**Глава II. Модульное программирование**

1. **Функции**

**1.1. Определение и объявление**

***Объявление функции*** *(прототип, заголовок, сигнатура)*

- имя,

- тип возвращаемого значения,

- список передаваемых параметров.

***Определение функции***

**-** объявление функции,

- *тело* функции.

[класс] тип имя **(** [список\_параметров]**)**[**throw** **(** исключения **)**]**{**

тело функции

**}**

**1.** Модификатор «класс»

**extern**

**static**

**inline**

**2.** Тип возвращаемого значения

**3.** Список параметров

- разделяются *запятыми*.

- *тип* и *имя* – для *каждого*.

**4.** Исключения - позже

**double power(double, double); //объявление функции**

**int main(){**

**double a = 16.0, b = 0.25;**

**printf("%lf**\n**",power(a,b));//вызов функции**

**return 0;**

**}**

**//определение функции**

**double power(double x, double y){**

**return exp(y\*log(x));**

**}**

***Исполнительный стек программы*** (***стек вызовов функций***)

*Автоматические переменные функции*:

- локальные переменные функции

- параметры функции, переданные по значению

*Кадр стека* содержит:

- адрес возврата,

- зарезервированную память под автоматические переменные вызванной функции.

**1.2. Обмен информацией**

**1.** Глобальные переменные

**2.** Возвращаемое функцией значение

**3.** Параметры функции

*формальные параметры* или просто *параметры*

*фактические параметры* или *аргументы*

Способы ***передачи параметров в функцию***:

1)  ***по значению***

копии значений → нельзя изменить исходные значения

2) ***по адресу***

*- при помощи указателей*

- *при помощи ссылки* (только в C++)

копии адресов → можно изменить исходные значения

**int my\_function(double a, double \*b){//определение**

**if (a == 0)**

**return 1;**

**else{**

**\*b = 5/a;**

**return 0;**

**}**

**}**

**int main(){**

**double x,y;**

**printf("Enter x: ");**

**scanf("%lf",&x);**

**int key = my\_function(x,&y); //вызов функции**

**if (key == 1)**

**printf("Error! Divizion by 0!\n");**

**else**

**printf("y = 5 / x = %lf\n",y);**

**return 0;**

**}**

С++

**int my\_function(double a, double &b){//определение**

**…**

**b = 5/a;**

**…**

**}**

**int main(){**

**…**

**int key = my\_function(x,y); //вызов функции**

**…**

**return 0;**

**}**

По умолчанию параметры любого типа, *кроме массива и функции,* передаются в функцию по значению. Массив всегда передается по адресу.

**int first\_element (int n, int massiv[100]){**

**massiv[0] = 0; …**

**}**

**first\_elements(size, m); // теперь m[0] == 0**

**int first\_elements(int n, const int massiv[100]){**

**massiv[0] = 0; … // ошибка компиляции**

**}**

**int first\_element (int n, int massiv[100]){…}**

Лучше так

**int first\_element (int n, int massiv[]){…}**

Или так

**int first\_elements(int n, int \*massiv){…}**

Еще лучше так

**int first\_elements(const int n, int \*massiv){…}**

Тогда

**int main(){**

**const int size = 10;**

**int m[size] = {1,-1,1,-1,1,-1,1,-1,1,-1};**

**first\_elements(size,m);**

**return 0;**

**}**

Внутри функции *многомерный массив* *интерпретируется как одномерный*, а его индекс пересчитывается.

Лучше работать с двумерным массивом естественным образом – используем динамические массивы.

**int new\_matr(const int, const int, double\*\*&);**

**int del\_matr(const int, double\*\*&);**

**int rand\_matr(const int, const int, double\*\*);**

**int print\_matr(const int, const int, double\*\*);**

**int change\_matr(const int, const int, double\*\*);**

**int main(){**

**int n=5, m=3;**

**double\*\* A;**

**new\_matr(n,m,A);**

**rand\_matr(n,m,A);**

**print\_matr(n,m,A);**

**change\_matr(n,m,A);**

**print\_matr(n,m,A);**

**del\_matr(n,A);**

**return 0;**

**}**

**int new\_matr(const int n\_str, const int n\_stb, double\*\* &matr){**

**matr = new double \*[n\_str];**

**for(int i=0; i<n\_str; i++)**

**matr[i] = new double [n\_stb];**

**return 0;**

**}**

**int del\_matr(const int n\_str, double\*\* &matr){**

**for(int i=0; i<n\_str; i++)**

**delete [] matr[i];**

**delete [] matr;**

**return 0;**

**}**

**int rand\_matr(const int n\_str, const int n\_stb, double\*\* matr){**

**srand(time(0));**

**for(int i=0; i<n\_str; i++)**

**for(int j=0; j<n\_stb; j++)**

**matr[i][j] = ((double)rand() / RAND\_MAX);**

**// случайное число на отрезке [0,1]**

**return 0;**

**}**

**int print\_matr(const int n\_str, const int n\_stb, double\*\* matr){**

**for(int i=0; i<n\_str; i++){**

**for(int j=0; j<n\_stb; j++)**

**printf("%8.5lf ",matr[i][j]);**

**printf("\n");**

**}**

**printf("\n");**

**return 0;**

**}**

**int change\_matr(const int n\_str, const int n\_stb, double\*\* matr){**

**for(int i=0; i<n\_str; i++)**

**for(int j=0; j<n\_stb; j++)**

**matr[i][j] = i;**

**return 0;**

**}**

***Произведение матриц***

**int\*\* new\_matr(const int n){**

**int\*\* matr = new int \*[n];**

**if (!matr) {**

**printf("Error in new.\n");**

**return NULL;**

**}**

**for(int i=0; i<n; i++){**

**matr[i] = new int [n];**

**if (!matr[i]) {**

**printf("Error in new.\n");**

**return NULL;**

**}**

**}**

**return matr;**

**}**

**int\*\* del\_matr(const int n, int\*\* matr){**

**for(int i=0; i<n; i++)**

**delete [] matr[i];**

**delete [] matr;**

**return NULL;**

**}**

**int\*\* read\_matr(int \*pn){**

**char s[80];**

**printf("Enter filename for read: ");**

**gets(s);**

**FILE \*f = fopen(s,"r");**

**if (!f){**

**printf("Error in open.\n");**

**return NULL;**

**}**

**if (fscanf(f,"%i",pn) != 1){**

**printf("Error in read.\n");**

**fclose(f);**

**return NULL;**

**}**

**int\*\* matr= new\_matr(\*pn);**

**if (!matr){**

**fclose(f);**

**return NULL;**

**}**

**for(int i = 0; i < \*pn; i++){**

**for(int j = 0; j < \*pn; j++){**

**if (fscanf(f,"%i",&matr[i][j]) != 1){**

**printf("Error in read.\n");**

**matr = del\_matr(\*pn,matr);**

**fclose(f);**

**return NULL;**

**}**

**}**

**}**

**fclose(f);**

**return matr;**

**}**

**int print\_matr(const int n, int\*\* matr){**

**for(int i = 0; i < n; i++){**

**for(int j = 0; j < n; j++){**

**printf("%7i ",matr[i][j]);**

**}**

**printf("\n");**

**}**

**printf("\n");**

**return 0;**

**}**

**int mult\_matr(const int n, int\*\* A, int\*\* B){**

**int\*\* matr;**

**matr = new\_matr(n);**

**if (!matr){return 1;}**

**for(int i = 0; i < n; i++)**

**for(int j = 0; j < n; j++){**

**matr[i][j] = 0;**

**for(int k = 0; k < n; k++)**

**matr[i][j] += A[i][k] \* B[k][j];**

**}**

**print\_matr(n,matr);**

**matr = del\_matr(n,matr);**

**return 0;**

**}**

**int main(){**

**int n;**

**int\*\* A;**

**char ch;**

**char s[81];**

**do{**

**A = read\_matr(&n);**

**if(A){**

**print\_matr(n,A);**

**mult\_matr(n,A,A);**

**A = del\_matr(n,A);**

**}**

**printf("Repeat (y/n)? ");**

**gets(s);**

**ch = s[0];**

**while ((ch != 'y')&&(ch != 'n')){**

**printf("Press \"y\" for repeat or**

**\"n\" for exit: ");**

**gets(s);**

**ch = s[0];**

**}**

**}while(ch == 'y');**

**return 0;**

**}**

**1.3. Рекурсивные функции**

***Рекурсия***

1. *Прямая*. 2. *Косвенная*.

**int f\_i(int n){**

**int result = 1;**

**for(int i = 1; i <= n; i++)**

**result = result \* i;**

**return result;**

**}**

**int f\_r(int n){**

**if (n == 1)**

**return 1; // 1! = 1 НЕРЕКУРСИВНАЯ ВЕТВЬ**

**else**

**return n\*f\_r(n-1); //n! = n \* (n-1)!**

**}**

**1.4. Функция main()**

Количество параметров, передаваемых в функцию, включая имя функции

1. Без параметров:

**тип main() {…}**

2. С двумя параметрами:

**тип main(int argc, char\* argv[]) {…}**

**#include<stdio.h>**

**void main(int argc, char\* argv[]){**

**for (int i = 0; i<argc; i++)**

**printf("%s\n",argv[i]);**

**}**

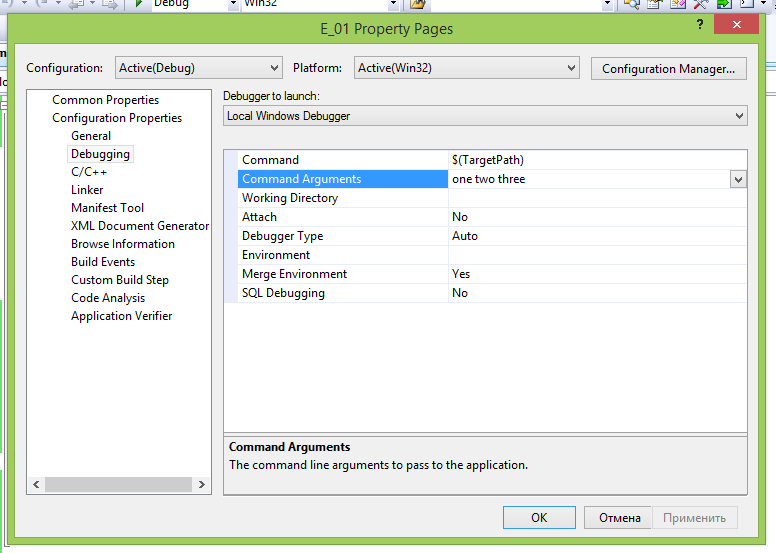
Файл вызывается из командной строки: **d:\main.exe one two three**

D:\MAIN.EXE

one

two

three



**1.5. Функции стандартной библиотеки**

- ввод/вывод,

- обработка строк,

- математические функции,

- работа с динамической памятью,

- поиск и сортировка и т. д.

***1.5.1. Функции ввода/вывода***

1. Функции в С.

2. Потоковые классы в C++.

Смешивать эти два способа в одной программе не рекомендуется.

*Ввод/вывод в стиле С++*

(позже, в разделе «Потоковые классы»)

- легче в использовании в случаях, не требующих форматирования

- можно переопределить для собственных классов.

*Ввод/вывод в стиле С*

- функции удобнее при форматированном выводе

- не используется объектно-ориентированный подход.

1. Заголовочный файл: **<stdio.h>** (**<cstdio>**).

2. При вводе/выводе *данные рассматриваются как поток байтов*.

3. Поток:

- файл

- устройство

- клавиатура

- монитор

***1. Открытие потока***

- для чтения и/или записи,

- в двоичном или текстовом режиме.

**FILE\* fopen(const char\* filename, const char\* mode);**

Первый параметр — имя открываемого файла.

Второй — *режим открытия файла*:

По умолчанию – текстовый режим

«**r**» «**w**» «**а**»

«**r+**» «**w+**» «**а+**»

Двоичный режим

«**t**» «**b**»

**FILE \*f = fopen("test.txt","rb+");**

Далее для работы с файлом используется указатель **f**

***2. Ввод/вывод в поток***

Последовательно, начиная *с текущей позиции потока*, определяемой положением *указателя потока.*

**ftell** и **fgetpos**

**fseek** и **fsetpos**

**fread –**

**fwrite) –**

**fgetc** **getchar**

**fputc** **putchar**

**fgets** **gets**

**fputs** **puts**

**fscanf** **scanf**

**fprintf** **printf**

При работе со строками в функциях **fscanf** и **scanf** операцию **&** не используют, так как имя **s** – это указатель на начало строки.

**char s[81];**

**scanf("%s",s); fscanf(f,"%s",s);**

**//возвращают число переменных,**

**//которым присвоено значение**

Лучше использовать функции **fgets** и **gets**.

**char s[81];**

**int n;**

**gets(s);**

**//символ '\n' в строку не включается**

**fgets(s,n,f);**

**//не более n-1 байт до '\n' или конца файла;**

**//символ '\n' помещается в конец строки + '\0'**

***3. Закрытие потока***

**int** **fclose(FILE\*)**

**fclose(f);**

***4. Обработка ошибок***

**int feof (FILE\*)**

**int ferror (FILE\*)**

***Сохранение матрицы в текстовом файле***

**int save\_matr(int n\_str, int n\_stb, double\*\* matr){**

**char s[81];**

**printf("Enter filename for save: ");**

**gets(s);**

**if (strchr(s,'.')==NULL) strcat(s,".txt");**

**FILE \*f = fopen(s,"w");**

**if (f == NULL){**

**printf("Error in open.\n"); return 1;**

**}**

**fprintf(f,"%i\n",n\_str);**

**fprintf(f,"%i\n",n\_stb);**

**for(int i = 0; i < n\_str; i++)**

**for(int j = 0; j < n\_stb; j++)**

**fprintf(f,"%lf\n",matr[i][j]);**

**fclose(f);**

**return 0;**

**}**

***Посимвольное и буферизованное чтение из файла***

Из файла можно читать посимвольно, но это медленно.

**FILE \*f\_in = fopen("test.txt","r");**

**while (!feof(f\_in)){**

**putchar(fgetc(f\_in));**

**//чтение символа из файла и вывод его на экран**

**}**

**fclose(f\_in);**

Написать программу, которая считывает текст из файла и выводит на экран только вопросительные предложения этого текста.

**FILE \*f\_in;**

**f\_in = fopen("test.txt","r");**

**if (!f\_in) {puts("Error in open."); return 1;}**

**fseek(f\_in,0,SEEK\_END);**

**//перемещает указатель на 0 байт от конца**

**//файла (SEEK\_SET - начало SEEK\_CUR - текущее)**

**long len = ftell(f\_in);**

**//текущая позиция в файле - теперь это размер файла**

**char \*buf = new char[len+1];**

**//в buf поместим текст из файла, +1 байт для '\0'**

**const int l\_block = 1024;**

**int num\_block = len / l\_block + 1;**

**fseek(f\_in,0,SEEK\_SET);**

**fread(buf, l\_block, num\_block, f\_in);**

**buf[len] = '\0';**

**long n = 0; //начало текущего предложения**

**long i = 0;**

**while (buf[i] != '\0'){**

**if (buf[i] == '?'){**

**for(long j = n; j <= i; j++)**

**putchar(buf[j]);**

**n = i+1;**

**}**

**if ((buf[i] == '.') || (buf[i] == '!'))**

**n = i+1;**

**i++;**

**}**

**fclose(f\_in);**

**printf("\n");**

***1.5.2. Функции работы со строками и символами***

**<string.h>** (**<cstring>**) – самостоятельно

**<stdlib.h>** (**<cstdlib>**)

**double atof(const char\* s)**

**int atoi (const char\* s)**

**long atol (const char\* s)**

**<stdio.h>** (**<cstdio>**)

**sprintf**

**<ctype. h>** (**<cctype>**)

**is…** – самостоятельно

**tolower** и **toupper**

***1.5.3. Математические функции***

**<math.h>** (**<cmath>**):

**abs**, **fabs**

**ceil**, **floor**

**sqrt**

**pow**

значения тригонометрических функций,

**exp**

**log**, **log10**

**modf**

**fmod**

и другие.

**2. Директивы препроцессора**

# - первый символ в строке после пробелов.

**2.1. Директива #include**

**#include <имя\_файла>**

**#include “имя\_файла“**

*Вставляет содержимое указанного файла в ту точку исходного файла, где она записана*.

Отделяйте интерфейс от реализации!

**//файл file1.h**

**int new\_matr(int, int, double\*\* &);**

**int del\_matr(int, double\*\* &);**

**int save\_matr(int, int, double\*\*);**

**int read\_matr(int &, int &, double\*\* &);**

**int rand\_matr(int, int, double\*\*);**

**int print\_matr(int, int, double\*\*);**

**int change\_matr(int, int, double\*\*);**

**//файл main.cpp**

**#include"file1.h"**

**int main(){…}**

**//файл file1.cpp**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<time.h>**

**#include<string.h>**

**int new\_matr(int n, int m, double\*\* &matr){…}**

**int del\_matr(int n, double\*\* &matr){…}**

**int save\_matr(int n, int m, double\*\* matr){…}**

**int read\_matr(int &n, int &m, double\*\* &matr){…}**

**int rand\_matr(int n, int m, double\*\* matr){…}**

**int print\_matr(int n, int m, double\*\* matr){…}**

**int change\_matr(int n, int m, double\*\* matr){…}**

**2.2. Директива #define**

*Определяет подстановку в тексте программы.*

***Символические константы***

**#define имя текст\_подстановки**

**#define PI 3.14159**

**#define PI = 3.14159**

**//в выражении x = r\*PI; произойдет подстановка**

**//x = r\*=PI**

**double pi = 3.14159;**

**//меньше ошибок, но требуется дополнительное**

**//место в памяти**

***Макросы***

**#define имя(параметры) текст\_подстановки**

**#define CIRCLE\_AREA(x) (PI \* (x) \* (x))**

**//тип данных аргументов макроса не проверяется**

**#define CIRCLE\_AREA(x) PI \* x \* x**

**//при вызове CIRCLE\_AREA(x + 1) произойдет подстановка**

**//PI \* x + 1 \* x + 1**

**double circle\_area (double x){return 3.14159 \* x \* x;}**

**//меньше объем программы, но медленнее**

**//(тратится время на вызов функции)**

**//предпочтительнее inline-функции (встроенные функции):**

**//код функции помещается в точку вызова**

Лучше использовать константыи функции.

**2.3. Директивы условной компиляции**

*Применяются для того, чтобы исключить компиляцию отдельных частей программы.*

1. Временный комментарий фрагмента кода.

**#if 0**

**int i;**

**/\***

**int j;**

**\*/**

**#endif**

2. Проверка, определена ли символическая константа.

**#ifndef \_NULL\_**

**#define \_NULL\_ 0**

**#endif**

**2.4. Предопределенные символические константы**

**#ifdef \_\_cplusplus**

**printf("It's C++!!!\n");**

**#endif**

**printf("%s\n", \_\_FILE\_\_);**

**printf("%s\n", \_\_TIME\_\_);**

**printf("%s\n", \_\_DATE\_\_);**

**printf("%i\n", \_\_LINE\_\_);**