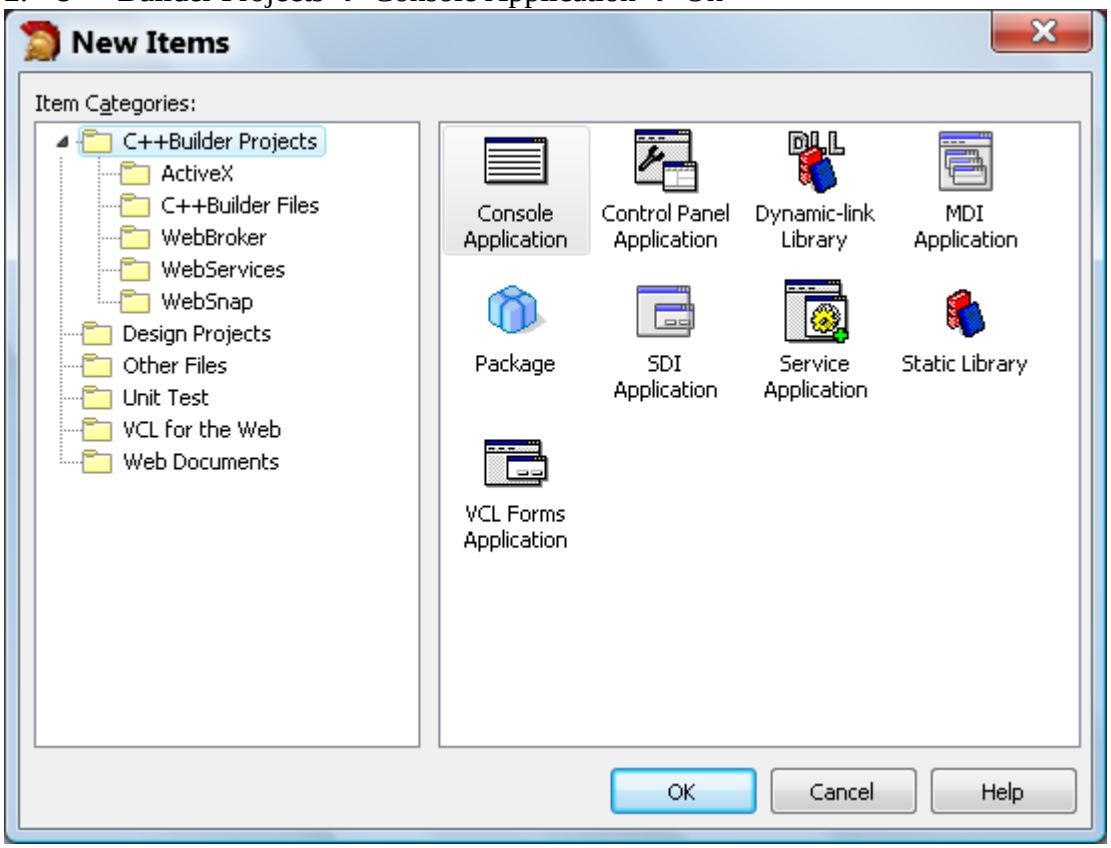
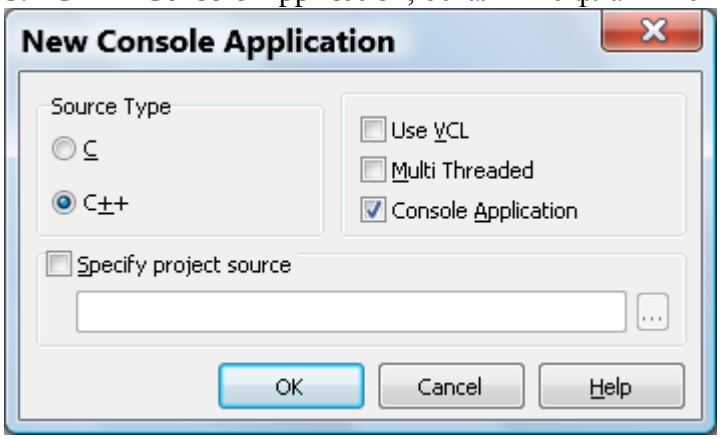
Лекция 7.
Введение в язык Си

Создание консольного приложения на C++

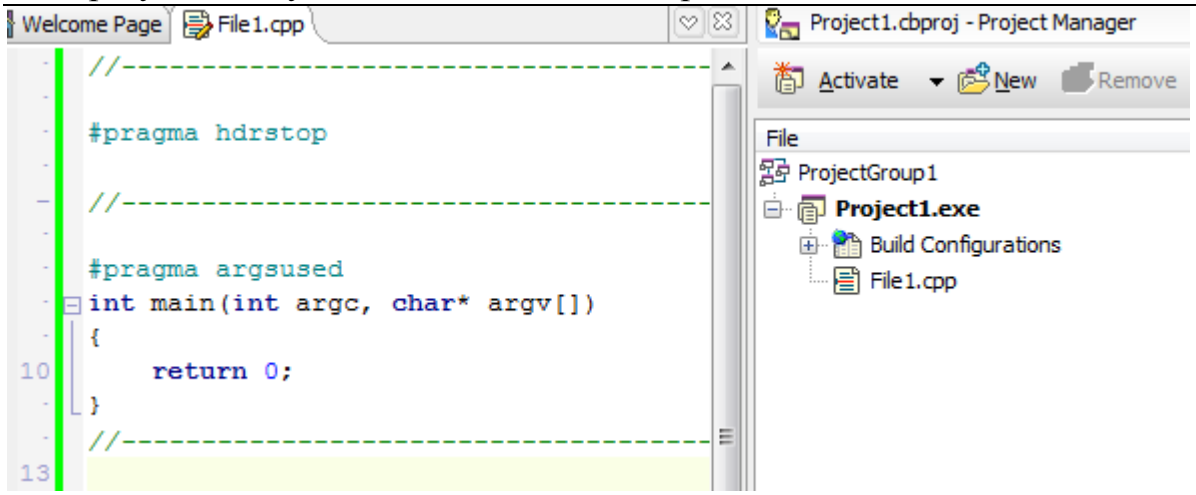
1. File → New → Other
2. C++ Builder Projects → Console Application → Ok



3. C++ и Console Application, остальные флажки снять:

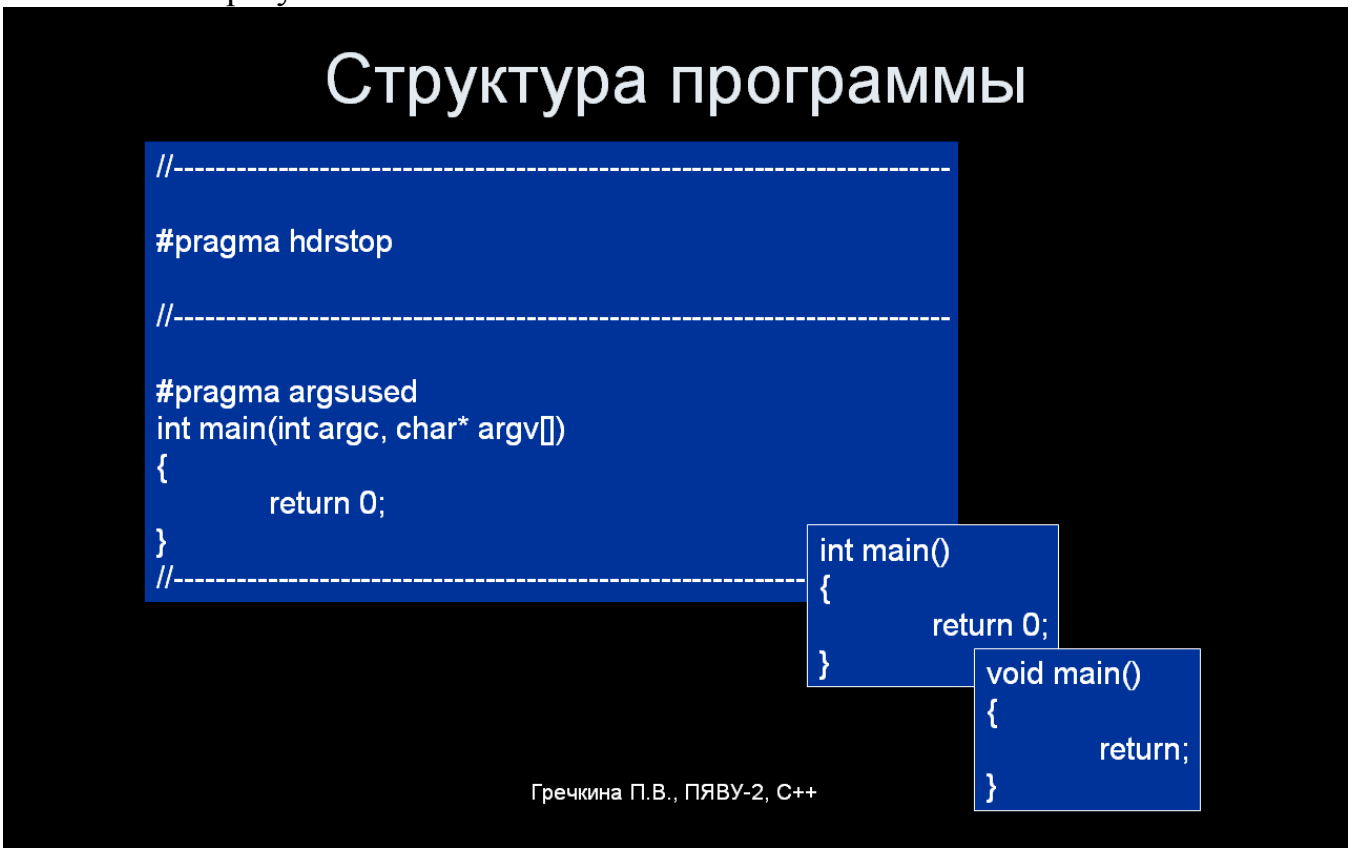


4. В результате будет создана заготовка приложения вида:

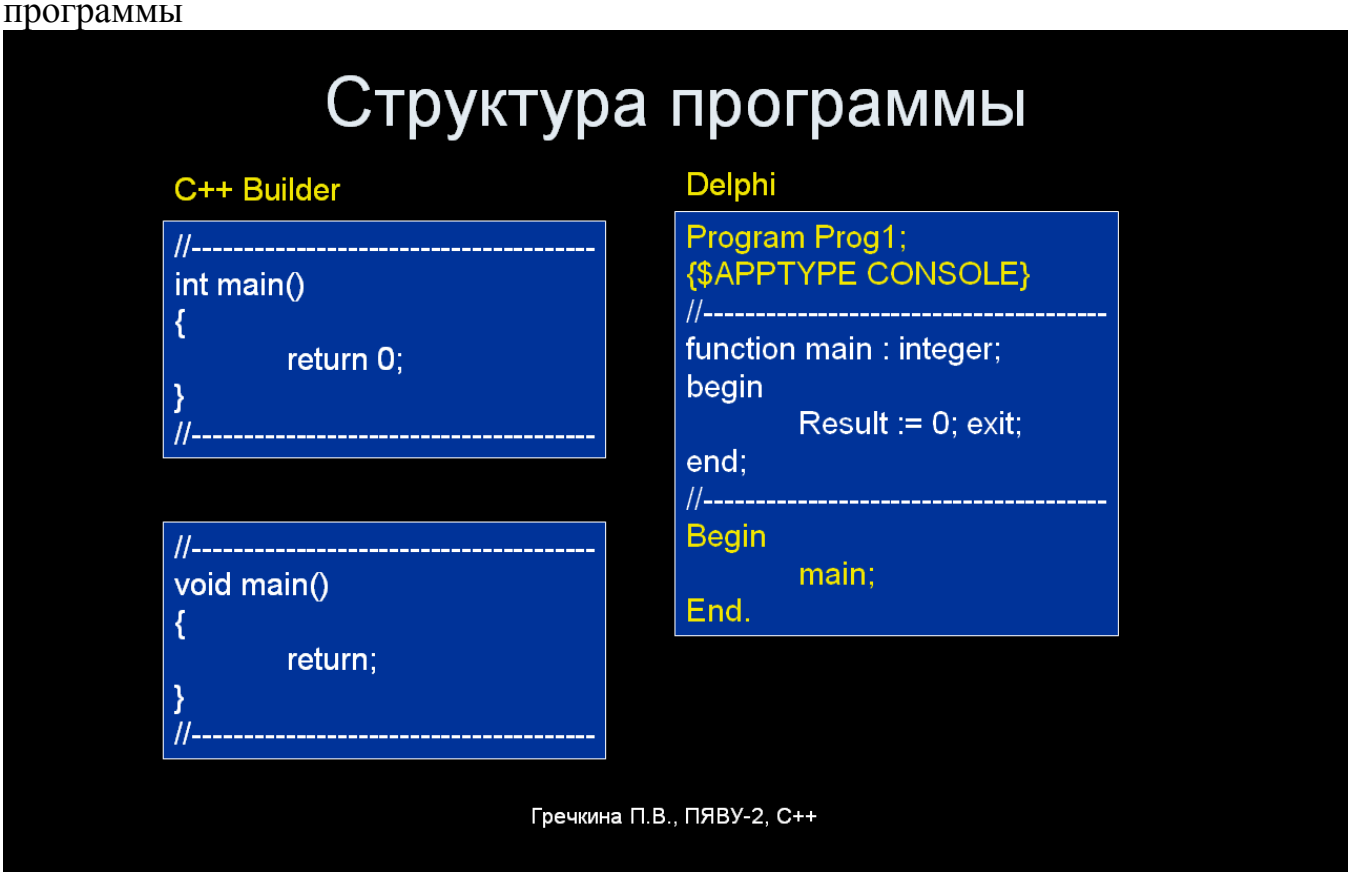


В зависимости от версии компилятора могут быть подключены некоторые модули – их не удалять. Вместо main могут быть немного другие названия, но некоторые из них говорят о том, что приложение создано НЕ правильно (см. предыдущие шаги)

Вид функции main можно изменить до более короткого, особенно, если пишите его сами, убрав параметры функции main и обе директивы компилятору #pragma, и изменить тип результата:



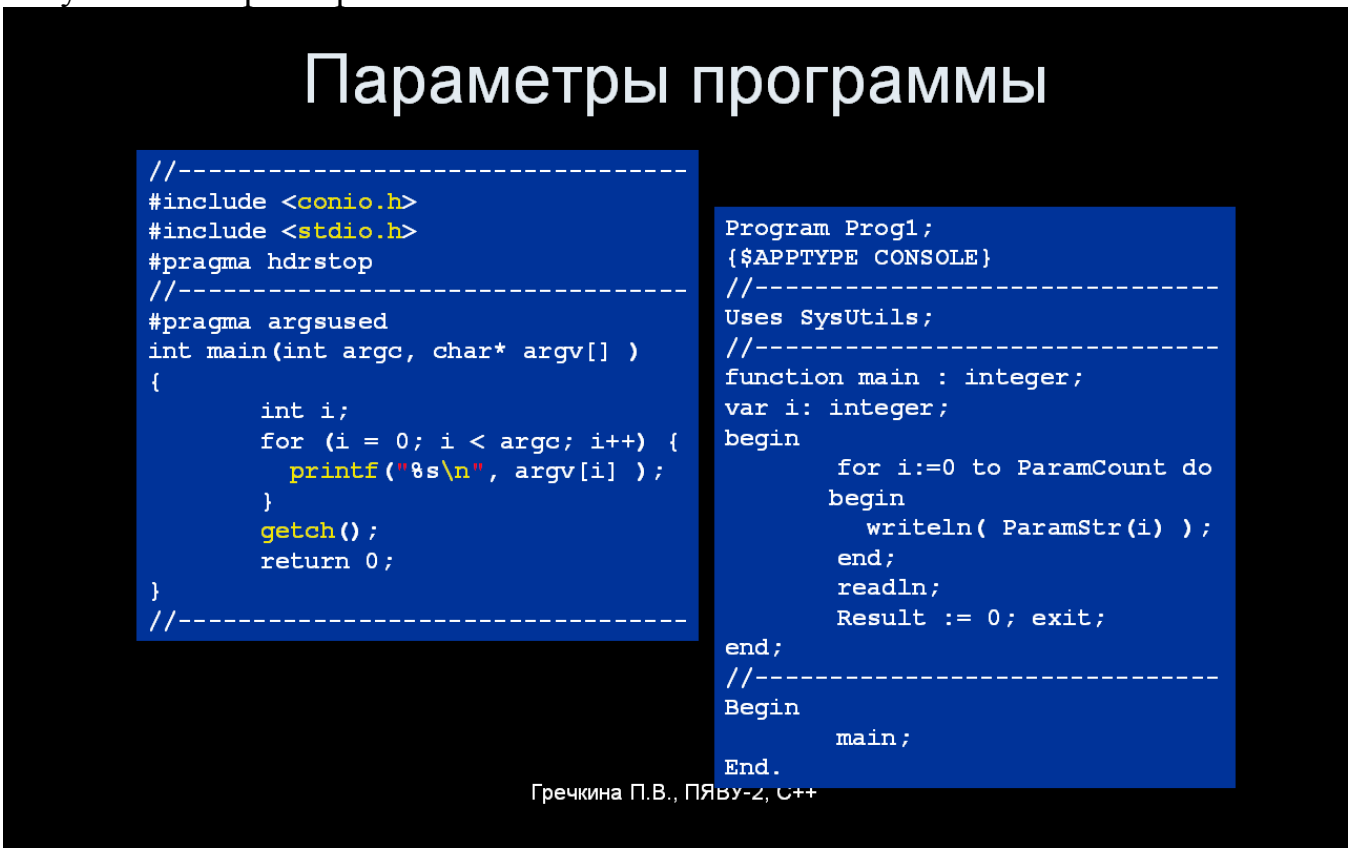
Аналогами первого варианта будет функция на *Delphi*, вызываемая из программы



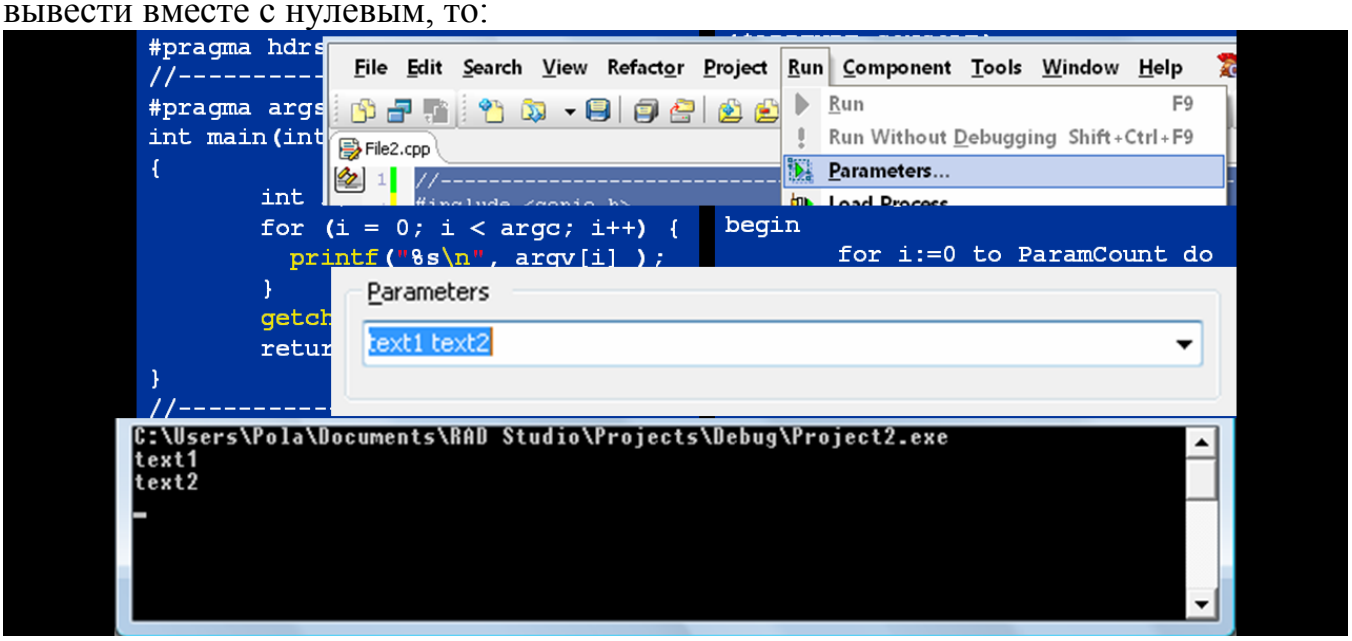
Аналогом второго варианта будет процедура в *Delphi*:

Поскольку вызов *main* – это единственное содержание программы (см, аналог на *Delphi*), то программа, вызывающая main в приложении на Си вовсе не пишется, хотя и есть соответствующий файл в проекте в среде *Borland Developer Studio* 2006.

Параметры функции *main* – это параметры программы, через которые в первом семестре надо было передавать имена файлов для разных тестов в программу. Их можно убрать, а можно воспользоваться. Пример вывода полученных параметров:



В *Delphi* и *C* есть разница в подсчете количества параметров. В *Delphi* считаются только пользовательские параметры, а обязательный нулевой (с именем запущенной программы с полным путем) не считается, а в языке *C* – он считается. Например, если передать два параметра (меню *Run* → *Parameters*), а потом их вывести вместе с нулевым, то:



в *C argc* будет равно 3 (параметры 0,1,2), а в *Delphi ParamCount*=2 (1,2)

Константы в С можно описать двумя разными способами:

Через директиву препроцессора *#define* (будет подставлена в код перед компиляцией) или с созданием константы с помощью ключевого слова const.

Описания констант и переменных могут быть глобальными, если разместить их ДО функции, либо локальными, если описать их в теле функции.

Опишем глобальные константы и локальные переменные разных простых типов:

Простые типы

```
//---------------------------------------------
#include <conio.h>
#include <stdio.h>
#pragma hdrstop
//---------------------------------------------
const CN = 33;      //  константы
#define DCN 35
//---------------------------------------------
int main() {
       int i=-2000000000;  unsigned int ui = 4000000000;    // 4 байта

       short sh=-32000; unsigned short ush = 64000;  // 2 байта

       char c=-128; unsigned char uc = 255;          // 1 байт

       float f=2.3; double d = -4.3e-5;      // с плавающей точкой

       char ch = 'A', ch2 = 49;       // символ  1 байт
```

Здесь описаны переменные целых типов трех размеров (4, 2 и 1 байт), беззнаковые (*unsigned*) и со знаком; двух вещественных типов (одинарной и двойной точности) и символ. Аналогичные описания в *Delphi* выглядели бы так:

Простые

```
Const
    CN=33;
    DCN =35;
Var
    i: integer = -2000000000;  // 4 байта
    ui: cardinal = 4000000000;
    sh: Smallint = -32000;  // 2 байта
    ush: Word = 64000;
    c: ShortInt = -128;  // 1 байт
    uc: Byte = 255;
    f: Single = 2.3; // float point - 4 байта
    d: Real = -4.3e-5; // float point 8  = double
    ch: char = 'A';
    ch2: char = #49; // «1»
```

```
//------------------
#include <conio.h>
#include <stdio.h>
#pragma hdrstop
//------------------
const CN = 33;
#define DCN 35
//------------------
int main() {
       int i=-2000000000;  unsigned int ui = 4000000000;    // 4 байта

       short sh=-32000; unsigned short ush = 64000;  // 2 байта

       char c=-128; unsigned char uc = 255;          // 1 байт

       float f=2.3; double d = -4.3e-5;      // с плавающей точкой

       char ch = 'A', ch2 = 49;       // символ  1 байт
```

Таблица 7.1. Сравнение типов в языках *Delphi (Object Pascal), C|C++*

Тип переменной	Object Pascal	C / C++	Visual Basic
8-битовое целое со знаком	ShortInt	char	Нет
8-битовое целое без знака	Byte	unsigned char	Нет
16-битовое целое со знаком	SmallInt	short	Short
16-битовое целое без знака	Word	unsigned short	Нет
32-битовое целое со знаком	Integer, Longint	int, long	Integer
32-битовое целое без знака	Cardinal, LongWord	unsigned long	Нет
64-битовое целое со знаком	Int64	__int64	Нет
4-байтовое вещественное	Single	float	Single
6-байтовое вещественное	Real48	Нет	Нет
8-байтовое вещественное	Double	double	Double
10-байтовое вещественное	Extended	long double	Нет
64-битовое денежное	currency	Нет	Currency
8-битовое дата/время	TDataTime	Нет	Data
16-байтовый вариант	Variant, OleVariant, TVarData	Variant*, OleVariant*	По умолча-нию
1-байтовый символ	Char	char	Нет
2-байтовый символ	WideChar	WCHAR	Нет
Строка фиксированной длины	ShortString	Нет	Нет
Динамическая строка	AnsiString	AnsiString*	$
Строка с завершающим нулевым символом	PChar	char*	Нет
Строка 2-байтовых символов с завершающим нулевым символом	PWideChar	LPCWSTR	Нет
Динамическая 2-байтовая строка	WideString	WideString*	Нет
1-байтовое булево	Boolean, ByteBool	(Любое 1-байтовое)	Нет
2-байтовое булево	WordBool	(Любое 2-байтовое)	Boolean
4-байтовое булево	BOOL, LongBool	BOOL	Нет

* Классы Borland C++ Builder для эмуляции соответствующих типов Object Pascal.

Вывод в языке сделан более сложно и управляемо, можно выводить одни и те же значения в разных форматах и системах счисления:

Таблица 7.2. Задание типа в форматной строке.

Символ	Тип	Формат вывода
c	int или wint_t	При использовании с функцией printf определяет однобайтовый символ, при использовании с функцией wprintf определяет расширенный символ.
C	int или wint_t	При использовании с функцией printf определяет расширенный символ, при использовании с функцией wprintf определяет однобайтовый символ.
d	int	Знаковое десятичное целое.
i	int	Знаковое десятичное целое.
o	int	Беззнаковое восьмеричное целое.
u	int	Беззнаковое десятичное целое.
x	int	Беззнаковое шестнадцатеричное целое с использованием символов «abcdef».
X	int	Беззнаковое шестнадцатеричное целое с использованием символов «ABCDEF».
e	double	Знаковое число в форме [–]d.dddd e[знак]ddd, где d есть одна десятичная цифра, dddd – одна или более десятичных цифр, ddd – три десятичные цифры and знак есть + или –.
E	double	Идентичен формату e, за исключением того, что символ E, а не e вводит экспоненту.
f lf	float double (long float)	Знаковое число в форме [–]dddd.dddd, где dddd есть одна или более десятичных цифр. Количество цифр перед десятичной точкой зависит от длины числа, а количество цифр после десятичной точки – от требуемой точности.
g	double	Знаковое число в формате f или e, в зависимости от того, какой формат более компактен для заданного значения и точности.
G	double	Идентичен формату g, за исключением того, что символ E, а не e вводит экспоненту.
s	string	При использовании с функцией printf задаёт строку однобайтовых символов, при использовании с функцией wprintf задаёт строку расширенных символов. Символы печатаются до достижения признака конца строки.
S	string	При использовании с функцией printf задаёт строку расширенных символов, при использовании с функцией wprintf задаёт строку однобайтовых символов. Символы печатаются до достижения признака конца строки.
p	pointer to void	Печатает адрес, заданный аргументом.

Управляющие \t– табуляция; \n – конец строки ; \r – перевод каретки;\\ – обратный слеш

Пример указания формата при выводе с помощью функции *printf*:

Управляющая последовательность Ширина и точность Список вывода

printf("\t Выводим на экран числа: %0*.*lf и число %5.2f", 8, 4, 2.5 , 1.4);

Обычные символы Спецификация формата

Подстановка аргументов будет производиться следующим образом:

printf("\t Выводим на экран числа: %0*.*lf и число %5.2f", 8, 4, 2.5 , 1.4);

printf("\t Выводим на экран число: %8.4lf и число %5.2f", 2.5 , 1.4);

для типов *double*, если указаны две буквы *lf* (*long float*), и *float*, если *f* (*float*).

Простые типы

```
…
  printf("int i=-2000000000 -> i=%li i=%d i=%x i=%o \n", i,i,i,i);
  printf("unsigned int ui = 4000000000 -> ui=%u\n", ui);
  printf("short sh=-32000 -> sh=%d, ", sh);
  printf("unsigned short ush = 64000 -> ush=%d \n\n", ush);
  printf("char c=-128 -> c=%d, ", c);
  printf("unsigned char uc = 255 -> uc=%d \n\n", uc);

  printf("float f=2.3 -> f=%3.1f   f=%f \n", f, f);
  printf("double d = -4.3e-5 -> d=%lf d=%3.1lf d=%g\n\n", d, d, d);

  printf("unsigned char ch='A', ch2=49; -> ch=%c, ch2=%c \n", ch, ch2);

  printf("const CN = 33; #define DCN 35 -> CN=%d, DCN2=%d \n",CN,DCN);

  getch();
  return 0;
}
//---------------------------------------------
```

Данное окончание предыдущей программы с описание переменных разных типов выполнит следующий форматированный вывод:

```
int i=-2000000000 -> i=-2000000000 i=-2000000000 i=88ca6c00 i=21062466000
unsigned int ui = 4000000000 -> ui=4000000000
short sh=-32000 -> sh=-32000, unsigned short ush = 64000 -> ush=64000

char c=-128 -> c=-128, unsigned char uc = 255 -> uc=255

float f=2.3 -> f=2.3   f=2.300000
double d = -4.3e-5 -> d=-0.000043 d=-0.0 d=-4.3e-05

char ch = 'A', ch2 = 49; -> ch=A, ch2=1
const CN = 33; #define DCN 35 -> CN=33, DCN2=35
```

Операторы присвоения, сравнения и логические в *Delphi*(*Object Pascal*) и *C*

Оператор	Object Pascal	C
Присвоение	:=	=
Сравнение	=	==
Неравенство	<>	!=
Меньше, чем	<	<
Больше, чем	>	>
Меньше или равно	<=	<=
Больше или равно	>=	>=
Логическое *и*	and	&&
Логическое *или*	or	||
Логическое *нет*	not	!

C: int i=j=0; // не путать с i=j==0
if (i==j) // не путать с if (i=j)

Арифметические операторы в Delphi(Object Pascal) , C , Basic

Оператор	Object Pascal	C	Visual Basic
Сложение	+	+	+
Вычитание	-	-	-
Умножение	*	*	*
Деление с плавающей точкой	/	/	/
Деление целых чисел	div	/	\
Деление по модулю	mod	%	Mod
Возведение в степень	Отсутствует	Отсутствует	^

В С оба вида деления обозначаются одинаково и зависят **от типа** операторов.

```
y=1/3*x;  // будет 0, т.к. операнды целые
y=1.0/3*x;  // правильно  1.0 – не целое
int x=1, z=3;  y= x / z; // будет 0
y=x / (double)z;  // с приведением типа одного из целых операндов к вещественному
y= (double)x /z;  // с приведением типа одного из целых операндов к вещественному
```

Возведение в степень ^ из *Basic* (в том числе *Visual Basic for Application*) многие в первом семестре пытались использовать и в *Delphi*. Но теперь вы уже знаете, что это операция разыменования при работе с указателями и при объявлении типа указателя. А в *C* это оператор побитовой операции (см. далее)

Побитовые операторы

Побитовые операторы

Оператор	Object Pascal	C	Visual Basic
И	and	&	And
Не	not	~	Not
Или	or	|	Or
Исключающее или	xor	^	Xor
Сдвиг влево	shl	<<	Нет
Сдвиг вправо	shr	>>	Нет

```
5 & 6 => 4      5 | 6 => 7      5 ^ 6 => 3
0101            0101            0101
0110            0110            0110
0100            0111            0011
```

```
+=   -=   *=   /=   %=
&=   |=   ^=   <<=  >>=

Sum=Sum+A[i]; → Sum+=A[i];
i = i +1;  →  i+=1;  →  i++;
K = K % 2; →  K%=2;
```

Гречкина П.В.,
Программирование, Язык С

В *C* операторы увеличения и уменьшения могут быть частью арифметического выражения (вычисляются до или после него) или списка фактических параметров функции (вычисляются в порядке передачи параметров – см. стандарты вызовов в первой части курса – в *C* с конца списка параметров).

Операторы

Процедуры увеличения и уменьшения

Процедуры увеличения и уменьшения генерируют оптимизированный код для добавления или вычитания единицы из целой переменной. Object Pascal не предоставляет таких широких возможностей, как постфиксные и префиксные операторы ++ и -- в языке C. Тем не менее процедуры `Inc()` и `Dec()` обычно компилируются в одну команду процессора.

```
Delphi (фрагмент)
Var k: integer; pk: ^integer;
Begin
K:=10; // Регистр! K, k
Inc( k );  // k := integer(Ord(k)+1);
Dec( k );  // -1

Inc( k, 2);  //+2
Dec( k, 3); //-3

AllocMem( pk, 2* SizeOf(integer));
Inc( pk ); // + SizeOf(integer);
Dec( pk );   pk^:=5;
FreeMem(pk, 2* SizeOf(integer));
```

```
int sum (int k1, int k2)
{
        return k1+k2;
}
```

```
C++ (фрагмент)
int k=10, *Pk;
k++;   --k;
k= k + ++k;  printf("%d\n", k);  //22
k=10; k= sum (k, ++k); printf("%d", k); //22
k=10; k= sum (k, k++); printf("%d", k); //21
k=10;
k= sum (++k, ++k); printf("%d", k); //23
k=10;
k= sum (k++, k++); printf("%d", k); //21

Pk = (int*) calloc(2, sizeof(int)) ;
Pk++;  Pk--;   *Pk=5;  Pk[0]=1; Pk[1]=2;
free (Pk);
```

Гречкина
аммирован

Одномерный массив

	Динамический (#include <stdlib.h> или<alloc.h>	Статический

Описание переменной	int a[] = {0,1,2,30,-40}; // с инициализацией
	int *b;
		int a[5];
		int b[3] = {0,1,2}

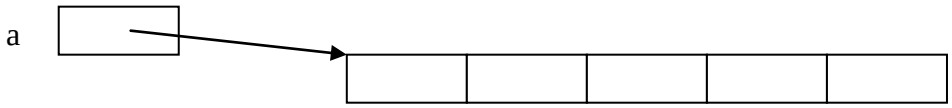
Выделение памяти	a = new int[5]; // только в c++
	b = (int*)calloc(3, sizeof(int)); // с обнулением
	b = (int*)malloc(3*sizeof(int)); // без обнуления
	b = (int*)realloc(3, sizeof(int)); // перевыделение
		При описании

Обращение к элементу	a[0] *a b[i] *(b+i)	a[0] *a b[i] *(b+i)

Освобождение памяти	delete [] a; // только в c++
	free(b);
		При выходе из области видимости, если не static

Описание параметра	Для изменения значений
	void proc1(int a[], int *b);
	void proc2(int *a, int b[]);
	void proc3(int a[5], int b[3]);	

	Для выделения/освобождения памяти
	void proc4(int (*a)[], int **b);
	void proc5(int **a, int (*b)[]);
	void* func_a(int a[]); // вызов a=(int*)func_a(a);
	int* func_b(int *b); // вызов b=func_b(b);
		Невозможно освободить/ перевыделить память

Структура	Указатель на первый элемент (int a[5]) 	

Основные управляющие структуры языка С

1.Следование

1.1. ; является частью оператора, которые просто следуют один за другим s=0; for (i=0; i<n; i++) {s+=a[i];}

1.2. , пишется между операторами, объединяя их в один. Результат – результат последнего оператора, например, внутри ***for:***

for (s=0, i=0; i<n; s+=a[i], i++); и for (s=0, i=0; i<n;) s+=a[i], i++;

2.Ветвление

2.1. структура *if* и оператор ?:

C (см. скобки круглые и фигурные в этом примере)	Аналог в *Delphi*

if (a[i]==3 && i%2==0) {
a[i]=1;
}
else {
a[i]=i;
}
	If (a[i] = 3) and (i mod 2 = 0) then
	begin
	a[i]:=1
	end
	Else
	begin
	a[i]:=i
	end

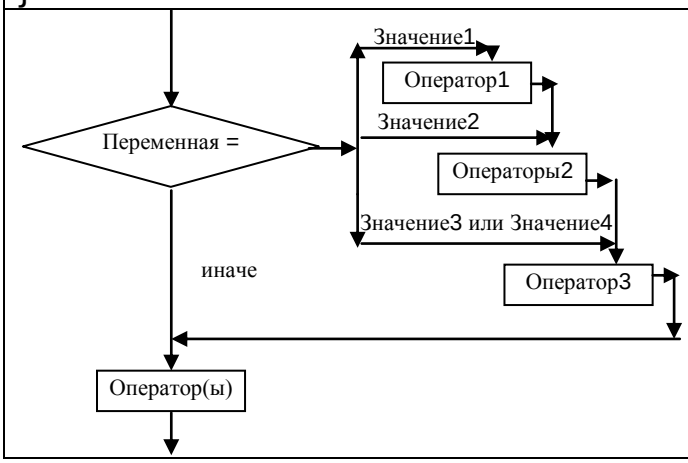
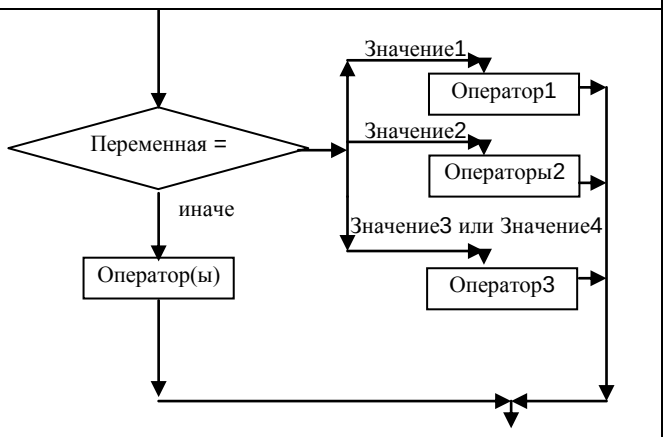
C	Аналог в *Delphi*
if (x==4) y=x; else y=8;	If x=4 then y:=x else y:=8;
y=(x==4 ? x : 8)	y:=abs(ord(x=4))*x + (1- abs(ord(x=4)))*8;

2.2. switch

C	Аналог в *Delphi*

switch (k){
case 1: y=0; // не конец
case 2: y++;
case 3:
case 4: y+=k;
default: y+=2*k;
}
	case k of
	1: begin y:=0; y:=y+1 +k +2*k; end;
	2: y:=y+1 +k +2*k;
	3, 4 : begin y:=y+k;
	y:=y+2*k;
	end;
	else y:=y+2*k;
	end;

switch (k){
case 1: y=0; **break**;
case 2: y=1; **break;**
case 3:
case 4: y+=k; **break**;
default: y+=2*k;
}
	case k of
	1: y:=0;
	2: y:=1;
	3, 4 : y:=y+k;
	else y:=y+2*k;
	end;

3.Циклы

3.1. for

for (подготовка **;** условие_продолжения **;** подготовка_к_след_итерации) {тело_цикла}

все три части могут быть пусты: for(;;) { if(условие_выхода) break; }

3.2. while - цикл итерационный с предусловием

while (условие**_продолжения**) {тело цикла }

3.3. do while - цикл итерационный с постусловием

do {
тело цикла
}while(условие**_продолжения**);
// в Delphi в цикле с постусловием repeat until – условие**_выхода**

Пример программы с одномерным статическим массивом:

```
//---------------------------------------------------------------------------
#include <conio.h>  //там getch
#include <stdio.h>  // scanf , printf, FILE, fprintf, fscanf, fgets и др.функции стандартного в/в
#include <math.h> // M_PI, pow, ceil\floor, log, exp, sqrt, fabs и другие математические функции
//---------------------------------------------------------------------------
void main()
{
        int n,i;
        float a[20]; // статический массив a[0]…a[19]
        float sum;

        printf("n=? "); // приглашение
        scanf("%d", &n); //ввод десятичного(%d) цел.числа
                                // и сохранение его по адресу переменной n
        if (n<1 || n>20) {
                printf("Invalid n! \nPress any key");
                getch(); // ожидание нажатия клавиши (любой неслужебной)
                return;  // выход из функции main
        }

        printf("Input a(%d): \n", n); // приглашение и вЫвод n в виде десятичного числа
        for (i = 0; i < n; i++) {     // ввод   поэлементный  a[0]...a[n-1]  в цикле по i
                scanf("%f", &a[i]);       // типа float (%f)
        }

        printf("a(%d): \n", n); // вывод n
        for (i = 0; i < n; i++) {     // вывод поэлементный  a[0]...a[n-1] в цикле по i
                printf("%4.1f ", a[i]);   // типа float в формате :4:1  через пробел
        }
        printf("\n"); // переход на след строку

        // поиск суммы абсолютных значений элементов массива
        sum=0.0;
        for (i = 0; i < n; i++) {
                sum+= fabs(a[i]); // abs - для целых; fabs - для чисел с плавающей точкой
        }

        // вывод десятичной n и суммы sum типа float в формате  :6:1
        printf("Summa a(%d) = %6.1f \nPress any key", n, sum);
        getch(); // ожидание нажатия клавиши

        return; // выход из функции main
}
```

Пример аналогичной программы с одномерным динамическим массивом:

```
//---------------------------------------------------------------------------
#include <conio.h>  //там getch
#include <stdio.h>  // scanf , printf, FILE, fprintf, fscanf, fgets и др.функции стандартного в/в
#include <math.h> // M_PI, pow, ceil\floor, log, exp, sqrt, fabs и другие математические функции
//---------------------------------------------------------------------------

void main()
{
        int n,i;
        float *a; //динамический массив
        float sum;

        printf("n=? "); // приглашение
        scanf("%d", &n);  // ввод десятичного(%d) n
        if (n<1 || n>200) {
                printf("Invalid n! \nPress any key");
                getch(); // ожидание нажатия клавиши
                return;  // выход из функции main
        }

        a = new float [n];  // выделяем память для n элементов массива типа float

        printf("Input a(%d): \n", n); // приглашение с выводом n
        for (i = 0; i < n; i++) {     // ввод поэлементный  a[0]...a[n-1] в цикле по i
                scanf("%f", &a[i]);       // типа float (%f)
        }

        printf("a(%d): \n", n); // вывод n и
        for (i = 0; i < n; i++) {     // вывод поэлементный  a[0]...a[n-1] в цикле по i
                printf("%4.1f ", a[i]);   // типа float в формате :4:1  через пробел
        }
        printf("\n"); // переход на след строку

        // поиск суммы абсолютных значений элементов массива
        sum=0.0;
        for (i = 0; i < n; i++) {
                sum+= fabs(a[i]); // abs для целых, fabs для чисел с плавающей точкой
        }

        delete [] a;// освобождаем память, выделенную для массива a

        // вывод десятичной n и суммы sum типа float в формате  :6:1
        printf("Summa a(%d) = %6.1f \nPress any key", n, sum);
        getch(); // ожидание нажатия клавиши

        return; // выход из функции main
}
//---------------------------------------------------------------------------
```