Языки программирования

Лекция 6

ПМИ 2 курс

Демяненко Я.М.

ЮФУ 2025

Строки в стиле C++

В 1980 году появился класс string:

```
string s1 = "Hello ";
string s2 = "world !!!";

s1.size();    // s1 — экземпляр класса, он помнит свою длину
s1[0];        // Индексация символов с нуля
s1 = s2;      // Строки можно присваивать друг другу
s1 += s2;     // Строки можно прибавлять друг другу
s1 == s2;
s1 < s2;      // Строки можно сравнивать
```

Преимущества

- Скрывает способ хранения строки
- Длина строки контролируется и автоматически увеличивается при необходимости
- Снижается вероятность ошибок
- Перегружены операции + и +=
- Можно присваивать одной строке значение другой

Несмотря на то, что скорость работы С-строк немного выше, чем у класса string, на прикладном уровне лучше использовать string.
Однако системным программистам чаще приходится пользоваться char*.

Ввод-вывод

```
cout << s
cin >> s;
```

Опять пробелы((

Если мы вводим "Hello world" в s будет храниться только "Hello", поэтому надо использовать

```
getline(cin, s);
```
или
```
getline(cin, s,'!');
```

Считывает неформатированные данные из потока в строку. Останавливается, как только найден символ, равный разделителю, или исчерпан поток.
Первая версия использует в качестве разделителя '\n', вторая — ‘!’.
Символ-разделитель удаляется из потока и не помещается в строку.

Как передавать в функции C++-строки

C++-строки

```
void p1(string &s)          // s можно менять
void p1(const string &s)    // s нельзя менять
```

Обработка строк в стиле языка C++

Класс string

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

 string s1,s2;                    //Пустые строки

 string s3 = "Hello!";            //Инициализированная строка
 string s4("I am");               //Еще один пример инициализации
 string s5(s3);                   //И еще один пример инициализации

 s2 = "By";                       //Присваивание

 s1 =s3 + " " + s4;               //Слияние строк – конкатенация
 s1 += " Good ";
 cout  << s1 + s2 + "!" << endl;
```

```
string ident1 ("max");
string ident2 ("min");
char ident3[]="sum";
…
if (ident1<ident2)
ident1.append("less");
if (ident1==ident3)
ident1.append("equal");
if ("avg"!=ident2)
ident2.append("not equal");
```

Работа с переменными класса string через методы класса

```
string s;
s = "string srt srt";
cout <<  s.length() << endl;
int pos = s.find("St");
if (pos != string::npos)
  cout<<pos<<endl;
else
  cout <<"Not found\n";
```

Переменная **string::npos** является статическим элементом класса string.

Ее значение равно максимально возможному количеству символов в строковом объекте 4294967295.

Это значение используется как признак неудачного завершения поиска

Можно работать со строкой посимвольно

```
string s = "Hello!";
for (int i=0; i <s.length(); ++i)
      cout<<s[i];
s[5]=‘N’;
```

Рекомендуется не смешивать использование строк в стиле С и строк в стиле С++.

Для преобразования строк из типа string в тип char* используется метод **c_str()** класса string, который возвращает указатель типа const char * на строку, содержащую те же символы, что и строка типа string.

Необходимость такого преобразования возникает крайне редко только в тех случаях, когда требуется использовать для обработки строк функции библиотеки cstring.

```
string s1="C++";
const char *s2;
s2 = s1.c_str();
```

Пример

Заменить в строке s1 каждое вхождение подстроки s2 подстрокой s3.

```
//ReplaceAll.h

#ifndef REPLACEALL_H
#define REPLACEALL_H
#include <string>
void replaceAll (string &context, const string &from,  const string & to);
#endif    //REPLACEALL_H
```

```
//ReplaceAll.cpp

#include "ReplaceAll.h"
using namespace std;

void replaceAll (string & context, const string & from, const string & to) {
  size_t lookHere=0;
  size_t foundHere;
  while ((foundHere = context.find(from, lookHere)) != string::npos) {
    context.replace(foundHere, from.size(), to);
    lookHere = foundHere + to.size();
  }
}
```

Пространства имён

- Пространство имён — это декларативная область, в рамках которой определяются различные идентификаторы (имена типов, функций, переменных, и т. д.).

- Пространства имён используются для организации кода в виде логических групп и **с целью избежания конфликтов имён**, которые могут возникнуть, особенно в таких случаях, когда база кода включает несколько библиотек.

Доступ

Все идентификаторы в пределах пространства имён доступны друг другу без уточнения.

Идентификаторы за пределами пространства имен могут получить доступ к членам с помощью:

1. полного имени для каждого идентификатора, например std::vector<std::string> vec;
2. объявления для одного идентификатора using std::string;
3. директивы using для всех идентификаторов в пространстве имен using namespace std;

Код в **файлах заголовков** всегда должен **содержать полное имя** в пространстве имен.

Создание пространства имён

```
namespace MyNamespace {
	class ObjectManager {
		public: void DoSomething() {…}
	};
	void Func(ObjectManager) {…}
}
```

Использование полного имени

```
MyNamespace ::ObjectManager mgr;
mgr.DoSomething();
MyNamespace ::Func(mgr);
```

Добавить в область видимости один идентификатор

```
using MyNamespace ::ObjectManager;
ObjectManager mgr;
mgr.DoSomething();
```

Добавить в область видимости все идентификаторы пространства имён

```
using namespace MyNamespace;

ObjectManager mgr;
mgr.DoSomething();
Func(mgr);
```

Рекомендации

- Директиву using можно поместить в верхнюю часть cpp-файла (в области видимости файла) или внутрь определения класса или функции.

- Без особой необходимости не размещайте директивы using в файлах заголовков (*.h), так как любой файл, содержащий этот заголовок, добавит все идентификаторы пространства имён в область видимости, что может вызвать скрытые конфликты имён или конфликты, которые очень трудно отлаживать.

- В файлах заголовков всегда используйте полные имена. Если эти имена получаются слишком длинными, используйте псевдоним пространства имен для их сокращения.

```
//contosoData.h
#pragma once
namespace ContosoDataServer {
    void Foo();
    int Bar();
}

//contosoData.cpp
#include "contosodata.h"
using namespace ContosoDataServer;

// use fully-qualified name here
void ContosoDataServer::Foo() {
        Bar(); // no qualification needed for Bar()
}

int ContosoDataServer::Bar(){return 0;}
```

Реализации функций в .cpp-файле должны использовать полное имя, даже если вы размещаете using директиву в верхней части файла

Объявления пространства имен

- Пространство имён может быть объявлено в нескольких блоках в одном файле или в нескольких файлах.
- Компилятор соединит вместе все части во время предварительной обработки и полученное в результате пространство имён будет содержать все члены, объявленные во всех частях.
- Примером этого является пространство имён std, которое объявляется в каждом из файлов заголовка в стандартной библиотеке.

```
namespace V {
   void f();
}

void V::f() { }       // ok
void V::g() { }       // C2039, g() is not yet a member of V

namespace V {
   void g();
}
```

Эта ошибка может возникнуть, когда члены пространства имён объявляются в нескольких файлах заголовка и эти заголовки не включены в правильном порядке.

Глобальное пространство имён

Если идентификатор не объявлен явно в пространстве имён, он неявно считается входящим в глобальное пространство имён

Старайтесь избегать объявления в глобальной области, если это возможно, за исключением основной функции точки входа, которая должна находиться в глобальном пространстве имён

Чтобы явно указать глобальный идентификатор, используйте оператор разрешения области видимости без имени
::SomeFunction(x);

Пространство имён std

Все типы и функции стандартной библиотеки C++ объявляются в пространстве имён std или пространствах имён, вложенных внутри std.

Вложенные пространства имён

```
namespace MyNamespace {
    void Foo();

    namespace Details {
        int CountImpl;
        void Ban() { return Foo(); }
    }

      int Bar(){...};
      int Baz(int i) { return Details::CountImpl;}
}
```

Псевдонимы пространств имён

Имена пространств имен должны быть уникальными, из-за чего зачастую они получаются не слишком короткими.

Можно создать псевдоним пространства имен, который служит сокращенным для фактического имени.

```
namespace a_very_long_namespace_name { class Foo {}; }
namespace AVLNN = a_very_long_namespace_name;
void Bar(AVLNN::Foo foo){ }
```

Анонимные или безымянные пространства имён

```
namespace {
   int MyFunc(){}
}
```

Это называется неименованным или анонимным пространством имён, и полезно, если нужно сделать объявления переменных невидимыми для кода в других файлах, не создавая именованное пространство имён.

Весь код, находящийся в том же файле, может видеть идентификаторы в безымянном пространстве имён, но эти идентификаторы, а также само пространство имён, будет невидимым за пределами этого.