Лекция 9.
Функции в языке Си

```
<тип_результата> <имя_функции>(<список_формальных_параметров>){
        <операторы>
        return <результат>;
}
```

После указания типа результата может быть дополнительно указано используемое соглашение о вызовах: *__cdecl* (по умолчанию, параметры вычисляются справа налево), *__stdcall* (обычно в *DLL*), *__fastcall* (с использованием регистров *ECX* и *EDX* для передачи двух первых слева параметров подходящего размера).

return – это не только запоминание результата , который надо вернуть, но и *выход* из функции.
Если <тип_результата> *void*, то значение результата указывать не надо. Просто: *return;*

Например, вычислим для целых чисел ***c= a+b***

***c* – возвращаемое значение в точку вызова функции**	
Кратко	Аналог на *Delphi*

Способ1
int func1(int a, int b){
return a+b;
}
 	
Вызов: c=func1(a,b);
	Function func1(a, b: integer):integer;
	Begin
	func1:=a+b; exit;
	End;
	
	Вызов: c:=func1(a,b);

Подробно	Аналог на *Delphi*

Способ2
int func2(int a, int b){
int res;
res=a+b;
return res;
}
Вызов: c=func2(a,b);
	Function func2(a, b: integer):integer;
	Var res: integer;
	Begin
	res:=a+b;
	func2:=res; exit;
	End;
	Вызов: c:=func2(a,b);

Изменяем *c* через параметр	
Передаем указатель (*адрес*) для изменения значения по этому адресу	Аналог на *Delphi* – явная передача указателя (Type **PInteger** = ^Integer;)

Способ3 (указатель)
void func3(int a, int b, **int*** c){
int res;
res=a+b;
***c**=res;
return;
}
Вызов: func3(a,b,**&c**); например, scanf("%d", &n);
	Procedure func3(a,b: integer; c: **PInteger**);
	Var res: integer;
	Begin
	res:=a+b;
	c^:=res; exit;
	End;
	
	Вызов: func3(a,b,**@c**); // var a,b,c:integer;

Передача *по ссылке* (только в C++)	Аналог на *Delphi* – передача параметра по ссылке

Способ4 (ссылка)
void func4(int a, int b, int **&**c){
c=a+b;
return;
}
Вызов: func4(a,b,c);
ЖЕЛАТЕЛЬНО ЗНАТЬ ОБА СПОСОБА (явной (№3) и косвенной (№4) передачи адреса параметра) – способ №3 используется в большинстве библиотечных функций
	Procedure func4(a,b: integer; **var** c: integer);
	Begin
	c:=a+b;
	exit;
	End;
	
	Вызов: func4(a,b,c);

Пример из Лабораторной работы №5 с разными способами передачи параметров:

```
#include <stdio.h>
#include <conio.h> // getch();
#include <math.h>
#include <stdlib.h>
#include <windows.h> // SetConsoleOutputCP(1251); SetConsoleCP(1251);
// #include <time.h>  // для установки srand по текущему времени
//---------------------------------------------------------------------------
const double xx[7] = {0.00001, -0.99, -1, -0.1, 0.1, 1, 0.99};

// первый способ - явная передача значений указателей - адресов n, x и e
void inputNX(int *n, double **x, double *e){
	printf("Введите e=? ");
	scanf("%lf", e); // e - это уже адрес, амперсанд не нужен
	if (*e<1e-13 || *e>0.11) { // сравниваем значение по адресу e
		printf("Некорректная точность e (0..0.1] \nPress any key");
		getch();
		exit(1); // выход из программы, а не только функции, с ошибкой (значением) 1
//		вместо		return 1; // выход только из этой функции
	}
	fflush(stdin);
	printf("Введите n=? ");
	scanf("%d", n);  // n - это уже адрес, амперсанд не нужен
	if (*n<1 || *n>20) {  // сравниваем значение по адресу n
		printf("Invalid n [1..20]! \nPress any key");
		getch();
		exit(2); // выход из программы, а не только функции, с ошибкой (значением) 2
//		вместо		return 2; // выход только из одной функции
	}
	fflush(stdin);

	*x = new double [*n]; // везде *x и *n, *e

	printf("Введите n=%d значений X из интервала (-1,+1):\n", *n);
	for (int i = 0; i < *n; i++) {
		scanf("%lf", &(*x)[i]);

		if (fabs((*x)[i])>=1) {
			(*x)[i]=xx[rand() % 7];
			if (fabs((*x)[i])==1) {
				(*x)[i]=(*x)[i]*(rand()%100) /100;
				if ((*x)[i]==0) (*x)[i]=*e;
			}
			printf("Некорректное значение заменено на %15.10lf\n", (*x)[i]);
		}
	}
	return;
}

// передача по ссылке
void inputNX2(int &n, double* (&x), double &e){
	printf("Введите e=? ");
	scanf("%lf", &e); // явная передача адреса e
	if (e<1e-13 || e>0.11) { // сравниваем значение e
```

```
                printf("Некорректная точность e (0..0.1] \nPress any key");
                getch();
                exit(1); // выход из программы, а не только функции, с ошибкой (значением) 1
//              вместо          return 1; // выход только из этой функции
        }
        fflush(stdin);
        printf("Введите n=? ");
        scanf("%d", &n);
        if (n<1 || n>20) {
                printf("Invalid n [1..20]! \nPress any key");
                getch();
                exit(2); // выход из программы, а  не только процедуры, с ошибкой (значением) 2
//              вместо          return 2;
        }
        fflush(stdin);

        x = new double [n];

        printf("Введите n=%d значений X из интервала (-1,+1):\n", n);
        for (int i = 0; i < n; i++) {
                scanf("%lf", &x[i]);

                if (fabs(x[i])>=1) {
                        x[i]=xx[rand() % 7];
                        if (fabs(x[i])==1) {
                                x[i]=x[i]*(rand()%100) /100;
                                if (x[i]==0) x[i]=e;
                        }
                        printf("Некорректное значение заменено на %15.10lf\n", x[i]);
                }
        }
        return; // конец функции
}

// сумма бесконечного ряда для одного x. последний параметр можно не передавать, тогда он=500
double endless(double x, double e, int *K, const int kmax=500){
                double sum, sl; // локальные переменные
                sum = sl = pow(x,2);
                int k=1; // k вместо *K - это удобнее, в конце добавлено обратное присваивание
                while ((fabs(sl)>=e) && (k!=kmax)){
                   //if (k<20) printf("%17.10lf", sl);
                        sl*=-pow(x,2)/(2*k)/(2*k+1);
                        sum+=sl;
                        k++;
                };
                *K=k; // чтобы изменить значение по адресу K(параметр)
                return sum; // конец функции со значением функции из переменной sum
}

int main()
{
        int n,i,k,z;
        double e, sl, sum, f, *x;
        SetConsoleOutputCP(1251); // вывод в кодировке 1251
```

```
	randomize();  // или  в Visual Studio srand(time(0));

	inputNX(&n,&x,&e); // передача по адресу. или можно вторым способом:
	// inputNX2(n,x,e);  // передача по ссылке в C++  как с var в Delphi

	z = ceil(fabs(log(e)/log(10.0)))+1;

	printf("e = %*.*lf\n", z+2, z, e);
	printf("N |       X       |      Sum(X)     | K|       F(X)      | |Sum(X)-F(X)|\n");
	for (i=1; i < 80; i++) printf("="); printf("\n");
	for (i = 0; i < n; i++) {
		sum = endless(x[i],e,&k); // передача k по адресу, возвращение sum как результата
		f=  x[i]*sin(x[i]);
		printf("%2d|%17.*lf|%17.*lf|%2d|%17.*lf|%17.*lf\n",
			i+1, z, x[i],  z, sum,  k,    z, f,   z+2, fabs(sum-f));
	}

	delete [] x;

	printf("Нажмите любую клавишу - Press any key");
	getch();
	return 0;
}
```

Прототипы функций и разделение на несколько файлов

Для возможности раздельной разработки и компиляции, для создания библиотек функций есть возможность создания многофайловых проектов.

Функции, выделяемые помимо главной функции приложения *main*, обычно располагают не до, а после *main*. Но, поскольку их описание должно быть сделано ДО использования, в начало файла помещают прототипы этих функций, например:

Функция ДО main	Функция ПОСЛЕ main	Комментарии ко 2 столбцу

#include <conio.h> // getch
#include <stdio.h> // printf, scanf
//----------
int sum(int a, int b){
return a+b;
}
//----------
void main(){
int a,b;
printf("a=? b=?");
scanf("%d %d", &a, &b);
 	
int c=sum(a,b);
printf("c=%d+%d=%d", a,b,c);
 	
printf("\nPress any key");
getch();
return;
}
//----------
	#include <conio.h> // getch
	#include <stdio.h> // printf, scanf
	//----------
	int sum(int, int);
	//----------
	void main(){
	int a,b;
	printf("a=? b=?");
	scanf("%d %d", &a, &b);
	
	int c=sum(a,b);
	printf("c=%d+%d=%d", a,b,c);
	
	printf("\nPress any key");
	getch();
	return;
	}
	//----------
	int sum(int a, int b){
	return a+b;
	}
		Подключение библиотек
		
		←Прототип *sum*
		(можно без имен параметров)
		
		←Главная функция приложения *main*
		
		←Вызов функции *sum* и объявление *c*
		
		←Функция *sum*

Описания прототипов и констант обычно выносят в **заголовочные файлы (*.h – header** (заголовок)), которые имеют **то же имя**, что и файл с полным описанием этих функций (*.cpp (C Plus Plus) или *.c), но **другое расширение**. Эти пары файлов (*.h и *.c/*.cpp) имеют смысл аналогичный разделам *INTERFACE* и *IMPLEMENTATION* в модулях (*Unit*) в языке *Delphi*.

Если вынести функцию *sum* из предыдущего примера в отдельный файл, получится код:

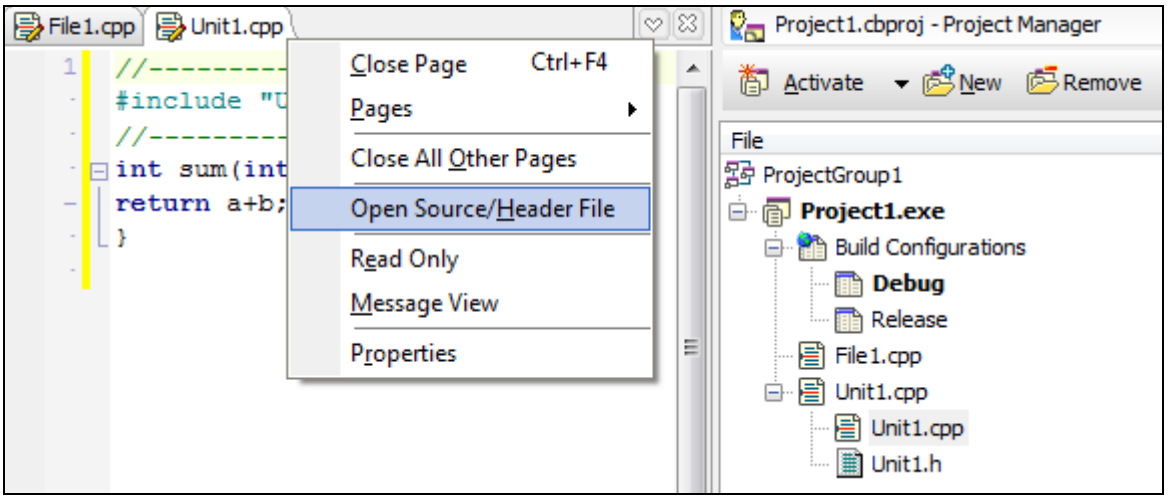
Заголовочный файл ***(Unit1.h):***	Файл с главной функцией *Main*	Комментарии ко второму столбцу

int sum(int, int);	#include <conio.h> // getch
	#include <stdio.h> // printf, scanf
	//----------
	#include "**Unit1.h**" // sum
	//----------
	void **main**(){
	int a,b;
	printf("a=? b=?");
	scanf("%d %d", &a, &b);
	
	int c=sum(a,b);
	printf("c=%d+%d=%d", a,b,c);
	
	printf("\nPress any key");
	getch();
	return;
	}
		←Подключение стандартных библиотек (в угловых скобках)

Файл с функцией ***(Unit1.cpp):***		←Подключение своей библиотеки (путь в двойных кавычках)

//если в заголовочном файле
//есть описания констант, типов,
//то его надо подключить и
//к этому файлу, как и другие
//необходимые модули:
//----------
#include "**Unit1.h**"
//----------
int **sum**(int a, int b){
return a+b;
}		

При создании модуля (New→ Unit C++ Builder) в среде Borland/CodeGear RAD Studio 2003-2009, как и в некоторых других средах сразу создается заготовка для двух файлов: заголовочного h-файла (h – header) и c/cpp-файла (C/ C Plus Plus), и дается возможность переключения межу ними:



После раздельной компиляции исходного кода всех модулей создаются объектные файлы, которые после редактирования связей компоновщиком преобразуются в исполнимый код программы в виде *exe*-файла (либо в *dll*-файл, например, в зависимости от исходно выбранного типа проекта).

Указатели

Нетипизированный указатель в *Delphi*	Аналог на языке С // #include <stdlib.h> или #include <alloc.h>

Var p: **pointer**;
Выделение памяти:
GetMem(p, 5*SizeOf(integer));
AllocMem(p, 5*SizeOf(integer)); // с обнул.
ReAllocMem(p, 10*SizeOf(integer));
Освобождение памяти:
FreeMem(p, 5*SizeOf(integer));
	**void ***p;
	Выделение памяти:
	p = malloc(5*sizeof(int));
	p = calloc(5, sizeof(double));//с обнулением
	p = realloc(p, 10*sizeof(int));// перевыдел.
	Освобождение памяти:
	free(p);

Типизированный указатель	Аналог на С++ // #include <alloc.h>

Var uk: ^**integer**;
Выделение памяти:
New(uk);
Освобождение памяти:
Dispose(uk);
Присваивание адреса другой переменной:
Если есть Var i: integer;
То можно uk:=@i; или uk:=Addr(i);
uk:=**nil**;
Разыменование указателя:
uk^:=1234567890;
	**int ***uk;
	Выделение памяти:
	uk = new int;
	Освобождение памяти:
	delete uk;
	Присваивание адреса другой переменной:
	Если есть int i;
	То можно uk=&i;
	uk = **NULL**;
	Разыменование указателя:
	*uk=1234567890;

К **типизированным** применимы и способы выделения (в паре с освобождением) для **нетипизированных**:	

AllocMem(uk, 5*SizeOf(integer));
AllocMem(uk, 5*SizeOf(integer));//с обнул.
ReAllocMem(uk, 10*SizeOf(integer));
FreeMem(uk, 5*SizeOf(integer));
	uk =(int*) calloc(5, sizeof(int));
	uk = (int*) malloc(5*sizeof(int));
	uk = (int*) realloc(p, 10, sizeof(int));
	free(uk);

Указатель на функцию

Указатель на функцию	Аналогом в Delphi является **процедурный тип** (см. Лекцию 15 первой части курса).
typedef double (***Tf**) (double); // тип	**Type Tfun** = function (x: integer): real; **Tproc** = procedure (x:real; var y: real);

Переменные типа указатель на функцию:
double (***fun**) (int); // без опис.типа выше
void (***proc**) (double, double&);
int (***fnc**) (int, const char*);
double (***f**) (double) = &**sin**;
y = **(*f**)(3.14); или y = **f**(x); // вызов **sin**
 	
Tf Mas[2] = {**sin**, cos};
double y = Mas[0](0.5); // вызов **sin**(0.5);
	Function **pervaya**(x: integer): real; // **Tfun**
	Begin {тело функции}End;
	
	Var fun: Tfun; y: real;
	proc: Tproc;
	Mas: array [1..2] of **Tfun**;
	begin
	fun:= pervaya; Mas[1]:=**pervaya**;
	y:=Mas[1](5); // вызов **pervaya**(5)

Пример с указателем на функцию – параметром другой функции:
Дано n – целое значение в градусах, написать функцию ***doit*** (n, <функция>), которая переводит градусы в радианы и вычисляет результат переданной функции *f* от вычисленных радиан. В качестве <функции> передать синус, косинус, и *f(x)=x*.

```
//---------------------------------------------------------------------------
#include <stdio.h>  // printf, scanf
#include <conio.h>  // getch
#include <math.h> // Математические функции (sin cos atan exp pow и т.д.) и константа Пи - M_PI
//---------------------------------------------------------------------------

double doit(int n, double (*f)(double)){ // второй параметр -  указатель на функцию, у которой
                                         один параметр типа double и результат тоже double
        return (*f)(n/180.0*M_PI);   // вызов функции (по переданному указателю f на нее)
                // и перевод n градусов в радианы перед передачей значения параметра

        // можно без (*):   return f(n/180.0*M_PI);
}

double my_f(double x){  // функция нужного типа (один параметр типа double и результат double)
        return x;
}

void main()
{
        int n; // целое
        double z1, z2, z3; // вещ.число с плавающей точкой двойной точности

        printf("n (grad -360..360)=?");
        scanf("%d", &n);  // ввод десятичного(%d) n в градусах


        z1=doit(n, &cos); // подставляем разные функции,(подходящие по типу параметра
                        // и результата) при вызове
        z2=doit(n, sin);  // можно без &
        z3=doit(n, my_f); // со своей функцией, описанной выше, тоже можно без &

        printf("cos(n)=%10.7lf sin(n)=%10.7lf  %d grad =%10.7lf rad\n", z1, z2, n, z3);
                        // %lf - LongFloat = double

        printf("\nPress any key to exit");
        getch(); // задержка консольного экрана до нажатия любой клавиши
        return; //выход из main()
}
//---------------------------------------------------------------------------
```

Такая возможность – передавать имя нужной функции в другую функцию – пригодится вам в Лабораторной работе **№8** для передачи каждой из двух заданных математических функций в каждую функцию поиска корня.