0 0 0 0**

}

cout<<endl;

for(int i = 0; i < size; i++){

p[i].set(i);

// (p+i)->set(i);

// \*(p+i).set(i);

cout<<p[i].get()<<' '; **// 0 1 2 3**

}

cout<<endl;

cout<<"\*\*\*\*\*\*"<<endl; // \*\*\*\*\*\*

delete [] p;

}