



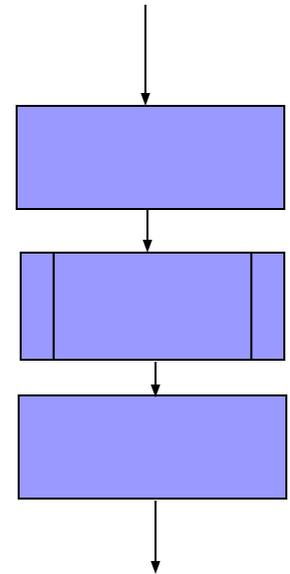
# Язык С

## **Лекция №4**

### **Подпрограммы**

# Назначение подпрограмм

- Экономия кода
- Структуризация программы



# Два вида подпрограмм

- Функция – возвращает значение через свое имя
  - `a=sqrt(x);`
- Процедура – выполняет какое-то действие
  - `printf("Hello");`

В языке C/C++ нет процедур, но есть функции типа **void**.

# Пример: возведение в степень

```
double power(double a, int n)
{ int i;
  double x=1;
  for (i=0; i<abs(n); i++) x*=a;
  if (n<0) x=1.0/x;
  return x;
}
```

Заголовок  
функции

Формальные  
параметры  
функции

Описание  
функции

Вызов  
функции

```
main ()
{ int i;
  for (i=-10; i<=10; i++)
    printf ("%lf\n", power(2, i) );
}
```

Фактические  
параметры

# Формальные и фактические параметры

- Параметры передаются по значению, т.е. создаются локальные переменные, соответствующие формальным параметрам, и в них копируются значения фактических параметров
- Типы и число фактических параметров должны соответствовать формальным параметрам
- Для проверки этого соответствия функция должна быть либо описана ранее своего использования, либо до использования функции должен быть описан ее заголовок, обычно находящийся во включаемом файле

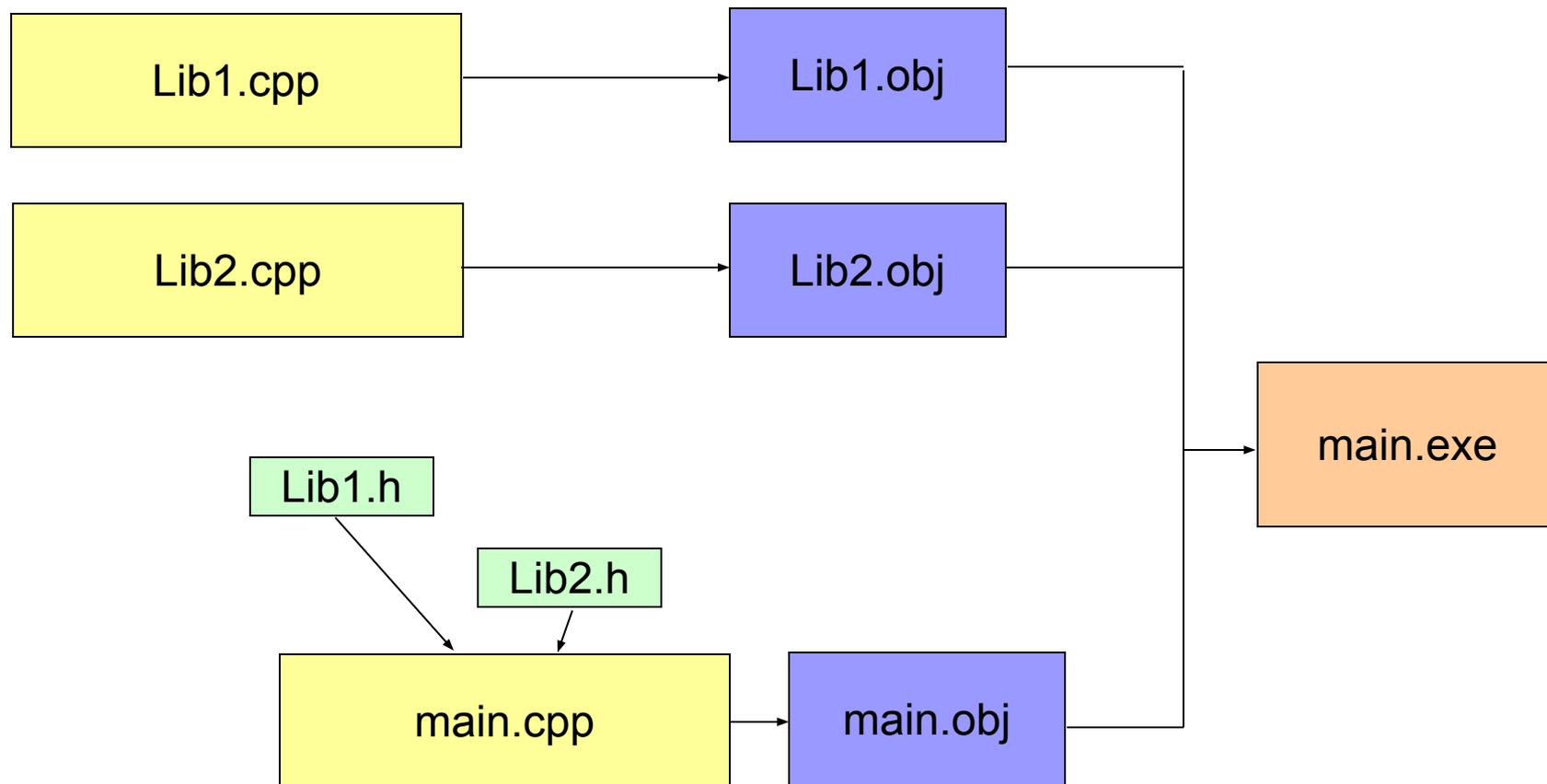
# Заголовок функции

```
double power(double a, int n);
```

```
main()  
{ int i;  
  for (i=-10; i<=10; i++)  
    printf("%lf\n", power(2, i) );  
}
```

```
double power(double a, int n);  
{ int i;  
  double x=1;  
  for (i=0; i<abs(n); i++) x*=a;  
  if (n<0) x=1.0/x;  
  return x;  
}
```

# Создание библиотек функций



# Файл myfunc.c

```
#include <math.h>
```

```
#include "myfunc.h"
```

```
double power(double a, int n)
```

```
{
```

```
    int i;
```

```
    double x=1;
```

```
    for (i=0; i<abs(n); i++) x*=a;
```

```
    if (n<0) x=1.0/x;
```

```
    return x;
```

```
}
```

# Файл myfunc.h

```
double power(double, int) ;
```

# Файл prog.c

```
#include <stdio.h>
```

```
#include "myfunc.h"
```

```
main()
```

```
{ int i;
```

```
  for (i=-10; i<=10; i++)
```

```
    printf("%1f\n", power(2,i) );
```

```
}
```

# Область видимости имен

```
int m = 10; // Глобальная переменная
```

```
int f(int n)
{ int i;      // Локальная переменная
  int k=0;    // Инициализация каждый вызов
  char m='a'; // Перекрытие глобального имени
  return n;   // Параметр – тоже локальная переменная
}
```

```
main()
{ int i;      // Локальная переменная
  i = m;     // Обращение к глобальной переменной
  i = f(i);  // Вызов функции
}
```

# Область видимости внутри модуля

```
main ()
{  int i;
   i=0;
   {  int i, j;
      i=5;
   }
   j=0; // здесь ошибка
   i++; // здесь i станет 1
}
```

# Область видимости внутри модуля

```
main ()
{  int i;
   i=0;
   for (int k=0; k<10; k++)
   {
       printf ("%d\n",k) ;
   }
   k=0; // здесь ошибка
   i++; // здесь i станет 1
}
```

# void-функции

```
void prt(int n)
{
    printf("%10d\n", n);
    return;
}
```

```
main()
{ int i;
  for (i=0, i<10; i++) prt(i*i);
}
```

# Встраиваемые функции

```
inline double sqr(double x)
{ return x*x;
}
```

*Помещать лучше в заголовочный файл*

# Классы памяти переменных

- **auto** – локальные переменные создаются при входе и уничтожаются при выходе из функции
- **static** – переменные создаются при компиляции программы и «живут» всё время выполнения программы
- **extern** – переменная описана «где-то в другом месте» как глобальная
- **register** – переменную следует хранить в регистре процессора, не может иметь адреса

## Область видимости и классы памяти

Файл m.c

```
int i=0;
void f1 ();

main ()
{
    f1 (); f1 ();
}
```

Файл a.c

```
#include <stdio.h>

void f1 ()
{
    extern int i;
    static int n=i;
    printf ("%d\n", n++);
}
```

# Способы передачи аргументов в С

## // Передача по значению

```
void swap(int a, int b)
{ int tmp=a;
  a=b;
  b=tmp;
}
```

swap (x, y) – не работает!

## // Передача по адресу

```
void swap (int *a, int *b)
{ int tmp = *a;
  *a = *b;
  *b = tmp;
}
```

swap (&x, &y) – работает!

# Передача массивов

Варианты заголовков функций

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

# Передача массивов

```
#include <stdio.h>
#define N 10

int sum(int a[], int n)
{
    int b = 0;
    for (int i=0; i<n; i++) b+=a[i];
    return b;
}

int main()
{
    int m[N];
    for (int i=0; i<N; i++) m[i]=i+1;
    printf("%d\n", sum(m, N)); // m - уже указатель
}
```

# Функция сортировки массива

```
#include <stdio.h>
#define N 10

void sort(int a[], int n)
{
    for (int i=0; i<n-1; i++)
        for (int j=i+1; j<n; j++)
            if (a[i]>a[j]) { int x=a[j]; a[j]=a[i]; a[i]=x; }
}

int main()
{
    int m[N];
    for (int i=0; i<N; i++) m[i]=10-i;
    sort(m,N);
    for (int i=0; i<N; i++) printf("%4d",m[i]);
}
```

# Задача: бинарный поиск

```
int findbin(int x, int *a, int n)
{ int left, right;
  left=0; right=n-1;
  while (right>left)
  { k=(right+left)/2;
    if (a[k]==x) return k;
    else if (x>a[k]) left=k+1;
    else right=k;
  }
  return -1;
}
```

# Передача строк

```
char str[12] = "Borland C++";
```

B	o	r	l	a	n	d		C	+	+	\0
---	---	---	---	---	---	---	--	---	---	---	----

```
int strlen(char *s);  
{ int k=0;  
  while (*s++) k++;  
  return k;  
}
```

```
printf("%d\n", strlen("abc"));
```

# Задача: поиск символа в строке

```
int findc(char c, char *s)
{ int n=0;
  while (*s)
  { if (c==*s++) return n;
    else n++;
  }
  return -1;
}
```

# Передача функций как параметров

```
double deriv(double f(double), double x, double eps)
{
    return (f(x+eps*0.5)-f(x-eps*0.5))/eps;
}
```

```
double g(double x)
{
    return x*x;
}
```

```
int main()
{
    printf("%1f", deriv(g, 2.0, 1E-6));
}
```

# Рекурсивные функции

```
fact(int n)
{ int a;
  if (n<=1) return 1;
  a = fact(n-1)*n;
  return a;
}
```

# Рекурсивные функции

```
fib (int n)
{ if (n<=2) return 1;
  else return fib(n-1)+fib(n-2);
}
```

```
main ()
{
  printf ("%d\n", fib(5));
}
```

# Функции с переменным числом параметров

```
void f(int a, int b, ...)
```

Тип        `va_list`

Макрос    `va_start(list, last_fixed)`

Макрос    `va_arg(list, arg_type)`

Макрос    `va_end`

# 1-й способ: последний параметр – 0

```
int sum1(int a, ...)  
{ va_list args;  
  int result=a, t;  
  va_start(args,a);  
  while(t=va_arg(args,int))  
    result+=t;  
  va_end(args);  
  return result;  
}
```

## 2-й способ: первый параметр – число аргументов

```
int sum2(int num, ...)  
{ va_list args;  
  int result=0, i;  
  va_start(args,num);  
  for (i=0; i<num; i++)  
      result+= va_arg(args,int);  
  va_end(args);  
  return result;  
}
```

### 3-й способ: первый параметр – строка формата

```
#include <stdarg.h>
#include <stdio.h>

void errmsg(char *fmt, ...)
{ va_list args;
  printf("Error:");
  va_start(args, 0);
  vprintf(fmt, args);
  va_end(args);
  return;
}

main()
{ errmsg("%s - (%s)\n", "First", "Second");
}
```