C++

Перечисления

Одномерные массивы

Лекция #3

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

Содержание

Перечисления C++ 98

Перечисление — это пользовательский тип, состоящий из набора **целочисленных констант**, называемых перечислителями.

Стандартный вид перечислений следующий:

```
enum ярлык { список перечислений } список переменных;
```

```
enum seasons { spring, summer, autumn, winter };
seasons myFavorite = spring;
```

Перечисления C++ 98

```
enum seasons { spring, summer, autumn, winter };

seasons myFavorite = spring,
        yourFavorite=summer,
        theirFavorite=winter;

cout << myFavorite;//0
cout << yourFavorite;//1
cout << theirFavorite;//3
```

Перечисления C++ 98

```
enum seasons { spring=1, summer, autumn=5, winter };

seasons myFavorite = spring,
        yourFavorite=summer,
        theirFavorite=winter;

cout << myFavorite;//1
cout << yourFavorite;//2
cout << theirFavorite;//6
```

Перечисления C++ 98

```
enum seasons { spring=123, summer, autumn, winter };
seasons myFavorite = spring;

cout << myFavorite;//123

int n = myFavorite; // разрешено присваивание
cout << n;//123
```

Перечисления C++ 98

```
enum seasons { spring, summer, autumn, winter };
seasons myFavorite = 5; // ОШИБКА!
```

```
enum seasons { spring, summer, autumn, winter };
seasons myFavorite = 5;
```

значение типа "int" нельзя использовать для инициализации сущности типа "seasons"

Перечисления C++ 98

```
enum seasons { spring, summer, autumn, winter };
//seasons myFavorite = 5; // ОШИБКА!

// преобразование типов
seasons myFavorite =(seasons)1;

if (myFavorite == summer)
        cout << "Summer is the best seasons!";
```

Перечисления C++ 98

```
enum seasons { spring, summer, autumn, winter };
//seasons myFavorite = 5; // ОШИБКА!

//  преобразование типов
seasons myFavorite =static_cast<seasons>(1);

if (myFavorite == summer)
        cout << "Summer is the best seasons!";
```

Перечисления C++ 98. Примеры

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

int main()
{   // перечисление coin
enum coin { penny, nickel, dime, quarter, half_dollar, dollar };

cout << penny <<" "<< nickel<<" "<< dollar << endl;// 0 1 5
return 0;
}
```

Перечисления C++ 98. Примеры

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


int main()
{   // перечисление coin с инициализацией
        enum coin { penny=1, nickel=5, dime=10, quarter=25,
                        half_dollar=50, dollar=100 };


 cout << penny <<" "<< nickel<<" "<< dollar << endl;// 1 5 100
        return 0;
}
```

Перечисления C++ 98. Примеры

```
coin money = dollar;
// вывод символов перечислений
switch (money) {
case penny: printf("penny");
        break;
case nickel: printf("nickel");
        break;
case dime: printf("dime");
        break;
case quarter: printf("quarter");
        break;
case half_dollar: printf("half_dollar");
        break;
case dollar: printf("dollar");
}
```

Перечисления C++ 11

в C++11 был введен **enum class** — строго типизированные перечисления с ограниченной областью видимости для предотвращения ошибок, связанных с коллизией имен в крупных проектах.

Так как при объявлении **enum** все имена перечисления экспортируются во внешнюю область видимости

```
enum class seasons { spring, summer, autumn, winter };
```

Перечисления C++ 11. Особенности

```
enum class seasons { spring, summer, autumn, winter };

// преобразование типов
seasons my1 = static_cast<seasons>(1);//summer
seasons my2 = (seasons)(1);//summer

// указание области видимости
seasons my3 = seasons::summer;

if (my1 == seasons::summer) cout << "static_cast<seasons>(1)"<<endl;
if (my2 == seasons::summer) cout << "(seasons)(1)" << endl;
if (my3 == seasons::summer) cout << «seasons::summer" << endl;
```

Перечисления C++ 11. Особенности

```
enum class seasons { spring, summer, autumn, winter };

seasons myFavorite = spring;
```

идентификатор "spring" не определен

```
enum class seasons { spring, summer, autumn, winter };

seasons myFavorite = seasons::spring;
```

Перечисления C++ 11. Особенности

```
enum class seasons { spring, summer, autumn, winter };

int mySeason = seasons::spring;
```

enum class seasons::spring = 0

значение типа "seasons" нельзя использовать для инициализации сущности типа "int"

Перечисления C++ 11. Особенности

```
enum class seasons { spring, summer, autumn, winter };

// При необходимости присваивания переменной типа
seasons
// целого числа
// можно воспользоваться оператором
// static_cast<type>(object);

seasons mySeason = static_cast<seasons>(1);
```

Одномерные статические массивы

Одномерные статические массивы

Массив - это непрерывный участок памяти, содержащий последовательность объектов одинакового типа, обозначаемый одним именем.

Каждый элемент массива характеризуется тремя величинами:

- **адресом элемента** - адресом начальной ячейки памяти, в которой расположен этот элемент;
- **индексом элемента** (порядковым номером элемента в массиве);
- **значением элемента**.

Одномерные статические массивы

- **Адрес массива** – адрес начального элемента массива.
- **Имя массива** – идентификатор, используемый для обращения к элементам массива.
- **Размер массива** – количество элементов массива
- **Размер элемента** – количество байт, которое занимает один элемент массива

Одномерные статические массивы

Графически расположение массива в памяти компьютера можно представить в виде непрерывной ленты адресов.

Адрес	w	w+q	w+2*q		w+(n-1)*q
Значение	a[0]	a[1]	a[2]	…	a[n-1]
Индекс	0	1	2		n-1

w – адрес первого элемента

q=sizeof(a[0]) – размер в байтах одного элемента

n – количество элементов массива

n= sizeof(a)/ sizeof(a[0]) или n= size(a)

адрес k-го элемента массива a[0]+(k-1)*q

Одномерные статические массивы

```
// Массив a имеет четыре элемента, но они неопределены
        int a[4];
// Можно выполнить инициализацию и присвоить элементам массива
некоторые значения:
        int a[4] = { 1,2,3,4 };

// Индексация массивов начинается с нуля
        cout << "a[0] = " << a[0] << endl;
        cout << "size of a = " << (size(a)) << endl;
```

Одномерные статические массивы

```
int a[4] = { 1, 2 ,3, 4 };
        // Обратившись к элементу по индексу,
        // мы можем получить его значение,
        // либо изменить его
        cout << a[0] <<endl;
        a[0] = 1024;
        cout << a[0] << endl;
```

Одномерные статические массивы

```
// Значения в фигурных скобках называют инициализаторами.
// Если инициализаторов меньше, чем элементов в массиве,
// то инициализаторы используются для первых элементов.

  int arr[4] = { 1, 2 };
 for (int i = 0; i < 4; i++) { cout << arr[i] << " "; }


// Результат: 1 2 0 0
```

Одномерные статические массивы

```
// Если инициализаторов больше, чем элементов в массиве, то при
// компиляции возникнет ошибка:
        int a[4] = { 1,2,3,4,5,6 };
```

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

int [4]{(int)1, (int)2, (int)3, (int)4}

слишком много значений инициализатора

Одномерные статические массивы

```
// Если размер массива не указан явно, то он выводится из
// количества инициализаторов:

int numbers[] = { 1,2,3,4,5,6,7,8 };
cout <<"size of numbers = " <<(size(numbers)) << endl; // 8
```

Одномерные статические массивы

```
// При инициализации символьного массива можно его
// инициализировать
// как набором символов, так и строкой:

        char s1[] = { 'A', 'B', 'C' };
        char s2[] = "ABC";
```

Одномерные статические массивы

```
char s1[] = { 'A', 'B', 'C' };

char s2[] = "ABC";

 cout << "size of s1 = " << (size(s1)) << endl; //  3

// s2 содержит 4 элемента
// в конец массива добавляется нулевой символ - признак
// завершения строки

cout << "size of s2 = " << (size(s2)) << endl; //  4
```

Одномерные статические массивы

```
// не допускается присвоение одному массиву другого массива:
        int a1[] = { 1, 2 };
        int a2[] = { 11, 12 };
        a1 = a2;  // ОШИБКА!!!
```

```
a1 = a2;
```

int a1[2]
не допускается присвоение одному массиву другого массива:

выражение должно быть допустимым для изменения левосторонним значением

Одномерные статические массивы

```
// не допускается присвоение одному массиву другого массива:
        int a1[] = { 1, 2 };
        int a2[] = { 11, 12 };
        int a3[] = a1; // ОШИБКА!!!
```

для агрегатного объекта требуется инициализация с использованием "{...}"

Одномерные статические массивы

```
// Число элементов массива также можно определять через
// константу:
      const int n = 4;
      int a[n] = { 1,2,3,4 };
```

Одномерные статические массивы

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


int main()
{
        int n;
        cin >> n;
        int a[n]; /* ОШИБКА */
    return 0;
}
```

int n

выражение должно иметь константное значение
значение переменная "n" (объявлено в строке 9) невозможно использовать как константу

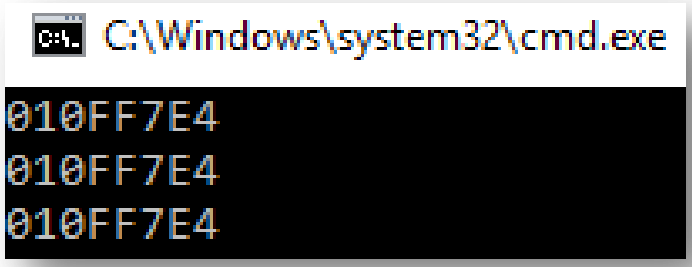
Одномерные статические массивы

```
// количество элементов массива
int a[] = { 0, 1, 2 ,3, 4, 5, 6, 7, 8, 9 };
cout << " Количество элементов массива " << size(a) << endl;

// размер массива
cout << " Размер массива " << sizeof(a) << endl;


// размер элемента массива
cout << " Размер элемента массива " << sizeof(a[0]) << endl;

// количество элементов массива
cout << " Количество элементов массива "
             << sizeof(a) / sizeof(a[0]) << endl;
```

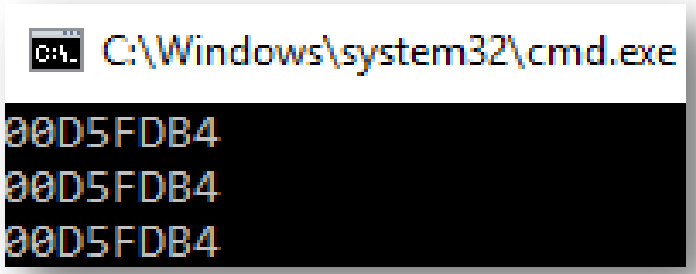
Одномерные статические массивы

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


int main()
{
    int a[3] = {123, 456, 789 };
        for (int i = 0; i < size(a); i++) {
                cout << a << endl;
        }
        return 0;
}
```

Результат 1-го запуска



Результат 2-го запуска



Одномерные статические массивы

```
int a[] = {0, 1, 2 ,3, 4, 5, 6, 7, 8, 9 };


// Вывод элементов массива. Вариант 1
        int n = size(a);// n=sizeof(a)/sizeof(a[0]);
        for (int i = 0; i < n; i++)
                cout << a[i] << " ";
        cout << endl;
```

Одномерные статические массивы

```
int a[] = {0, 1, 2 ,3, 4, 5, 6, 7, 8, 9 };


// Вывод элементов массива. Вариант 2
        // При переборе массива каждый перебираемый элемент
        // будет помещаться в переменную x,
        // значение которой в цикле выводится на консоль:
        for (int x : a)
                cout << x << " ";
        cout << endl;
```

Одномерные статические массивы

```
int a[] = {0, 1, 2 ,3, 4, 5, 6, 7, 8, 9 };


// Вывод элементов массива. Вариант 3
        // если неизвестен тип объектов в массиве,
        // то можно использовать спецификатор auto для
        //определения типа:
        for (auto x: a)
                cout << x << " ";
        cout << endl;
```

Одномерные статические массивы

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

int main()
{
	int a[] = { 1, 2, 3, 4, 5 };
	// x передается по ссылке, поэтому
	// доступ есть на чтение и запись
	for (int &x : a)
	{
		x += 1;
		cout << x << "";
	}

return 0;
}
```

Передача массива в функцию

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

void arrPow2(int a[], const int n) {
	for (int i = 0; i < n; i++) a[i] = a[i] * a[i];
}

int const n = 5;
void main() {

	int a[n] = { 1,2,3,4,5 };
	arrPow2(a, n);
	for (int i = 0; i < n; i++) { cout << " " << a[i]; }
	cout << endl;
}
```

Передача массива в функцию

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

int summArr(const int* a, const int n) {
        int s = 0;
        for (int i = 0; i < n; i++) s += a[i];
        return s;
}
int const n = 5;

int main() {
        int a[n] = { 1,2,3,4,5 };
        cout << " s = " << summArr(a, n) << endl;
return 0;
}
```

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
```

Передача массива в функцию

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

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

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

	printArr(begin, end);
return 0;
}
```

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;
}

int main()
{
	int a[] = { 1, 2, 3, 4, 500 };
	int *begin = std::begin(a);
	int *end = std::end(a);
	cout << summArr(begin, end) << endl;
return 0;
}
```

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;
}

int 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 << " ";
return 0;
}
```

Спасибо за внимание!