Указатели ссылки и адреса

Строки в стиле C и C++

Двумерные статические массивы

Определения типов

Структуры

Лекция #4

Пустовалова О.Г.
доцент. каф. мат.мод.
ИММиКН ЮФУ

Содержание

Указатели и ссылки в C++

Указатели предназначены для хранения **адресов** областей памяти,

Ссылки представляют собой **синоним** имени, указанного при инициализации ссылки.

Ссылки в C++

Объявляя ссылку мы задаём альтернативный идентификатор (псевдоним) для уже созданного объекта.

```
int  v = 10; // Объявление переменной v
int &r = v; // Создание ссылки на переменную "v"

// Вывод на экран адреса переменой v
cout << &v << endl;
// Ссылка "r" хранит в себе этот же адрес
cout << &r << endl;
```

Результат

00EEF898
00EEF898

При выводе адресов переменных используется оператор **&** взятия адреса

Ссылки в C++

- Ссылки нельзя объявлять без привязки к переменной (то есть не инициализировав при объявлении)

```
int v = 12;
int &r;
```

Нельзя объявлять и не инициализировать!

для ссылки переменная "r" требуется инициализатор

Нужно объявлять и ИНИЦИАЛИЗИРОВАТЬ!

```
int v = 12;
int &r=v;
```

Ссылки в C++

при обращении к ссылке происходит обращение по тому адресу, на который она указывает

```
int v = 12;
int &r = v;

v = 1024;

// v = 1024  r = 1024
cout << "  v = " << v << "  r = " << r;
```

Ссылки в C++. Пример

```
void swap(double& a, double& b) {

        double temp = a;
        a = b;
        b = temp;
}

void main() {

        int x = 5, y = 3;
        swap(x, y);
        cout << x << "   " << y << endl;
}
```

Ссылки на функции в C++

```
void print(int s) { cout << s << endl; }

int sum(int a, int b) { return a+b; }

void main()
{
	void(&r1)(int) = print;
	print(123);	//  123
	r1(321);	//  321

	// ссылка на функцию
	int(&r2)(int,int) = sum;
	cout << r2(2,3) << endl; //  5

}
```

Указатели в C++

```
int i = 10;
```

Выделение памяти
Инициализация
значением 10

Все обращения в программе к переменной по ее имени (i) заменяются компилятором на адрес области памяти, в которой хранится значение переменной

Указатели в C++

- Программист может определить **собственные** переменные для хранения адресов областей памяти.

- Такие переменные называются указателями

- **указатели** предназначены для **хранения адресов** областей памяти

Указатели в C++

В C++ различают три вида указателей:

- указатели на **объект**
- указатели на **функцию**
- указатели на **void**

Они отличаются свойствами и набором допустимых операций.

Указатель не является самостоятельным типом, он всегда связан с каким-либо другим конкретным типом.

Указатели в C++. Указатель на объект

Указатель на объект содержит адрес области памяти, в которой хранятся данные определенного типа

```
тип *имя;
```

```
int *a;
```

Звездочка относится непосредственно к имени, поэтому для того, чтобы объявить несколько указателей, требуется ставить ее перед именем каждого из них

```
// два указателя на целое с именами a и c
// и переменная b целого типа
int *a, b, *c;
```

Указатели в C++. Указатель на объект

Указатель на объект содержит адрес области памяти, в которой хранятся данные определенного типа

```
int *a;

int* a;

int * a;
```

Можно использовать три варианта для описания указателей

Указатели в C++. Указатель на объект

```
// целочисленная переменная
int x;

// указатель на целочисленную переменную
int *p;
```

Чтобы получить адрес переменной, нужно перед ее именем написать амперсанд

```
p = &x;
```

Данная конструкция будет выполняться справа налево. Сначала с помощью оператора &, примененного к переменной x, будет получен адрес x. Затем адрес x будет сохранен в указателе p.

Есть и обратная операция. Чтобы получить значение переменной по ее адресу, следует написать звездочку перед именем указателя.

```
int y = *p;
```

Такая операция называется не «разыменование», или dereference.

Указатели в C++. Операции с указателями

- **Операция разадресации**, или разыменования, предназначена для доступа к величине, адрес которой хранится в указателе. Эту операцию можно использовать как для получения, так и для изменения значения величины (если она не объявлена как константа):

```
int x=100; //объявление переменной целого типа
int *p; //объявление указателя на переменную целого типа

p = &x; //присвоить p адрес переменной x
cout << *p << endl;  // 100
cout <<  p << endl;  // адрес
```

Указатели в C++. Операции с указателями

```
int x=100; //объявление переменной целого типа
int *p; //объявление указателя на переменную целого типа

p = &x; //присвоить p адрес переменной x
// изменили значение *p и x
*p = 17;

// значения
cout << x << endl;// 17
cout << *p << endl;// 17

// адреса &x==p
cout << &x << endl; // 0059FDEC
cout << p << endl;  // 0059FDEC
```

Указатели в C++. Адресная арифметика

К указателям можно прибавлять числа. Из указателей можно вычитать числа. На основе этого сделана адресация в массиве.

```
int array[5] = { 100, 20, 3, 4, 5 };
// указатель на первый элемент
int *p = &array[0];

p++; // + 1 * sizeof(int)
cout << *p << endl; // 20

p--; // - 1 * sizeof(int)
cout << *p << endl; // 100
```

Указатели в C++. Адресная арифметика. Пример

К указателям можно прибавлять числа. Из указателей можно вычитать числа. На основе этого сделана адресация в массиве.

```
int array[5] = { 100, 20, 3, 4, 5 };

int *p = &array[0]; // указатель на первый элемент

for (int i = 0; i < size(array); i++)
        cout << *(p++) << " "; // 100 20 3 4 5

cout<<endl;
```

Указатели в C++. Адресная арифметика. Пример

К указателям можно прибавлять числа. Из указателей можно вычитать числа. На основе этого сделана адресация в массиве.

```
int array[5] = { 100, 20, 3, 4, 5 };

int *begin = &array[0]; // указатель на первый элемент
int *end=&array[4];  // указатель на последний элемент

for (int *p=begin; p<=end; ++p)
      cout << *p << " "; // 100 20 3 4 5
```

Указатели в C++. Адресная арифметика. Пример

К указателям можно прибавлять числа. Из указателей можно вычитать числа. На основе этого сделана адресация в массиве.

```
int array[5] = { 100, 20, 3, 4, 5 };

// имя массива - указатель на первый элемент
int *p = array;

for (int i = 0; i < size(array); i++)
      cout << *(p++) << " "; // 100 20 3 4 5
```

Указатели в C++. Адресная арифметика. Пример

К указателям можно прибавлять числа. Из указателей можно вычитать числа. На основе этого сделана адресация в массиве.

```
int array[5] = { 100, 20, 3, 4, 5 };

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

        // имя массива - указатель на первый элемент
        cout << *(array + i) << " "; // 100 20 3 4 5
```

Указатели в C++. Адресная арифметика. Пример

К указателям можно прибавлять числа. Из указателей можно вычитать числа. На основе этого сделана адресация в массиве.

```
int array[5] = { 100, 20, 3, 4, 5 };

// имя массива - указатель на первый элемент
int *p = array;

for (int i = 0; i < size(array); i++)
      // 100 20 3 4 5
      cout << i[array] << " ";
```

Указатели в C++. Адресная арифметика

- С указателем можно складывать число, представленное переменной или целочисленной константой

```
int array[5] = { 100, 20, 3, 4, 5 };
// имя массива - указатель на первый элемент
int *p = array;

const int n = 2;
cout << *(p + n) << endl; // 3

int k = 1;
cout << *(p + k) << endl; // 20
```

Указатели в C++. Указатель на void. Разыменование

Указатель на void применяется в тех случаях, когда конкретный тип объекта, адрес которого требуется хранить, не определен

```
int a=5;
float q = 3.0;
string s = "qwerty";

void *p;
p = &a; // допустимо
p = &q;
p = &s;

// Выражение должно представлять собой
//  указатель на полный тип объекта
cout <<*p ;// ОШИБКА!!!
```

Нужно преобразовать тип, чтобы не было ошибки

Указатели в C++. Указатель на void. Разыменование

Указатель на void нужен, если в одной и той же переменной в разные моменты времени требуется хранить адреса объектов различных типов.

```
int a=5;
float q = 3.5;

void *p;

// адрес переменной целого типа
p = &a;
// разыменование с преобразованием типов
cout << *(static_cast<int*>(p))<<endl;

// адрес переменной вещественного типа
p = &q;
// разыменование с преобразованием типов
cout << *(static_cast<float*>(p))<< endl;
```

Нужно преобразовать тип, чтобы не было ошибки

Указатели в C++. Примеры указателей

```
int i; // целая переменная
const int ci = 1; // целая константа

// указатель на целую переменную
int * pi;

// указатель на целую константу
const int * pci;
```

Указатели в C++. Примеры указателей

```
// указатель-константа на целую переменную
int * const cp = &i;

// указатель-константа на целую константу
const int * const cpc = &ci;
```

Указатели в C++. Примеры указателей

модификатор const, находящийся между именем указателя и звездочкой, относится к самому **указателю и запрещает его изменение**

```
// указатель-константа на целую переменную
int * const cp = &i;
```

const слева от звездочки задает **постоянство значения**, на которое он указывает

```
// указатель на целую константу
const int * pci;
```

Указатели в C++. Константный указатель

```
int i = 10;
int *const cp = &i;
cout << *(cp++) << endl;// ОШИБКА!!!
```

int *const cp

выражение должно быть допустимым для изменения левосторонним значением

Указатели в C++. Константный указатель

```
int i = 10;
int *const cp = &i;
// константный указатель
cout << *(cp++) << endl;// ОШИБКА!!!
```

int *const cp

выражение должно быть допустимым для изменения левосторонним значением

```
int i = 10;
int *const cp = &i;
// поменяли  значение переменной
cout <<++(*cp) << endl; // 11
```

```
int i = 10;
int *const cp = &i;
// поменяли  значение переменной
i++;
cout << *cp << endl;// 11
```

Указатели в C++. Указатель на целую константу

```
const int i = 10;
const int * pci;
// указатель на целую константу
cout <<++(*pci) << endl;          // ОШИБКА!!!
```

выражение должно быть допустимым для изменения левосторонним значением

```
const int i = 10;
const int * pci=&i;
cout << *(pci) << endl;  // 10
cout <<   pci << endl;  //  адрес 1

// поменяли адрес и уже будет другое значение
cout <<*(++pci) << endl; //  что-то по новому адресу
cout << ++pci   << endl; //  адрес 2
```

Указатели в C++. Способы инициализации указателей

Присваивание указателю адреса существующего объекта

```
// с помощью операции получения адреса :
int a = 5; // целая переменная
int* p = &a; //в указатель записывается адрес a
int* p(&a); // то же самое другим способом
```

Указатели в C++. Способы инициализации указателей

Присваивание указателю адреса существующего объекта

```
int a = 5; // целая переменная

int* p = &a; //в указатель записывается адрес a


// с помощью значения другого
// инициализированного указателя :

int* r = p;
```

Указатели в C++. Способы инициализации указателей

Присваивание указателю адреса существующего объекта

```
// с помощью имени массива или функции,
// которые трактуются как адрес :

// массив
int b[10];

// присваивание адреса начала массива
int* t = b;
```

Указатели в C++. Способы инициализации указателей

Присваивание пустого значения

```
int* p1 = NULL;    // C-style

int* p2 = 0;       // C++98

int* p3 = nullptr;// C++11
```

Поскольку гарантируется, что объектов с нулевым адресом нет, пустой указатель можно использовать для проверки, ссылается указатель на конкретный объект или нет.

```
int* p0 = nullptr;

int x = 5;
int *p = &x;

if (p != p0) cout << "Yes!";
```

Указатели в C++. Пример. swap

```
void swap(double* a, double* b) {

double temp = *a; *a = *b; *b = temp;
}

void main()
{
        int x = 5, y = 3;
        swap(x, y);
        cout << x << "  " << y << endl;

        int a = 5, b = 3;
        int *p1 = &a;
        int *p2 = &b;
        swap(p1, p2);
        cout << *p1 << "  " << *p2 << endl;
}
```

Указатели в C++. Пример. Передача массива в функцию

Передача указателей на начало и на конец массива

Можно использовать встроенные библиотечные функции **std::begin()** и **std::end()**:.

Причем std::end возвращает указатель не на последний элемент, а **адрес за последним элементом в массиве**.

```
int a[] = { 1, 2, 3, 4, 5 };


int *begin = std::begin(a); // указатель на начало массива
int *end = std::end(a);     // указатель на конец массива
```

Указатели в C++. Пример. Передача массива в функцию

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

void printArr(int *begin, int *end)
{
	for (int *ptr = begin; ptr != end; ptr++)
	cout << *ptr << " ";
}

void main()
{
	int a[] = { 1, 2, 3, 4, 5 };
	int *begin = std::begin(a);
	int *end = std::end(a);

	printArr(begin, end);
}
```

Указатели в C++. Пример. Передача массива в функцию

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

int summArr(int *begin, int *end)
{
	int s = 0;
	for (int *ptr = begin; ptr != end; ptr++)
	s += *(ptr);
return s;
}

void main()
{	int a[] = { 1, 2, 3, 4, 500 };
	int *begin = std::begin(a);
	int *end = std::end(a);

	cout << summArr(begin, end) << endl;
}
```

Указатели в C++. Пример. Передача массива в функцию

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

void mult2Arr(int *begin, int *end)
{
	for (int *ptr = begin; ptr != end; ptr++)
	*ptr = *(ptr) * 2;
}
void main()
{	int a[] = { 1, 2, 3, 4, 5 };
	int *begin = std::begin(a);
	int *end = std::end(a);

	mult2Arr(begin, end);

	for (int *ptr = begin; ptr != end; ptr++)
		cout << *ptr << " ";
}
```

Указатели в C++. Пример. Указатели на функцию

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

double area(double R) {
	const double PI = 3.1415;  return PI * R * R;
}

void main() {
	double r = 1.0;
	// переменная указатель на функцию
	double(*pfunc)(double);

	pfunc = &area;
// упоминание имени функции в операторе присвоения интерпретирует
// как её адрес
	// pfunc = area;

	cout<< (*pfunc)(r) << endl;
}
```

Указатели в C++. Пример. Указатели на функцию

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

void test1(void) { cout << "test 1" << endl; // ... }

void test2(void) { cout << "test 2" << endl; // ... }

void test3(void) { cout << "test 3" << endl; // ... }

// массив указателей на функции
void(*tests[])(void) = { test1, test2, test3 };

void main()
{
        for (int i = 0; i < size(tests); i++)
                tests[i]();
}
```

```
test 1
test 2
test 3
```

Строки в стиле С

Строки в стиле С

С-строки это массивы символов

```
char str[] = "Cogito ergo sum";
```

Длина строки = количество символов в строке + символ конца строки

Символ конца строки: \0 - нуль терминатор или нулевой байт: байт, все биты которого равны 0

Чтобы объявить строку, состоящую из 12 букв, необходимо зарезервировать дополнительную ячейку под нулевой символ

```
char myString[13] = "A posteriori";
```

Строки в стиле С. Ввод и вывод строк

```
#include <iostream>
//using namespace std;
int main() {
        char myString[20];
        std::cin.getline(myString,20);
        std::printf(myString);
return 0;
}
```

Строки в стиле С. Функция printf

```
// два десятичных целых числа
printf("%d %d \n", 3, 56);

// два десятичных числа с плавающей точкой
printf("%3.2f %3.2f \n", 3.14, 2.71);

// строка
printf("%s \n", "a priori");

// строка, символ, число с точкой, целое число
printf("%s %4.2f %d \n", "a priori",  3.14, 17);
```

- Стандартная функция консольного вывода в языке С – **printf**.
- Описание её содержится в заголовочном файле **stdio.h**.
- При помощи этой функции можно выводить данные или сообщения в консоль.

Строки в стиле C. Функция printf

%d	Десятичное число целого типа со знаком
%i	Десятичное число целого типа со знаком
%e	Научная нотация (e нижнего регистра)
%E	Научная нотация (E верхнего регистра)
%f	Десятичное число с плавающей точкой
%g	Использует код %e или %f — тот из них, который короче (при использовании %g используется e нижнего регистра)
%G	Использует код %E или %f — тот из них, который короче (при использовании %G используется E верхнего регистра)
%o	Восьмеричное целое число без знака
%s	Строка символов
%u	Десятичное число целого типа без знака
%x	Шестнадцатеричное целое число без знака (буквы нижнего регистра)
%X	Шестнадцатеричное целое число без знака (буквы верхнего регистра)
%p	Выводит на экран значение указателя
%%	Выводит символ %

Строки в стиле C. Функция printf

```
// Выводит на экран значение указателя
int k = 123;
int &q=k;
printf("%p \n", q);

// Научная нотация (E верхнего регистра)
printf("%4.2E \n", 123.4);

// Научная нотация (e нижнего регистра)
printf("%4.2e \n", 123.4);

// Шестнадцатеричное целое число без знака
printf("%X \n", 16);
```

Строки в стиле С. Функция getline

Функция **getline** предназначена для ввода данных из потока.

getline извлекает данные из входного потока до строкового разделителя, который не записывается в получившийся массив данных.

```
cin.getline(string, streamsize, separator);
```

- string – переменная типа char*,
- streamsize - максимально количество символов
- separator – строковый разделитель, показывающий на конец строки. Последний параметр функции можно опустить, тогда будет задан сепаратор по умолчанию - '\n'.

```
char str[256];
cin.getline(str, 256, ';');
```

Строки в стиле С. Передача строки в функцию

```
#include <iostream>
using namespace std;
// передаем строку, как массив
void print1(char str[]){ cout << str << endl; }

// передаем через указатель *str
void print2(char *str) { cout << str << endl; }

// передаем через адрес. строка из 80-ти символов
void print3(char(&str)[80]) { cout << str << endl;}

void main() {
char s1[] = "void print1 (char str[]);\n";    print1(s1);

char s2[] = "void print2 (char *str); \n";    print2(s2);

char s3[80] ="void print3 (char &str[]);\n";  print3(s3);
}
```

Строки в стиле С. Передача строки в функцию

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

// const - нельзя менять
void print(const char str[]){ cout << str << endl; }

// можно менять
void change(char str[]) { str[0] = '\n'; }


void main() {
	char s[] = " String example\n";
	change(s);
	print(s);
}
```

Строки в стиле C++

- В современном стандарте C++ определен класс с функциями и свойствами (переменными) для организации работы со строками

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

- http://www.cplusplus.com/reference/string/string/
- https://ru.wikipedia.org/wiki/String_(C%2B%2B)

операторы и функции класса string

Строки в стиле C++. Функция getline

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

int main() {
	string str;
	cout << "Enter your name: ";

	getline(cin, str);

	cout << "Hello, " << str << "!!! \n";

return 0;
}
```

Строки в стиле C++. Пустая строка

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

int main() {
	setlocale(0, "");
	// пустая строка
	string myString;

// длина строки
cout << " Длина строки = " << myString.length() <<endl; // 0
cout << " Длина строки = " << myString.size() << endl;  // 0
cout << " Длина строки = " << size(myString) << endl;   // 0

return 0;
}
```

Строки в стиле C++. Длина строки

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

int main() {
	string s = "a priori";

	// В классе string определены методы size и length
	// возвращают количество
	// символов в строке без знака конца строки
	cout<<s.size()<<endl;		// 8
	cout << s.length() << endl; // 8

	// Индексация символов с нуля
	cout<<s[0]<<endl;
return 0;
}
```

Строки в стиле C++. Проверка на пустоту

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


int main() {
	setlocale(0, "");
	string s = "a priori";

	if (s.size()!=0) cout << s.size() << endl;
	if (s.length()!=0) cout << "Yes!" << endl;
	if (!s.empty()) cout << s << endl;

return 0;
}
```

Строки в стиле C++. Инициализация строки другой строкой

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

int main() {
        string s1 = "a priori";

      // Инициализация строки другой строкой
       string s2(s1);
       cout << s2 << endl;

       // Сравнение строк
       cout << (s1!=s2) << endl;

return 0;
}
```

Строки в стиле C++. Конкатенация строк

```
string s1 = "ABC";

// Копирование строки
string s2 = s1;

// Конкатенация строк
s1+=s2;
cout << s1<<endl;   // ABCABC

string s3 = s1 + s2;
cout << s3 << endl; // ABCABCABC
```

Строки в стиле C++. Конкатенация строк

```
// C-style
const char *p = ", ";

// C++ style
string s1("Hello");
string s2("World!");

// Конкатенация. Новая=старая + новые
string s3 = s1 + p + s2 + "\n";

cout << endl << s3; // Hello, World!
```

Строки в стиле C++. Присваивание старой строки новой

```
// Строка C++
string s;

// Строка C
const char *p = "C-style";

// Новая=Старая - работает
s = p;

cout << s << endl;
```

Строки в стиле C++. Присваивание новой строки старой

```
// Строка C++
string s="ABC";

char *str = s;//ОШИБКА!
```

std::string s
Строка C++

не существует подходящей функции преобразования из "std::string" в "char *"

Строки в стиле C++. Присваивание новой строки старой

```
string s="ABC";

// Чтобы осуществить такое преобразование,
// необходимо явно вызвать функцию-член с
// названием c_str() - "строка Си "

const char *str = s.c_str();

cout << str << endl; // ABC
```

Строки в стиле C++. Старый и новый подходы

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

void main() {
	string s("A*B*C*");
	int n = s.size();

	for (int i = 0; i <= n; i++)
		if (s[i] == '*') s[i] = ':';

cout << s;// A:B:C:
}
```

Строки в стиле C++. Старый и новый подходы

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

int main() {
	string s("A*B*C");

replace(s.begin(), s.end(), '*', ':');

	cout << s;// A:B:C

return 0;
}
```

Скорость работы **C-строк** немного выше чем у класса **string**, но на прикладном уровне удобнее использовать **string**

Строки в стиле C++. Передача строки в функцию

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

// const - нельзя менять
void print(const string &str){ cout << str << endl; }

// можно менять
void change(string &str) { str = str + str; }


void main() {
	string s = "String example:) ";
	change(s);
	print(s);
}
```

Двумерные статические массивы

Двумерные статические массивы

Двумерный массив — это одномерный **массив** одномерных **массивов**

```
// объявление двумерного массива:
int a[3][4];

// объявление и  инициализация двумерного массива:
int b[2][3] = { { 1, 2, 3 },{ 4, 5, 6 } };
```

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[1][2]

Двумерные статические массивы

```
#include <iostream>
#include <iomanip>
using namespace std;
void main() {
// объявление и  инициализация двумерного массива:
int a[2][3] = { { 1, 2, 3 },{ 11, 22, 33 } };

// строки
for (int i = 0; i < 2; i++)
{
        // столбцы
        for (int j = 0; j < 3; j++)// переключение по столбцам
        cout << setw(4)<<right<<a[i][j];

cout << endl;
}
}
```

Двумерные статические массивы. Расположение в памяти

Для двумерного С-массива выделяется единый блок памяти необходимого размера:

```
размер_массива1 * размер_массива2 * sizeof(тип_элемента_массива))
```

- Значения располагаются последовательно.
- Самый левый индекс изменяется медленнее всего.
- Имя (идентификатор) двумерного С-массива является указателем на первый элемент массива

Двумерные статические массивы. Примеры

Работа с указателями

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

void main() {

int arr[2][3] = { {1, 2, 3},{4, 5, 6} };

        int *ptr = (int *)arr;

        for (int i = 0; i < 2; i++) {
                for (int j = 0; j < 3; j++) {
                        cout << setw(3) << *(ptr + i * 3 + j);
                }
                cout << endl;
        }

}
```

```
// Результат
// 1 2 3
// 4 5 6
```

Двумерные статические массивы. Примеры

Работа с указателями

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

void main() {

int arr[2][3] = { { 1, 2, 3 },{ 4, 5, 6 } };

        int *ptr = (int *)arr;

                for (int i = 0; i < 2 * 3; i++) {
                cout << setw(4) << ptr[i];
        }

        cout << endl;
}
```

```
// Результат
// 1 2 3 4 5 6
```

Двумерные статические массивы. Передача в функцию

```
const unsigned int D1 = 3;
const unsigned int D2 = 5;

void func(int arr[D1][D2]) { ... }    // (1)

//  При передаче многомерного С-массива в функцию можно
//  не указывать длину самого левого измерения.

void func(int arr[][D2]) { ... }      // (2)
```

Двумерные статические массивы. Передача в функцию

Пример 1

```
#include <iostream>
#include <iomanip>
using namespace std;
const unsigned int D1 = 2;
const unsigned int D2 = 3;

void printArr(int a[D1][D2]) {
	for (int i = 0; i < D1; i++) {
		for (int j = 0; j < D2; j++)
			cout << setw(4) << right << a[i][j];
		cout << endl;
}}

void main() {
		int a[D1][D2] = { { 1, 2, 3 },{ 11, 22, 33 } };
		printArr(a);
}
```

Двумерные статические массивы. Передача в функцию

Пример 2

```
#include <iostream>
#include <iomanip>
using namespace std;
const unsigned int D1 = 2;
const unsigned int D2 = 3;
// ... код для printArr

void changeArr(int a[][D2]) {
	for (int i = 0; i < D1; i++)
		for (int j = 0; j < D2; j++)
			a[i][j]*=10;
}
void main() {
	int a[][D2] = { { 1, 2, 3 },{ 11, 22, 33 } };
	changeArr(a);
	printArr(a);
}
```

```
//  Результат
//  10  20  30
// 111 220 330
```

Определение типов

Определение типов

Синонимы (псевдонимы)типов определяются с помощью **typedef**

```
typedef тип новое_имя[размерность];
```

```
typedef int arrayInt[3];      // arrayInt имя типа
typedef int matrixInt[2][3]; // matrixInt имя типа

// Использование matrixInt
// Объявление переменной matrixInt a будет заменено на int a[2][3]
void printArr(matrixInt a, int m, int n);
```

Определение типов

Пример. matrixInt

```
#include <iostream>
#include <iomanip>
using namespace std;
const unsigned int D1 = 2;
const unsigned int D2 = 3;

typedef int matrixInt[2][3]; // matrixInt имя типа

void printArr(matrixInt a, int D1, int D2) {
	for (int i = 0; i < D1; i++) {
		for (int j = 0; j < D2; j++)
			cout << setw(4) << right << a[i][j];
	cout << endl;}}

void main() {
		matrixInt a = { { 1, 2, 3 },{ 11, 22, 33 } };
		printArr(a, D1, D2);
}
```

Структуры

Структуры

Структура — это, некое объединение различных переменных (возможно с разными типами данных), которому можно присвоить имя.

```
struct имя_структуры
{
        компоненты структуры
};
```

```
struct book
{
        string title;
        string author;
        int count;
        float price;
};
```

Структуры

```
Name: Mike        Age: 21
Name: Tom         Age: 17
```

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

struct person {
                int age;
                string name; };

void main() {

        person p2,p1 = {21, "Mike"};
        cout << "Name: " << p1.name << "\t Age: " << p1.age << endl;

        p2.name = "Tom";
        p2.age = 17;
        cout << "Name: " << p2.name << "\t Age: " << p2.age << endl;
}
```

Спасибо за внимание!