
Алгоритмизация и программирование I

Лекция 7



Что будет напечатано после выполнения следующих функций?

```
void swap1 (int a, int b)
{
    int r;
    r=a; a=b; b=r;
    cout<<a<<' '<<b<<endl;
}
```

```
void swap2 (int &a, int &b)
{
    int r;
    r=a; a=b; b=r;
    cout<<a<<' '<<b<<endl;
}
```

```
void swap3 (int *a, int *b)
{
    int r;
    r=*a; *a=*b; *b=r;
    cout<<*a<<' '<<*b<<endl;
}
```

```
void main()
{
    int a=10, int b=-50;
    cout<<a<<' '<<b<<endl;
    swap1(a,b);
    cout<<a<<' '<<b<<endl;
    swap2(a,b);
    cout<<a<<' '<<b<<endl;
    swap3(&a,&b);
    cout<<a<<' '<<b<<endl;
}
```



Лекция 7

- Составные части программы, локальные и глобальные переменные
- Функции
 - Параметры функции по умолчанию
 - Перегрузка функций
- Программные модули
- Генератор случайных чисел

Составные части программы

- Подключение *заголовочных файлов* — это строки, которые начинаются с **#include**
- •Объявление *констант* (постоянных величин):
 - **const N = 20;**
- *Глобальные переменные* — это переменные, объявленные вне основной программы и подпрограмм. К таким переменным могут обращаться все функции данной программы (их не надо еще раз объявлять в этих функциях).
- *Объявление функций* — обычно ставятся выше основной программы. По требованиям языка Си в тот момент, когда транслятор находит вызов подпрограммы, она должна быть объявлена и известны типы всех ее параметров.
- •*Основная программа* может располагаться как до всех подпрограмм, так и после них. Не рекомендуется вставлять ее между подпрограммами, так как при этом ее сложнее найти.

Локальные и глобальные переменные

- **Глобальные переменные** доступны из любой функции. Поэтому их надо объявлять вне всех подпрограмм.
- **Локальные переменные** объявляются в функциях, известны только той подпрограмме, где они объявлены.

Пример

```
#include <stdio.h>
int var = 0;           // объявление глобальной переменной
void ProcNoChange ()
{
    int var;          // локальная переменная
    var = 3;          // меняется локальная переменная
}
void ProcChange1 ()
{
    var = 5;          // меняется глобальная переменная
}
void ProcChange2 ()
{
    int var;          // локальная переменная
    var = 4;          // меняется локальная переменная
    ::var = ::var * 2 + var; // меняется глобальная переменная
}
void main()
{
    ProcChange1();    // var = 5;
    ProcChange2();    // var = 5*2 + 4 = 14;
    ProcNoChange();   // var не меняется
    printf ( "%d", var ); // печать глобальной переменной (14)
}
```


Глобальные и локальные переменные

- Глобальные переменные не надо заново объявлять в подпрограммах.
- Если в подпрограмме объявлена локальная переменная с таким же именем, как и глобальная переменная, то используется **локальная** переменная.
- Если имена глобальной и локальной переменных совпадают, то для обращения к глобальной переменной в подпрограмме перед ее именем ставится два двоеточия:
 - `::var = ::var * 2 + var;`
- Рекомендуются использовать как можно меньше глобальных переменных, а лучше всего – не использовать их вообще, потому что глобальные переменные
 - затрудняют анализ и отладку программы;
 - повышают вероятность серьезных ошибок — можно не заметить, что какая-то подпрограмма изменила глобальную переменную;
 - увеличивают размер программы, так как заносятся в блок данных, а не создаются в процессе выполнения программы.
- Поэтому глобальные переменные применяют в крайних случаях:
 - для хранения глобальных системных настроек (цвета экрана и т.п.);
 - если переменную используют три и более подпрограмм и по каким-то причинам неудобно передавать эти данные в подпрограмму как параметры.
- Везде, где можно, надо передавать данные в функции через их параметры. Если же надо, чтобы подпрограмма меняла значения переменных, надо передавать параметр по ссылке.

Повторение. Задание

- Дана последовательность N натуральных чисел. Найти число с максимальной суммой цифр.

Программа

```
#include <iostream>
using namespace std;
int sum_cifr(int x)
{
    int w=0;
    while(x)
    {
        w+=x%10;
        x/=10;
    }
    return w;
}
```

```
void main()
{
    int n,a,s,smax(0),amax(0);
    cout<<"n="; cin>>n;
    for (int i=1;i<=n;i++)
    {
        cin >>a;
        s=sum_cifr(a);
        if(s>smax)
        {
            smax=s; amax=a;
        }
    }
    cout<< "Число с максимальной суммой
цифр " << amax<<"\n";
}
```

Параметры функции по умолчанию

- С++ позволяет программам указывать для параметров значения по умолчанию.
- Значения по умолчанию для параметров указываются в заголовке функции при ее определении.
- Если вызов функции опускает значения одного или нескольких параметров, С++ будет использовать значения по умолчанию.
- Если вызов функции опускает значение определенного параметра, то должны быть опущены и значения всех последующих параметров.

Определение значений по умолчанию

- Чтобы обеспечить значения по умолчанию для параметров функции, надо присвоить значение параметру с помощью операции присваивания прямо при объявлении функции:

```
void yyy (int x, int a=0, int b=1)
{ cout<<x<<' '<<a<<' '<<b<<endl;}
void main()
{ yyy(5); yyy(6,1001), yyy(7,1001, 1002);}
```

C:\WINDOWS\system32\cmd.exe

5 0 1

6 1001 1

7 1001 1002

Для продолжения нажмите любую клавишу . . .

Правила для пропуска значений параметров

- При описании параметры по умолчанию должны оказываться в конце списка.
- Если программа опускает определенный параметр для функции, обеспечивающей значения по умолчанию, то следует опустить и все последующие параметры.
- Другими словами, нельзя опускать средний параметр.

Перегрузка функций

- Перегрузка функций позволяет использовать одно и то же имя для нескольких функций с разными типами параметров.
- Для перегрузки функций просто определите две функции с одним и тем же именем и типом возвращаемого значения, которые отличаются количеством параметров или их типом.

Пример перегрузки функций

- Допустим, описаны функции:
int min (int x, int y) {return x<y ? x :y;}
double min (double x, double y) {...}
int min (int x, int y, int z)
{ int m; m=x<y? x:y;
return m<z? m :z;}

Основная программа:

```
double a, b; int k, m, n;  
  
cout<<min(a,b)<<endl;  
cout <<min (k,m)<<endl;  
cout <<min (k,m,n)<<endl;
```

Модульное программирование

- Основной метод структурного программирования
- Предполагает разделение текста программы на несколько файлов, в каждом из которых сосредоточены независимые части программы (сгруппированные по смыслу функции) - модули.
- Каждый из этих модулей можно компилировать отдельно, а затем объединить их в процессе построения конечной программы.

Использование библиотечных функций

- Для использования библиотечных функций необходимы два файла (для примера рассмотрим функцию `sqrt()`):
 - **заголовочный файл** (`<math.h>`) с прототипом функции `sqrt()` и многих других библиотечных функций (`#include <math.h>`);
 - **файл реализации** (для пользовательских функций – это файлы с исходным текстом на C++, а библиотечные функции обычно хранятся в скомпилированном виде в специальных библиотечных файлах, например, «`libcmt.d.lib`»). Файлы реализации пользовательских функций (расширение «`.cpp`») содержат определения этих функций.

Создание и использование модулей

- Чтобы создать отдельные модули программы, необходимо в окне Solution Explorer с помощью контекстного меню для пункта Source File создать отдельные файлы проекта. Все они будут иметь расширение .cpp и будут компилироваться отдельно (пункт меню Build, Compile или Ctrl-F7).
- Основная программа при этом может содержать только главную функцию main и прототипы всех остальных функций, которые содержатся в других модулях.
- После успешной компиляции каждого отдельного модуля необходимо выполнить компоновку проекта с помощью пункта Build, Build Solution или клавиши F7.
- Далее можно запускать программу на выполнение обычным способом.

Пример

- Написать программу с использованием модульного программирования.
- Вводятся три целых числа. Найти среднее двух и трех чисел.

Программа

- Программа состоит из трех файлов:
 - главного файла;
 - заголовочного файла с описаниями двух функций расчета среднего значения;
 - соответствующего файла реализации.

Главный файл

```
#include "averages.h"
void main()
{
    setlocale(LC_ALL,"rus");
    int A, B, C;
    cout << "Введите три целых числа ";
    cin >> A >> B >> C;
    cout << "Целочисленное среднее чисел " << A << " и ";
    cout << B << " равно ";
    cout << average2(A, B) << ".\n";
    cout << "Целочисленное среднее чисел " << A << ", ";
    cout << B << " и " << C << " равно ";
    cout << average3(A, B, C) << ".\n";
}
```

Заголовочный файл averages.h

```
#include <iostream>
#include <locale.h>
using namespace std;
```

```
// ПРОТОТИП ФУНКЦИИ ДЛЯ ВЫЧИСЛЕНИЯ  
ЦЕЛОЧИСЛЕННОГО СРЕДНЕГО
```

```
// ЗНАЧЕНИЯ 3-Х ЦЕЛЫХ ЧИСЕЛ:
```

```
int average3( int numb1, int numb2, int numb3);
```

```
// ПРОТОТИП ФУНКЦИИ ДЛЯ ВЫЧИСЛЕНИЯ  
ЦЕЛОЧИСЛЕННОГО СРЕДНЕГО
```

```
// ЗНАЧЕНИЯ 2-Х ЦЕЛЫХ ЧИСЕЛ:
```

```
int average2( int numb1, int numb2);
```


Файл реализации averages.cpp

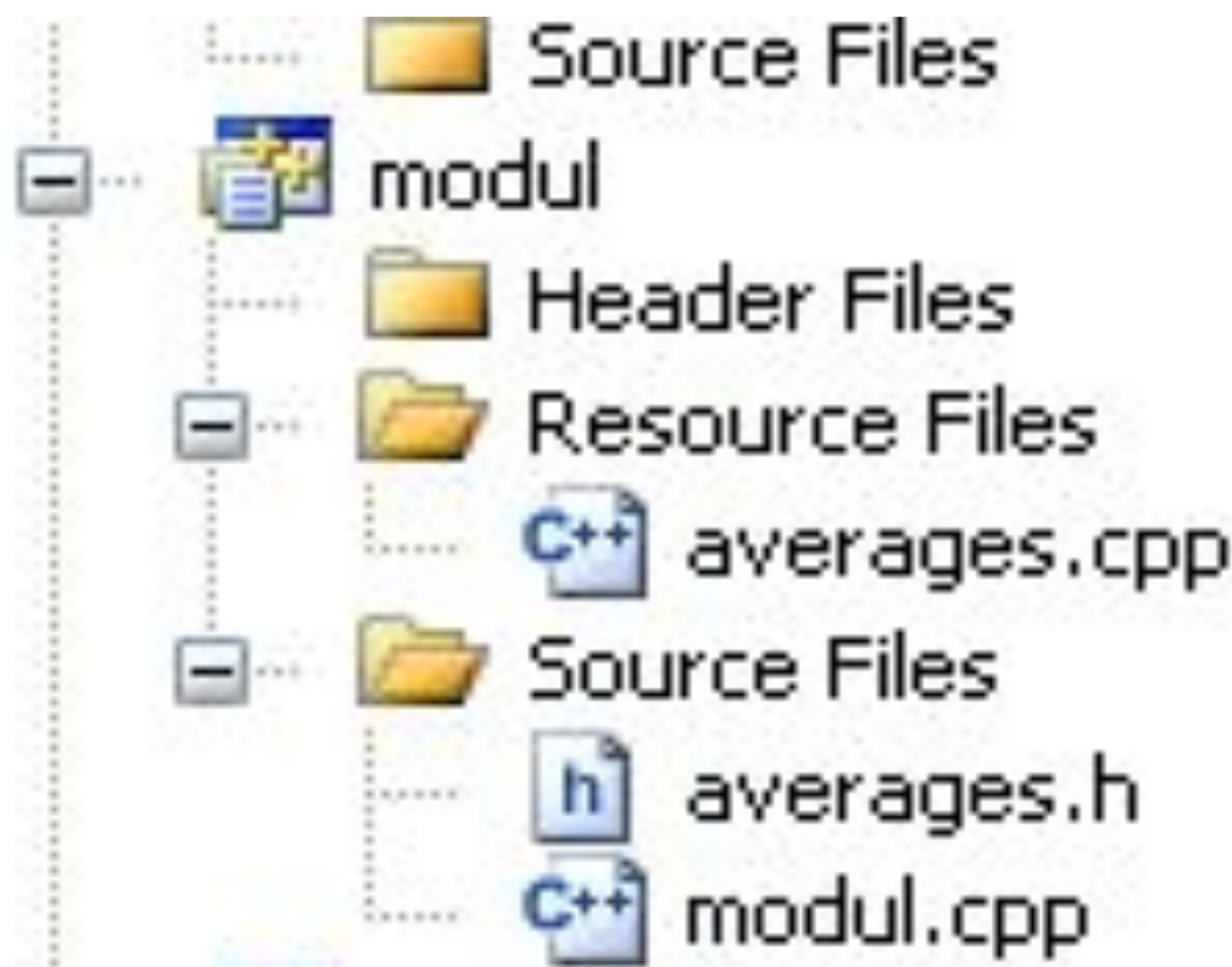
```
#include "averages.h"
```

```
// ФУНКЦИЯ ДЛЯ ВЫЧИСЛЕНИЯ ЦЕЛОЧИСЛЕННОГО СРЕДНЕГО  
// ЗНАЧЕНИЯ 3-Х ЦЕЛЫХ ЧИСЕЛ:
```

```
int average3( int numb1, int numb2, int numb3 )  
{  
    return ((numb1 + numb2 + numb3)/3);  
}
```

```
// ФУНКЦИЯ ДЛЯ ВЫЧИСЛЕНИЯ ЦЕЛОЧИСЛЕННОГО СРЕДНЕГО  
// ЗНАЧЕНИЯ 2-Х ЦЕЛЫХ ЧИСЕЛ:
```

```
int average2( int numb1, int numb2 )  
{  
    return ((numb1 + numb2)/2);  
}
```



modul.cpp

averages.cpp

averages.h

prog.cpp

pr2.cpp

prim1.cpp

(Global Scope)

main()

```
#include "averages.h"
```

```
void main()
```

```
{
```

```
    setlocale(LC_ALL, "rus");
```

```
    int A, B, C;
```

```
    cout << "Введите три целых числа ";
```

```
    cin >> A >> B >> C;
```

```
    cout << "Целочисленное среднее чисел " << A << " и ";
```

```
    cout << B << " равно ";
```

```
    cout << average2(A, B) << ".\n";
```

```
    cout << "Целочисленное среднее чисел " << A << ", ";
```

```
    cout << B << " и " << C << " равно ";
```

```
    cout << average3(A, B, C) << ".\n";
```

```
}
```

modul.cpp

averages.cpp

averages.h

prog.cpp

pr2.cpp

prim1.cpp

(Global Scope)

```
#include "averages.h"

// ФУНКЦИЯ ДЛЯ ВЫЧИСЛЕНИЯ ЦЕЛОЧИСЛЕННОГО СРЕДНЕГО
// ЗНАЧЕНИЯ 3-Х ЦЕЛЫХ ЧИСЕЛ:
int average3( int numb1, int numb2, int numb3 )
{
    return ((numb1 + numb2 + numb3)/3);
}

// ФУНКЦИЯ ДЛЯ ВЫЧИСЛЕНИЯ ЦЕЛОЧИСЛЕННОГО СРЕДНЕГО
// ЗНАЧЕНИЯ 2-Х ЦЕЛЫХ ЧИСЕЛ:
int average2( int numb1, int numb2)
{
    return ((numb1 + numb2)/2);
}
```

modul.cpp

averages.cpp

averages.h

prog.cpp

pr2.cpp

prim1.cpp

(Global Scope)

```
#include <iostream>
#include <locale.h>
using namespace std;

// ПРОТОТИП ФУНКЦИИ ДЛЯ ВЫЧИСЛЕНИЯ ЦЕЛОЧИСЛЕННОГО СРЕДНЕГО
// ЗНАЧЕНИЯ 3-Х ЦЕЛЫХ ЧИСЕЛ:
int average3( int numb1, int numb2, int numb3);
// ПРОТОТИП ФУНКЦИИ ДЛЯ ВЫЧИСЛЕНИЯ ЦЕЛОЧИСЛЕННОГО СРЕДНЕГО
// ЗНАЧЕНИЯ 2-Х ЦЕЛЫХ ЧИСЕЛ:
int average2( int numb1, int numb2);
```

Генератор случайных значений C++

- Функция `rand()` генерирует числа в диапазоне от 0 до `RAND_MAX`.
 - `RAND_MAX` - это константа, определённая в библиотеке `<cstdlib>`.
- Функция `srand` выполняет инициализацию генератора случайных чисел `rand`.

```
#include <cstdlib>
```

```
#include <ctime>
```

```
srand(time(0));
```

```
cout<<rand();
```

Примеры

- Какие значения будут сгенерированы:
 - `rand() % 2`
 - `rand() % 101`
 - `rand() % 201`
 - `rand() % 201 + 200`
- Написать выражение для получения целых чисел в диапазоне от -100 до 100.
- Получить числа на интервале от 0.01 до 1.

```
#include <iostream>
```

```
#include <cstdlib>
```

```
#include <ctime>
```

```
void main()
```

```
{
```

```
    srand(time(0));
```

```
    cout<<"1 " <<rand()<<endl;
```

```
    cout<<"2 " <<rand() % 2<<endl;
```

```
    cout<<"3 " <<rand() % 101<<endl;
```

```
    cout<<"4" <<rand() % 201<<endl;
```

```
    cout<<"5 " <<rand() % 201 + 200<<endl;
```

```
}
```

C:\WINDOWS\system32\cmd.exe

1 11481

2 1

3 20

4 135

5 279

Для продолжения нажмите любую клавишу . . .

C:\WINDOWS\system32\cmd.exe

1 11589

2 1

3 16

4 155

5 312

Для продолжения нажмите любую клавишу . . .

C:\WINDOWS\system32\cmd.exe

1 11844

2 0

3 53

4 71

5 327

Для продолжения нажмите любую клавишу . . .

```
#include <iostream>
```

```
#include <cstdlib>
```

```
#include <ctime>
```

```
void main()
```

```
{
```

```
//srand(time(0));
```

```
cout<<"1 " <<rand()<<endl;
```

```
cout<<"2 " <<rand() % 2<<endl;
```

```
cout<<"3 " <<rand() % 101<<endl;
```

```
cout<<"4" <<rand() % 201<<endl;
```

```
cout<<"5 " <<rand() % 201 + 200<<endl;
```

```
}
```

C:\WINDOWS\system32\cmd.exe

1 41

2 1

3 72

4169

5 274

Для продолжения нажмите любую клавишу . . .

C:\WINDOWS\system32\cmd.exe

1 41

2 1

3 72

4169

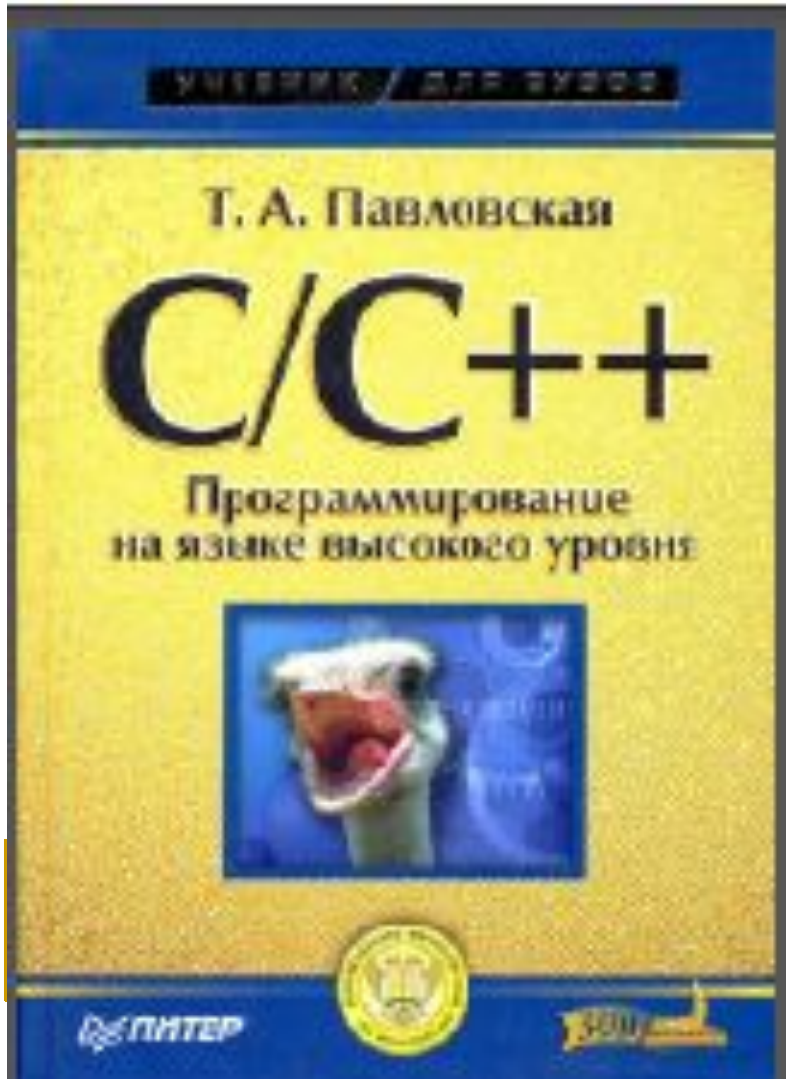
5 274

Для продолжения нажмите любую клавишу . . .

Задание

- Разработайте набор функций для выполнения действий с обыкновенными дробями.
- Все эти функции должны возвращать результат в виде обыкновенной несократимой дроби (не обязательно правильной).
- Например, функция, вычисляющая сумму обыкновенных дробей на вход получает числитель и знаменатель каждой из двух складываемых дробей и в результате должна получить числитель и знаменатель дроби, являющейся суммой двух исходных.
- Если в результате получается отрицательное значение, то знак «минус» хранить в числителе.
- Набор требуемых функций необходимо определить самостоятельно, исходя из условия задачи. Вывод результата осуществлять в виде 1 строки (например, 12/45).
- Для вывода дроби так же использовать функцию.
- Весь набор функций оформить в виде отдельного программного модуля

Домашнее задание



1) Стр. 81

2) Стр. 83 - 85