Программирование и алгоритмизация

Санкт-Петербургский государственный политехнический университет

19 ноября 2014

Ключевое слово extern

Используется для того, чтобы сообщить компилятору, что понятие определенр в другом модуле, например, в другом файле.

```
//Server.cpp
int x;
int arr[10][20];

//Client.cpp
extern int x;
extern arr[][20];

int main() { x = 7; }
```

В случае, если переменная с таким именем не была определена в программе, то возникает ошибка при ЛИНКОВКЕ.
Массив объявленный с ключевым словом extern должен содержать информацию обовсех младших размерностях.

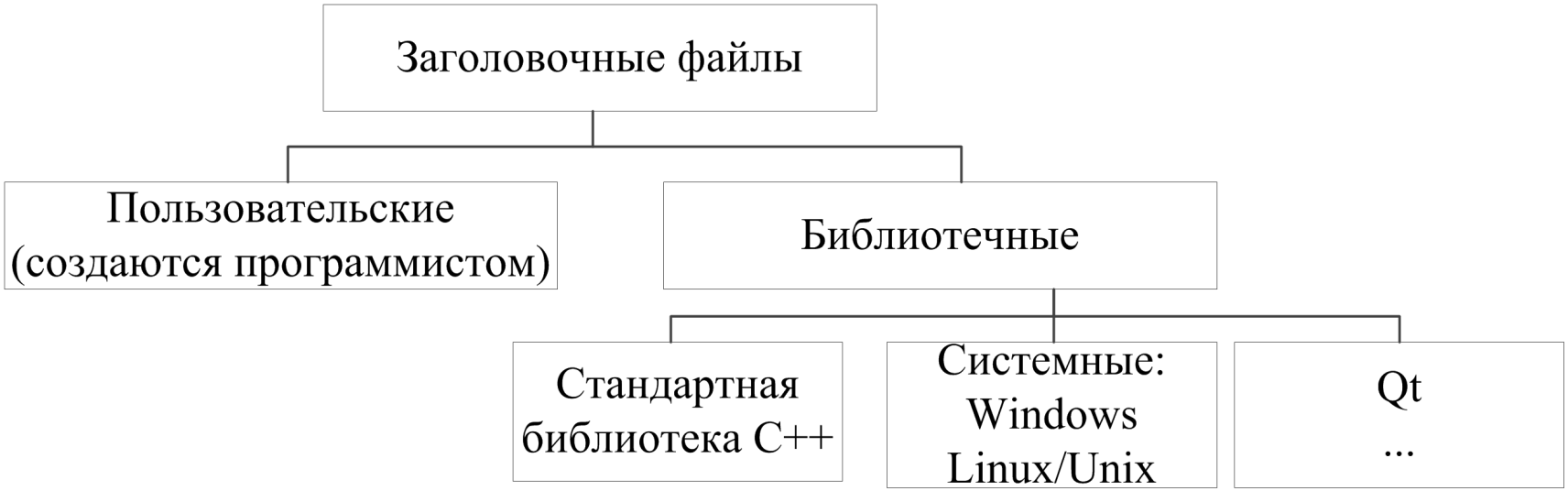
Заголовочные файлы

Задаются директивой препроцессора #include, обычно содержат:

- описания пользовательских внешних зависимостей;
- объявление внешних переменных;
- определение встроенных (inline) функций;
- перечисления;
- директивы условной трансляции и макроопределения.
- определение констант и объявления функций;

Позволяют эффективно использования функции стандартной библиотеки, а также функции сторонних библиотек.

Группы заголовочных файлов



Заголовочные файлы

Чего НЕ ДОЛЖНО быть в заголовочных файлах:

- определений невстроенных функций int f() { /* ... *// }
- определений данных: int a; char Arr[] = { 'a', 'b', 'c' };
- неименованых пространств имен namespace {};

Разрешение внешних зависимостей

Неэффективное:

```
//Server.cpp
int X;

//Client1.cpp
extern int X;

void f()
{
  X = 5;
}

//Client2.cpp
extern int X;

void f2()
{
  X = 6;
}
```

Разрешение внешних зависимостей

Эффективное: Разделить на два файла: файл интерфейса (заголовочный файл) и файл реализации

```
//Server.cpp
int X;

//Server.h
extern int X;

//Client1.cpp
#include "Server.h"
void f() {
  X = 5;
}

//Client2.cpp
#include "Server.h"
void f2() {
  X = 6;
}
```

Заголовочные файлы

Формы директивы include

```
#include <файл>
#include "файл"

#include <iostream>
#include "myHeader.h"
```

Заголовочные файлы

Защита от повторного включения заголовочных файлов

```
//Client.cpp
#include "1.h"
#include "2.h"

//1.h
#include "2.h"

//2.h
const int N = 5;
```

Заголовочные файлы

Защита от повторного включения заголовочных файлов

```
//Client.cpp
#include "1.h"
#include "2.h"

//1.h
#include "2.h"

//2.h
#if !defined 2_H //#ifndef 2_H
#define 2_H

const int N = 5;

#endif //2_H
```

Заголовочные файлы

Защита от повторного включения заголовочных файлов, через директиву pragma. pragma — передает настройки компилятору или линковщику.

```
//Client.cpp
#include "1.h"
#include "2.h"

//1.h
#include "2.h"

//2.h
#pragma once

const int N = 5;
```

Ссылки (references)

Тип появившийся в C++, ссылка в отличие от указателя неявно содержит адрес. В отличие от указателя подчиняется разным синтаксическим правилам.

double* :	double& :
double* p = x;	double& ref = x;

Ссылки (references)

Инициализация ссылки в отличие от указателя является обязательной!

```
double *p;

double &pRef; //???

extern double &pRef2; //???
```

Ссылки (references)

Получение значений по ссылке и указателю:

```
double tmp = 0;
double *p = &tmp;
(*p)++;

double &pRef = tmp;
pRef++;
```

Ссылки (references)

Модификация адреса на другую переменную, для типа ссылок не возможна.

```
double z = 7;
*p = &z; //???

pRef = z; //???

p = nullptr;
pRef = 0;
```

Ссылки (references)

Ссылки на ссылки (rvalue reference) из стандарта C++11.

```
double **pP = &p;

double &&ref = pRef;
```

Ссылки (references)

Инициализация ссылок:

```
int &r1 = 1; //???

const int &r2 = 5; //???

int &&r3 = 7; //???
```

Ссылки (references)

Ссылки на ссылки (rvalue reference) из стандарта C++11.

```
int i;

void *p = &i; //??
void &pRef = i; //??

int *p2 = 7; //?? FIXME

int &iRef = 7; //??
const int iRef = 6; //??
int &&r = 10; //??
```

Ссылки (references)

Ссылки на указатели:

```
double d = 4;
double *p = &d;

double *&refP = p;
*refP = 3.3; //???

double *&refM = &d;
```

Ссылки (references)

sizeof и ссылки

```
double d;
double *p = &d;
sizeof(p);   //
sizeof(*p);  //

double &dRef = d;
sizeof(dRef);
```

Функции

Использование функций позволяет:

- выполнять одни и теже действия с разными наборами данных;
- использовать посредством функций чужой код;
- улучшить читаемость и структуру программы.

Функции

Использование функции состоит из 3-х частей:

- объявление (прототип);
- определение (реализация, тело функции);
- вызов функции.

Функции

Пример:

```
//Server.h
объявление_функции();

//Server.cpp
определение_функции() { /* код функции */ }

//Client1.cpp
#include "Server.h"
вызов_функции();

//Client2.cpp
#include "Server.h"
вызов_функции();
```

Функции

Объявление функции:

```
[спецификатор] [тип] [соглашение по вызову]
    имя_функции( [список аргументов] || [void] );
```

Тип возвращаемого значения может быть как базового так и пользовательского типа. Либо указан, как тип void, что означает, что функция не возвращает значения.
В случае отсутствии поля «тип» — подставляется тип int.
Замечание: большинство компиляторов C++, выдают при этом ошибку.

Список параметров

```
(тип [имя], тип [имя])

char *strcpy(char *dest, const char *source); //
    прототип функции

strcpy(str1, str2); //вызов функции
```

Имя параметра является не обязательным в случае, если значение передаваемого параметра не используется.
Объявление функции (прототип) используется для проверки соответствия фактического и формального количества параметров в функции, а также их типов.

Определение функции

```
double Formula(double x) { //имя формального
    параметра
  const double A = 3.4, B = 4.4, C = 5.6;
  double result = A * x * x + B * x + C;
  return result;
}
double res = Formula(6);
```

Время жизни переменных внутри функции эквивалентно времени жизни локальных переменных в области видимости блока.

ИМЕНА формальных параметров в определении и объявлении могут различаться, в объявлении имена могут отсутствовать.

Вложенные функции в языке C/C++ не допускаются.

Возврат значения осуществляется конструкцийе return.

Имя функции — является указателем на начало блока выполнения команд (по аналогии с массивами, где имя массива — адрес первого элемента).

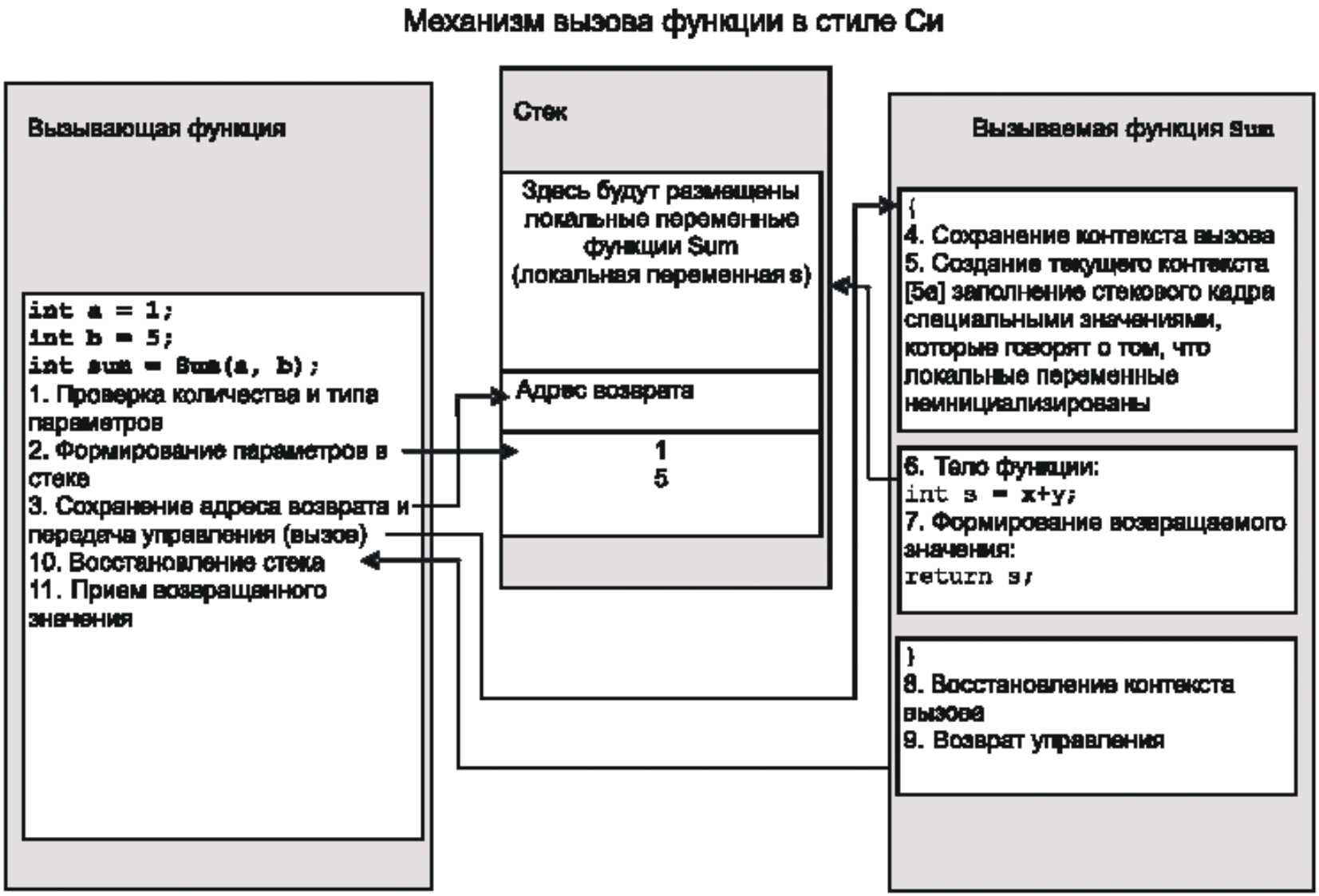
Пример функции

```cpp
//Sum.cpp
int sum(int a, int b)
{
  int res = a + b;
  return res;
}

//Sum.h
int sum(int a, int b);

//Client
int main()
{
  int x = 5;
  int y = 7;
  int c = sum(5, 7);
  int c2 = sum(x, y);
  return 0;
}
```

Механизм вызова функций



Возвращаемые значения

Если программист в объявлении и определении описал функцию как возвращающую значение, то в теле этой функциии должна быть инструкция return <выражение>. При выполнении инструкции return выполняются следующие действия:

- вычисляется выражение;
- формируется возвращаемое значение и при необходимости приводится к нужному типу;
- управление передается на закрывающую фигурную скобку функции, по которой восстанавливается контекст вызывающей функции и возвращается управление вызывающей функции.

Если функций ничего не возвращает void, то она может быть прервана инструкцией return; в любой момент времени.

Пример

```
int compare(int a)
{
  if(a > 0)
    return 1;
  else if(a < 0)
    return -1;
  else
   return 0;
} //переписать через тернарный
```

Встроенные функции

Функцию определенную со спецификатором inline называют встроенной.
Такие функции исключают накладные расходы на вызов функции и сохранение контекста, выполняю простую подстановку исхода кода функции, на место вызова этой функции. В результате может увеличиться объем исполняемого файла.
Обычно очень маленькие функции объявляют со спецификатором inline.
При описании такой функции объявление должно быть совмещено определением, и обычно содержится в заголовочном файле программы.
Ключевое слово inline, является рекомендацией компилятору, которое он может проигнорировать.

Пример встроенной функции

```
//Sum.h

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

//Client.cpp
#include "Sum.h"
int main()
{

  int res1 = sum(6, 7);
  res1 = sum(7, res1);
}
```

Замена define на inline

```
#define MAX(x, y) (x > y ? (x) : (y))
#define DIV(x) (x/2)

inline int Max(int x, int y) { return x > y ? x : y
    ; }
inline double Div(double x) { return x/2; }

int main()
{
  int a = 7, b = 8;
  double res1 = MAX(a, b); //??
  double res2 = Max(a, b); //??
  double dv = DIV(3); //??
  dv = Div(3); //??
}
```

Способы передачи параметров

- по значению — используется, когда требуется сохранить исходное передаваемое значение в функцию;
- по адресу — функция получает не копию объекта, а его адрес — это позволяет программисту изменять значение самого объекта:
  - по указателю;
  - по ссылке.

Примеры передачи параметров

```
void change(int a, int b)
{
  a = 5;
  b = 7;
}

void change2(int &a, int &b)
{
  a = 5;
  b = 7;
}

void change4(int *a, int *b)
{
  //FIXME
}
```

Примеры передачи параметров (продолжение)

```
int main()
{
  int c = 0, d = 1;
  change(c, d); // c - ? d - ?
  change2(c, d); //c - ? d - ?

  return 0;
}
```