Сложные объявления в C++

Строки C и C++

Лекция #6

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

Содержание

Сложные объявления

Сложные объявления. Константный указатель

```
int a = 8; // два обычных объекта типа int
int b = 1024;

int *const p = &a; // Константный указатель на целое
*p = 123; // Менять значение разрешено

p=&b; // Но изменять адрес не разрешается
```

int *const p

Константный указатель

выражение должно быть допустимым для изменения левосторонним значением

Сложные объявления. Указатель на константу

```
int a = 8; // два обычных объекта типа int
int b = 1024;

const int *p = &a; //Указатель на константу
p = &b; //Менять адрес разрешено


*p=110; //Менять значение нельзя
```

выражение должно быть допустимым для изменения левосторонним значением

Сложные объявления. Константный указатель на константу

```
int a = 8; // два обычных объекта типа int
int b = 1024;

//Константный указатель на целую константу
const int *const  p= &a;

*p=123; // Значение изменять нельзя
```

выражение должно быть допустимым для изменения левосторонним значением

```
p=&b;    // Адрес изменять нельзя
```

const int *const p
Константный указатель на константу

выражение должно быть допустимым для изменения левосторонним значением

Сложные объявления. Пример 1

```
int a = 8;
int b = 1024;

int *const p1 = &a;

// В какой строке ошибка?
      *p1 = 123;
      p1 = &b;
```

Сложные объявления. Пример 2

```
int a = 8;
int b = 1024;

const int * p1 = &a;

// В какой строке ошибка?
      *p1 = 123;
      p1 = &b;
```

Сложные объявления. Пример 3

```
int a = 8;
int b = 1024;

const int *const p1 = &a;

// В какой строке ошибка?
      *p1 = 123;
      p1 = &b;
```

Сложные объявления. Правила чтения

```
int a = 8;
int b = 1024;
```

```
const int *const p1 = &a;
```

Используйте для прочтения сложных объявлений прием

движения изнутри наружу чередуя направо и налево

Сложные объявления. Правила чтения

правила в порядке уменьшения приоритета:

1. Скобки, объединяющие части объявления.

2. Постфиксные операторы:
   - круглые скобки (), обозначающие функцию, `char f();`
   - квадратные [], обозначающие массив. `char f[5];`

3. Префиксный оператор: звёздочка *, обозначающая указатель.

```
char(*f)[5];

char(*f)(char *);
```

Сложные объявления. Правила чтения

```
char* foo[5];
```

1. Начинаем с имени foo: «foo — это…».
2. Дальше идут квадратные скобки: «foo — это массив из пяти…».
3. Осталась звёздочка: «foo — это массив из пяти указателей на…».
4. Добавляем тип и получаем: «foo — это массив из пяти указателей на символ».

Сложные объявления. Правила чтения

```
char(*foo)[5];
```

1. foo - это...
2. foo - это указатель на...
3. foo - это указатель на массив из пяти...
4. foo - это указатель на массив из пяти символов

круглые скобки повысили приоритет звёздочки, поэтому получается указатель на массив

Сложные объявления. Правила чтения

```
char* (*foo)(char*);
```

1. foo - это...
2. foo - это указатель на...
3. foo - это указатель на функцию, принимающую указатель на символ...
4. foo - это указатель на функцию, принимающую указатель на символ, возвращающую указатель на...
5. foo - это указатель на функцию, принимающую указатель на символ, возвращающую указатель на символ

Сложные объявления. Правила чтения

```
char* (*(*foo[5])(char*))[];
```

1. foo - это массив из 5 указателей на...
2. foo - это массив из 5 указателей на функцию, принимающую указатель на символ...
3. foo - это массив из 5 указателей на функцию, принимающую указатель на символ и возвращающую указатель на...
4. foo - это массив из 5 указателей на функцию, принимающую указатель на символ и возвращающую указатель на массив из...
5. foo - это массив из 5 указателей на функцию, принимающую указатель на символ и возвращающую указатель на массив из указателей на.
6. foo - это массив из 5 указателей на функцию, принимающую указатель на символ и возвращающую указатель на массив из указателей на символ

на практике такое не приветствуется ☹

Сложные объявления. Правила чтения. Примеры

```
int(*pf)();
```

указатель на функцию, возвращающую int

```
int * pf();
```

функция, возвращающая указатель на int

```
int * pf(int *);
```

функция от указателя на int, возвращающая указатель на int

Сложные объявления. Правила чтения. Примеры

```
void *(*(*f)(int))[10];
```

указатель на функцию, которая получает аргумент типа int и возвращает указатель на массив из 10 указателей типа void

```
float(*(*f)(int, int, float))(int);
```

указатель на функцию, которая получает три аргумента (int, int и float) и возвращает указатель на функцию, получающую аргумент int и возвращающую float

Сложные объявления. Правила чтения. Примеры

```
int(*(*f())[10])();
```

функция, котjрая возвращает указатель на массив из 10 указателей на функции, возвращающие значения типа int

```
1. char f;
2. char * f();
3. char f[5];
4. char * f[5];
5. char(*f)[5];
6. char *(*f)[5];
7. char(*f)(char *);
8. char* (*f)(char *);
```

Сложные объявления. Ограничение типа void

Тип «void» -это псевдо-тип, и переменные такого типа могут быть только «указатель на» и «функция возвращающая».

```
void *foo;  //разрешено
void foo(); //разрешено
```

Сложные объявления. Ограничение типа void

Запрещено (точнее невозможно) использовать «массив из void» и просто переменные типа «void».

```
void foo[]; //запрещено
```

<error-type> foo[]
запрещено

использование массива типа void не допускается

```
void foo;    //запрещено
```

void foo
запрещено

недопустимый неполный тип

Сложные объявления. Ограничение

Функция не может возвращать функцию

Функция может возвращать указатель на функцию

```
int(*(*foo)())();
```

foo — это указатель на функцию возвращающую указатель на функцию возвращающую значение типа int

Сложные объявления. Ограничение

Функция не может вернуть массив

```
char f()[];
```

использование функции, возвращающей массив, не допускается

функция может вернуть указатель на массив

```
char(*f())[];
char*(*f())[10];
```

Сложные объявления. Конструкция typedef

```
typedef unsigned short UShort;

typedef double(*(*(*myType)())[10])();
myType a, b, c;
```

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() {
// empty, size, length, max_size,
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;
        cout << s.max_size() << endl;//2147483647
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);
}
```

Строки в стиле C++

Примеры

Строки в стиле C++. Операции над строкой

- **clear** - очищает содержимое строки
- **insert** - вставка символов
- **erase** - удаление символов
- **push_back** - добавление символа в конец строки
- **pop_back** - удаляет последний символ
- **append** - добавляет символы в конец строки
- **+=** - добавляет символы в конец строки
- **compare** - сравнивает две строки
- **replace** - заменяет каждое вхождение указанного символа
- **substr** - возвращает подстроку
- **resize** - изменяет количество хранимых символов
- **swap** - обменивает содержимое

Строки в стиле C++. Пример. Часть 1. append

```
#include <iostream>
#include <string>

int main()
{
	std::string str;
	std::string str2 = "Writing ";
	std::string str3 = "print 10 and then 5 more";
```

Строки в стиле C++. Пример. Часть 2. append

```
// всю строку str2 добавляем в str:
str.append(str2);                   // "Writing «
// с 6-й позиции 3 символа
str.append(str3, 6, 3);             // "10 “
// 5 символов в самое начало
str.append("dots are cool", 5); // "dots "
str.append("here: ");               // "here: "
str.append(10, '.');               // ".....«
// " and then 5 more"
str.append(str3.begin() + 8, str3.end());
str.append(5u, '.');                // "....."

std::cout << str << '\n';
return 0; }
```

Writing 10 dots here: ..... and then 5 more.....

Строки в стиле C++. Пример. Часть 1. compare

```
// comparing apples with apples
#include <iostream>
#include <string>

int main()
{
	std::string str1("green apple");
	std::string str2("red apple");
// сравнивает str1 и str2
// 0, когда они равны
	if (str1.compare(str2) != 0)
std::cout << str1 << " is not " << str2 << '\n';
```

green apple is not red apple

Строки в стиле C++. Пример. Часть 2. compare

```
// с 6-й позиции 5 символов
if (str1.compare(6, 5, "apple") == 0)
std::cout << "still, " << str1 << " is an apple\n";

if (str2.compare(str2.size() - 5, 5, "apple") == 0)
std::cout << "and " << str2 << " is also an apple\n";

// с 6-й позиции 5 символов для str1
// с 4-й позиции 5 символов для str2
if (str1.compare(6, 5, str2, 4, 5) == 0)
std::cout << "therefore, both are apples\n";

return 0;
}
```

still, green apple is an apple
and red apple is also an apple
therefore, both are apples

Строки в стиле C++. Пример. Часть 1. replace

```
// replacing in a string
#include <iostream>
#include <string>

int main()
{ // Using positions:     0123456789*123456789*12345
       std::string base = "this is a test string.";
       std::string str2 = "n example";
       std::string str3 = "sample phrase";
       std::string str4 = "useful.";
```

Строки в стиле C++. Пример. Часть 2. replace

```
// "this is a test string."
std::string str = base;
//  0123456789*123456789*12345
// "this is an example string." (1)
str.replace(9, 5, str2);

// "this is an example phrase." (2)
str.replace(19, 6, str3, 7, 6);

// "this is just a phrase."     (3)
str.replace(8, 10, "just a");

// "this is a short phrase."    (4)
str.replace(8, 6, "a shorty", 7);

// "this is a short phrase!!!"  (5)
str.replace(22, 1, 3, '!');
```

```
base = "this is a test string.";
str2 = "n example";
str3 = "sample phrase";
str4 = "useful.";
```

Строки в стиле C++. Пример. Часть 3. replace

```
// Using iterators:

// "sample phrase!!!"
str.replace(str.begin(), str.end() - 3, str3);

// "replace phrase!!!"
str.replace(str.begin(), str.begin() + 6, "replace");
```

```
base = "this is a test string.";
str2 = "n example";
str3 = "sample phrase";
str4 = "useful.";
```

Строки в стиле C++. Пример. substr

```
std::string str = "We think in generalities, but we live in details.";
// (quoting Alfred N. Whitehead)

std::string str2 = str.substr(3, 5);  // "think"

std::size_t pos = str.find("live");   // position of "live" in str

std::string str3 = str.substr(pos);   // get from "live" to the end

std::cout << str2 << ' ' << str3 << '\n';}
```

think live in details.

Строки в стиле C++. Пример. swap

```
// swap strings
#include <iostream>
#include <string>
int main()
{
std::string buyer("money");
std::string seller("goods");

std::cout << "Before the swap, buyer has " << buyer;
std::cout << " and seller has " << seller << '\n';

seller.swap(buyer);

std::cout << " After the swap, buyer has " << buyer;
std::cout << " and seller has " << seller << '\n';
return 0;}
```

Before the swap, buyer has money and seller has goods
After the swap, buyer has goods and seller has money

Строки в стиле C++. Функции поиска

- **find** - поиск символов в строке
- **rfind** - поиск последнего вхождения подстроки
- **find_first_of** - поиск первого вхождения символов
- **find_first_not_of** - найти первое вхождение отсутствия символов
- **find_last_of** - найти последнее вхождение символов
- **find_last_not_of** - найти последнее вхождение отсутствия символов

Строки в стиле C++. Пример. Часть 1. find

```
// string::find
#include <iostream>       // std::cout
#include <string>         // std::string

int main()
{
      std::string str("There are two needles in this
      haystack with needles.");

      std::string str2("needle");
```

Строки в стиле C++. Пример. Часть 2. find

```
// different member versions of find in the same order as
above:

std::size_t found = str.find(str2);

// std::string::npos - maximum value for size_t
if (found != std::string::npos)
std::cout << "first 'needle' found at: " << found << '\n';
```

first 'needle' found at: 14

Строки в стиле C++. Пример. Часть 3. find

```
found = str.find("needles are small", found + 1, 6);

if (found != std::string::npos)
std::cout << "second 'needle' found at: " << found << '\n';
```

second 'needle' found at: 44

Строки в стиле C++. Пример. Часть 4. find

```
found = str.find("haystack");

if (found != std::string::npos)
std::cout << "'haystack' also found at: " << found <<
'\n';
```

'haystack' also found at: 30

Строки в стиле C++. Пример. Часть 5. find

```
// let's replace the first needle:
str.replace(str.find(str2), str2.length(), "preposition");

std::cout << str << '\n';

return 0;
}
```

There are two prepositions in this haystack with needles.

Строки в стиле C++. Проверка на вместимость строки

- **empty** - проверяет, является ли строка пустой
- **size** - возвращает количество символов в строке
- **length** - возвращает количество символов в строке
- **max_size** - возвращает максимальное количество символов

```
#include <iostream>
#include <string>
using namespace std;
int main() {

        string s = "a priori";
        cout << s.size() << endl;     //8
        cout << s.length() << endl;   //8
        cout << s.empty() << endl;    //0
        cout << s.max_size() << endl;//2147483647

return 0;
}
```

Спасибо за внимание!