

# Процедуры и функции

1. Понятие подпрограммы;
2. Пользовательские функции и процедуры;
3. Область видимости;
4. Реентерабельность;
5. Рекурсия.
6. Пример

# 1. Понятие подпрограммы

**Подпрограмма** – это именованная часть программы, представляющая собой некоторое собрание операторов, структурированных аналогично основной программе.

Подпрограммы не являются обязательными, но их наличие заметно облегчает работу программиста и увеличивает «ценность» кода.

Два типа подпрограмм: **процедуры** и **функции**

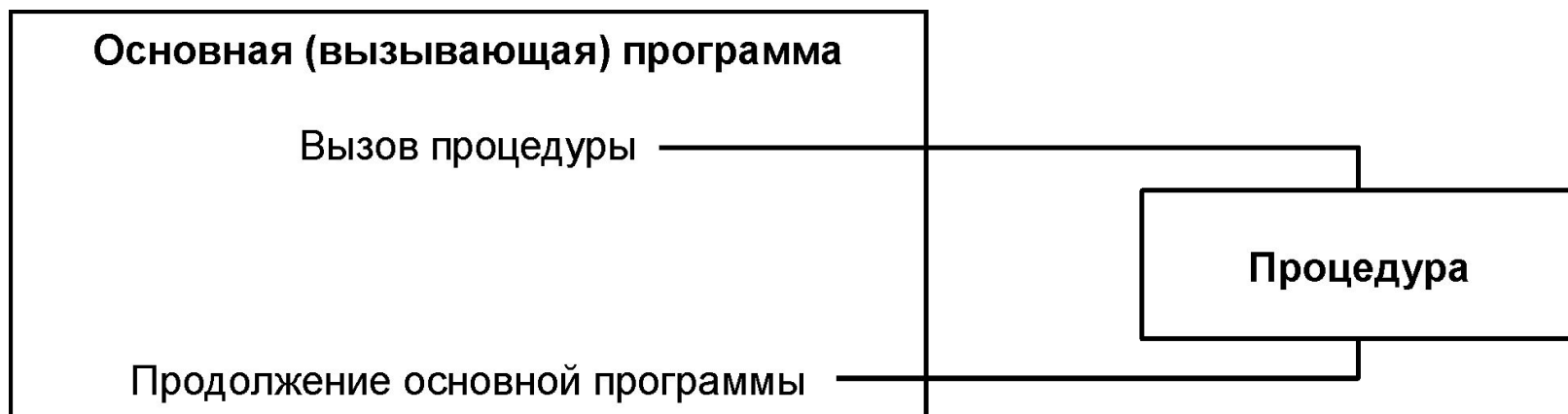
Для **функции** указывается тип возвращаемых данных, и обязательно должен присутствовать оператор **return**. Если возвращать значение не нужно, то у подпрограммы указывается тип **void (процедура)**.

# Процедуры и функции

1. Понятие подпрограммы;
2. **Пользовательские функции и процедуры;**
3. Область видимости;
4. Параметр-значение, параметр-переменная;
5. Реентерабельность;
6. Рекурсия.
7. Пример

## 2. Пользовательские функции и процедуры

**Процедурой** называется особым образом оформленный фрагмент программы, имеющий собственное имя. Упоминание этого имени в тексте программы приводит к активизации процедуры и называется её **вызовом**.



## 2. Пользовательские функции и процедуры

### Объявление функции

```
<тип данных, который будет возвращаться функцией> <имя>  
(<аргументы функции>) {  
    < блок  
        кода >  
}
```

**Тип данных функции**, который в конечном итоге будет передавать функция.

**Имя функции**, нельзя использовать зарезервированные слова в C++, имена начинающиеся с цифр, а также имена разделенные пробелом.

*Лучше всего задавать «КРИЧАЩЕЕ» имя, которое будет говорить за что отвечает эта функция.*

## 2. Пользовательские функции и процедуры

В C++ **нельзя** вызвать функцию до объявления самой функции. Все потому, что компилятор не будет знать полное имя функции (имя функции, число аргументов, типы аргументов). Таким образом в примере ниже компилятор сообщит нам об ошибке:

```
int main() {
    Summ(1, 2);

    return 0;
}

void Summ(int a, int b) {
    cout << a + b;
}
```

## 2. Пользовательские функции и процедуры

**Прототип функции** — это функция, в которой отсутствует блок кода (тело функции). В прототипе функции находятся:

Полное имя функции.

Тип возвращаемого значения функции.

```
int Sum_numbers(int a, int b); // прототип функции
```

```
int main() {  
    Sum_numbers(1, 2);  
  
    return 0;  
}
```

```
int Sum_numbers(int a, int b) { // сама функция  
    cout << a + b;  
}
```

## 2. Пользовательские функции и процедуры

**Пример.** Разработать функцию возведение числа в степень.

$$a^x = e^{x \ln a}$$

```
#include <iostream>
#define _USE_MATH_DEFINES
#include <cmath>
using namespace std;

double myPower (double a, double x)
{
    double y = 0.0;
    if (a > 0){
        y = exp(x*log(a));
    }
    return y;
}

int main() {
double a,x;
cout << "Введите основание и степень" << endl;
cin >> a >> x;
cout << myPower(a, x);
}
```



## 2. Пользовательские функции и процедуры

**Пример.** Разработать функцию возведение числа в степень.

$$a^x = e^{x \ln a}$$

```
#include <iostream>
#define _USE_MATH_DEFINES
#include <cmath>
using namespace std;
double myPower (double a, double x); //прототип функции

int main() {
double a,x;
cout << "Введите основание и степень" << endl;
cin >> a >> x;
cout << myPower(a, x);
}
double myPower (double a, double x)
{
    double y = 0.0;
    if (a > 0){
        y = exp(x*log(a));
    }
    return y;
}
```

## 2. Пользовательские функции и процедуры

**Пример.** Вывести на экран результат возведение числа в степень.

$$a^x = e^{x \ln a}$$

```
#include <iostream>
#define _USE_MATH_DEFINES
#include <cmath>
using namespace std;

void myPower (double a, double x)
{
    double y = 0.0;
    if (a > 0){
        y = exp(x*log(a));
    }
    cout >> y;
}

int main() {
double a,x;
cout << "Введите основание и степень" << endl;
cin >> a >> x;
myPower(a, x);
}
```

## 2. Пользовательские функции и процедуры

**Пример.** Разработать функцию, которая выводит на экран прямоугольник из заданных символов определенного размера.

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

void myBox (int w, int h, char s)
{
    for (int i = 1; i <= h; i++)
        {for (int j = 1; j <= w; j++)
            {
                cout << s;
            }
            cout << endl;
        }
}

int main() {
    myBox (20,4,'+');
    myBox (10,5,'=');
    myBox (11,6,'O');
}
```

```
+++++
+++++
+++++
+++++
=====
=====
=====
=====
=====
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
```

# Процедуры и функции

1. Понятие подпрограммы;
2. Пользовательские функции и процедуры;
3. **Область видимости;**
4. Реентерабельность;
5. Рекурсия.
6. Пример

## 3. Область видимости

**Локальные переменные** — это переменные созданные в блоках. Областью видимости таких переменных является блоки ( и все их дочерние ), а также их область видимости не распространяется на другие блоки.

**Глобальные переменные** имеют статическую продолжительность жизни, т.е. создаются при запуске программы и уничтожаются при её завершении. Глобальные переменные имеют глобальную область видимости (или «файловую область видимости»), т.е. их можно использовать в любом месте файла, после их объявления.

**Глобальные переменные уступают локальным.**

### 3. Область видимости

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

int main() {

    int a = 1;
    { // блок 1
        int b = 2;
        // здесь доступны локальная b и a из внешнего блока
        { // блок 1.1
            int c = 3;
            // здесь доступны локальная c и a, b из внешнего блока
        }
        { // блок 1.2
            int d = 3;
            // здесь доступны локальная d и a, b из внешнего блока.
        }
        //Переменная c здесь не доступна
    }
}
```

### 3. Область видимости

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

int main() {
    int a = 0;
    cout << "Введите значение переменной a = ";
    cin >> a ;

    if (a = 10) {
        int b = 1;

    }
    else {
        int b = 0;
    }
    cout << b;
}
```

Сообщение

In function 'int main()':

[Error] 'b' was not declared in this scope

recipe for target 'main.o' failed

Переменная b локальная

### 3. Область видимости

```
#include <iostream>
using namespace std;
int a = 100;

void PrintVal_1() {
    cout << "PrintVal_1" << endl;
    cout << a << endl;
}

void PrintVal_2() {
    int a = 200;
    cout << "PrintVal_2" << endl;
    cout << a << endl;
}

int main() {
    int b = 300;
    PrintVal_1();
    PrintVal_2();
}
```

```
PrintVal_1
100
PrintVal_2
200
```



## 3. Область видимости

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

```
int a = 1;
int b = 1;
```

```
int calc(int c) {
    int a = 3;
    return a*10 + c;
}
```

```
int main() {
    a = 4;
    cout << calc(a);
}
```

### 3. Область видимости

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

```
int a = 1;  
int b = 1;
```

```
int calc(int c) {  
    a = 6;  
    return a*10 + c;  
}
```

```
int main() {  
    a = 4;  
    cout << calc(a);  
}
```

### 3. Область видимости

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

```
int a = 1;  
int b = 1;
```

```
int calc(int c) {  
    return a*10 + c;  
}
```

```
int main() {  
    a = 3;  
    cout << calc(a);  
}
```

# Процедуры и функции

1. Понятие подпрограммы;
2. Пользовательские функции и процедуры;
3. Область видимости;
4. **Реентерабельность;**
5. Рекурсия.
6. Пример

## 4. Реентерабельность

**Реентерабельная, или повторно входимая функция** - это функция, которая может быть использована более чем одной задачей без риска потери данных.

Реентерабельная функция может быть в любое время прервана и продолжена позже без потерь данных. Реентерабельные функции либо используют локальные переменные, либо защищают свои данные, размещённые в глобальных переменных.

**Функция fn не является реентерабельной:**

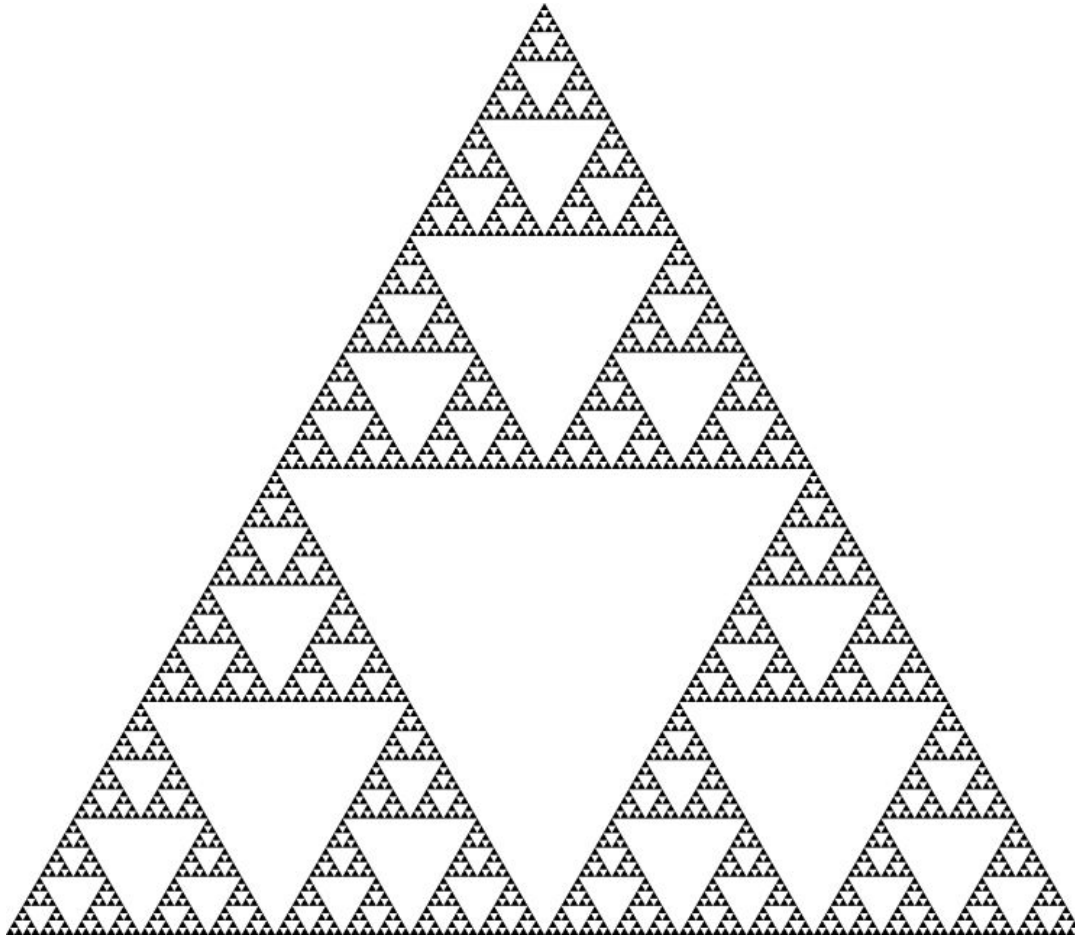
```
var
temp : integer;
function fn (x,y : integer):integer;
begin
  temp := x;
  x := y;
  y := temp;
  fn := x + y;
end;
```

# Процедуры и функции

1. Понятие подпрограммы;
2. Пользовательские функции и процедуры;
3. Область видимости;
4. Реентерабельность;
5. **Рекурсия.**
6. Пример

# 5. Рекурсия

**Рекурсия** — в определении, описании, изображении какого-либо объекта или процесса внутри самого этого объекта или процесса, то есть ситуация, когда объект является частью самого себя



## 5. Рекурсия

**Рекурсия** – это такой способ организации вычислительного процесса, при котором подпрограмма в ходе выполнения составляющих её операторов обращается сама к себе



# 5. Рекурсия

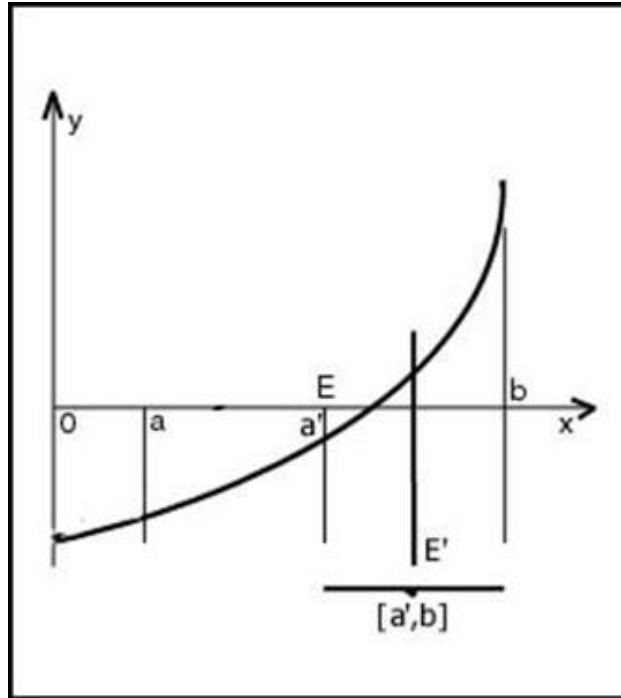
```
#include <iostream>
using namespace std;

unsigned long int fact(unsigned long int f)
{
    unsigned long int result;
    if (f == 1 || f == 0) {return 1; }
    else { result = f * fact(f - 1); }
    return result;
}

int main()
{
    int n;
    cout << "Enter n!: ";
    cin >> n;
    cout << n << "!" << "=" << fact(n) << endl;
    system("pause");
    return 0;
}
```

# 5. Рекурсия

**Пример:** Реализовать метод половинного деления на основе рекурсии



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

```
double myFunc(double x) {
return x*x*x + 8;
}
```

# 5. Рекурсия

```
double FindRoot(double a, double b) {
double eps = 0.1; //ТОЧНОСТЬ
double c = (a + b)/2;
if ( abs(c - a) > eps)
{   if (myFunc(a) * myFunc(c) < 0)
        { return FindRoot (a,c);}
    else
        { return FindRoot (c, b);}
}
else return c;
}
int main() {
    double a = -3;
    double b = 2;
    cout << "Корень уравнения  $x^3+x+8=0$  на отрезке [" << a << ";" << b << "] равен "<<
    FindRoot(a, b);

}
```