

Лабораторно упражнение № 1

Структура на C/C++ програма. Среда за програмиране. Въвеждане и извеждане на данни.

I. Теоретична обосновка

1. Структура на програма на език C/C++

Всяка програма, написана на език C/C++ се състои от отделни програмни единици, наречени *функции*. Броят на функциите в програмата е неограничен, но винаги трябва да има една функция, дефинирана като главна – това е функция **main()**.

Функция **main()** осъществява връзката с операционната система. При стартиране на изпълнимия модул (.COM или .EXE), операционната система предава управлението на главната функция **main()**. Съответно **main** функцията може да извиква други функции т.е. да им предава управлението.

Функция е набор декларации и последователно написани оператори на езика, осъществяващи действия, които представляват логическа цялост и реализират определен алгоритъм.

Общата структура на програма, написана на език C/C++ може да се представи чрез следните компоненти:

```
// описания на заглавни файлове
#include <io.h>
#include <stdio.h>
// дефиниции на външни променливи
int number;
unsigned char byte;

// главна функция
main()
// тяло на главната функция
{...
}
// функция 1
...
// функция 2
```

2. Етапи от развитието на програмите

Процесът на създаване на програмно осигуряване преминава през следните основни етапи:

2.1. Създаване и редактиране на първичния код

На този етап, въз основа на разработения алгоритъм, се създава първичния (*source*) код на програмата. Това е програмата, записана на алгоритмичен език. Първичният код се записва като ASCII файл, като обикновено разширението подсказва на какъв програмен език е кодирана програмата. За език C, разширението на файла е **.C**, а за език C++, разширението е **.CPP**.

2.2. Компилиране

На етап *компилиране* се превежда първичният код на програмата, написан на алгоритмичен език, на машинен език. Програмата-транслатор, която осъществява превода на source кода се нарича *компилятор*.

Компиляторът анализира първичния код и когато открие синтактични грешки или несъответствия, издава съответните съобщения. В този случай, е необходимо първичният код да бъде редактиран отново, с цел премахване на грешките, и повторно компилиран.

При успешно компилиране, се създава т.нар. *обектен код* – програмата, преведена на машинен език. Обектният код се записва във файл с име, което съвпада с името на първичния код, но с разширение **.OBJ**.

Компиляторите позволяват да се промени името на файла с обектния код т.е. имената на файловете с първичния и обектния код да не съвпадат. Обикновено, промяната на името на обектния код, рядко се практикува.

2.3. Свързване

На етап *свързване*, обектният код се свързва със системните библиотеки и други обектни модули. Програмата се прави изпълнима под управлението на операционна система. Свързването се осъществява с програма *свързващ редактор (linker)*.

При успешно приключване на етапа, свързващият редактор създава т. нар. *изпълним модул*, записан като изпълним файл за съответната операционна система. За операционна система WINDOWS, изпълнимите файлове са с разширение **.COM** или **.EXE**.

При условие, че на етап свързване възникнат грешки, необходимо е първичният код да бъде редактиран и повторно компилиран.

2.4. Тестване и настройка

Въпреки, че след етапа свързване, програмата е готова за изпълнение, това не означава, че тя работи коректно. Необходимо е, програмата да бъде тествана дали при съответните входни данни, се получават верни резултати.

В голяма част от случаите, първоначално, разработеното програмно осигуряване работи некоректно. Грешките се дължат на логическо несъответствие с алгоритъма или на неверен алгоритъм. В етапа *тестване и настройка* се откриват и отстраняват логическите грешки в програмата. Разработени са специални програми за тестване и настройка, които се наричат *дебъгери*. Дебъгерите използват следните средства за тестване на програми:

- постъпково изпълнение на програмата;
- задаване на точки на прекъсване;
- наблюдение на променливи.

3. Среда за програмиране Microsoft Visual Studio.

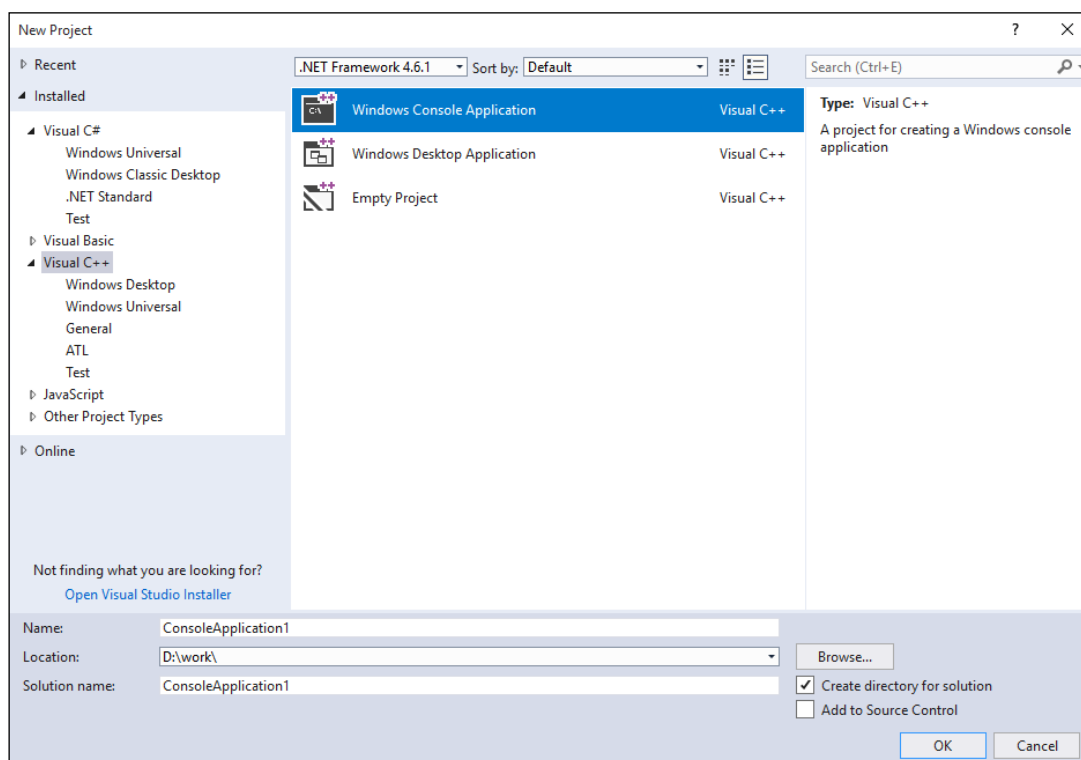
За създаването на програмно осигуряване, като е започне със създаването на първичния код и се приключи с тестване на изпълнимия модул, са необходими следните системни програми: *текстов редактор* за редактиране на първичния код, *компилятор*, *свързващ редактор* и *дебъгер*. В съвременните програмни системи, тези програми са обединени в една обща, интегрирана, **развойна програмна среда**. Развойната среда включва както средства за създаване и редактиране на текстови файлове, така и средства за компилиране, свързване, тестване и настройка на програмно осигуряване. MS Visual Studio е интегрирана развойна среда, чрез която се изграждат различни приложения.

3.1. Създаване на конзолни приложения чрез Microsoft Visual Studio.

Когато се започне с изграждането на дадено приложение, първо се създава **проект**. Чрез проектът се групират множеството файлове, които изграждат една цялостна програмна система. За начинаещи програмисти е препоръчително всяка една отделна задача, колкото и елементарна да е, да се оформя като самостоятелен програмен проект.

Създаването на проект е със следните стъпки:

- Команда File/New Project и се отваря диалог New Project (фиг.2.1).
- Избира се език за програмиране, в случая C++. Да се има предвид, че средата за програмиране MS Visual Studio поддържа няколко програми езика, освен C++.



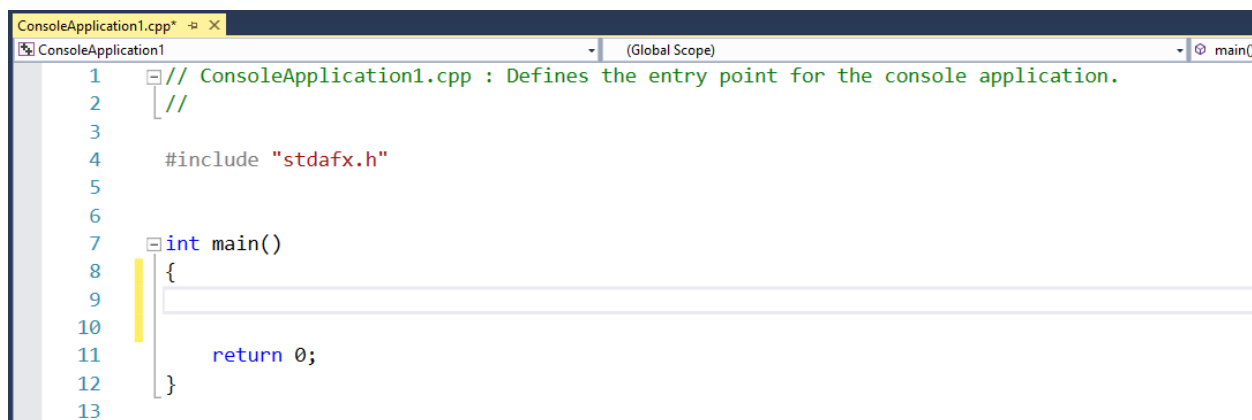
Фиг. 2.1. Създаване на нов проект с Visual Studio

- Избира се тип на проекта, в случая конзолно приложение (**Windows Console Application**).

Конзолните приложения са програми, чрез които входът и изходът е на **системната конзола** (клавиатурата и екрана) т.е. те очакват вход от клавиатурата и отпечатват като текст на екрана изходен резултат. Този тип приложения са удобни за начално обучение по програмиране.

- Определя се името на проекта (поле Name) и папката в която ще се съхранява (поле Location). Изборът на дисково устройство и папка е чрез бутон Browse

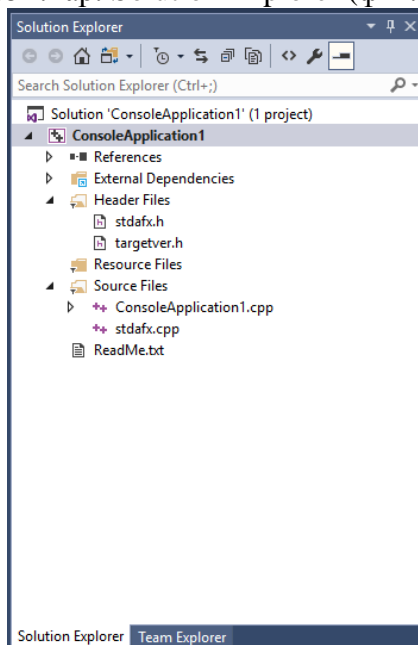
При създаването на нов проект MS Visual Studio генерира програмен код, който формира т.нар. **скелет на приложението**. Генерира се и дефиницията на функцията main(), в чието тяло програмистът може да добави съответния програмен код. (фиг.2.2)



```
1 // ConsoleApplication1.cpp : Defines the entry point for the console application.
2 //
3
4 #include "stdafx.h"
5
6
7 int main()
8 {
9
10
11     return 0;
12 }
13
```

Фиг. 2.2. Програмен код на функция *main*

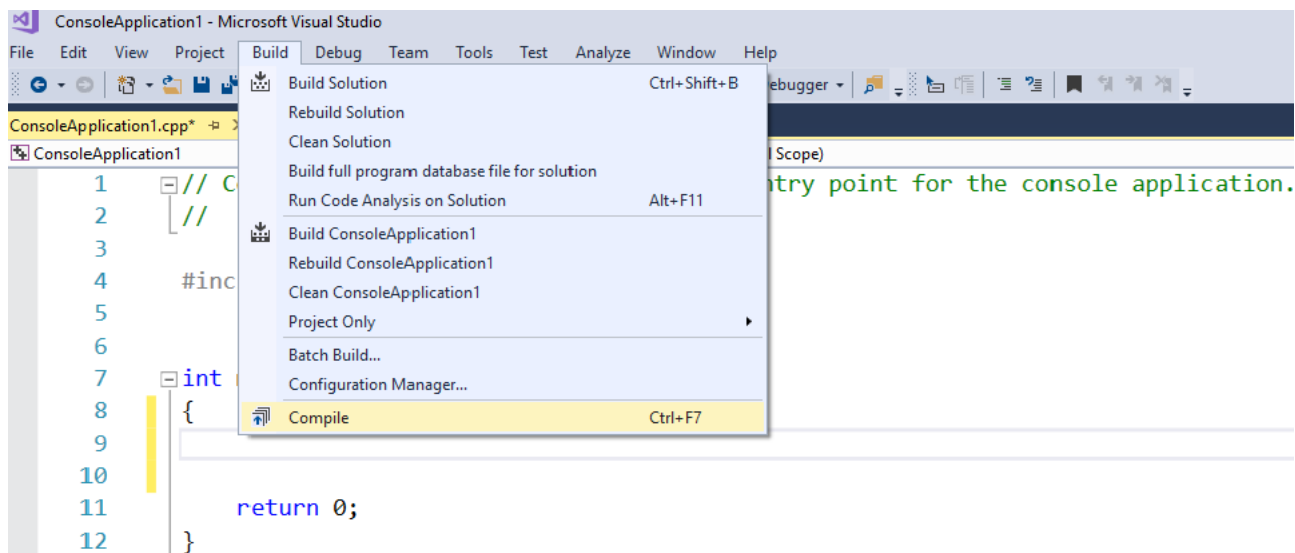
Функцията *main* се съхранява във файл с име, определено от програмиста и разширение *.cpp*. Освен този, са създадени още няколко файла, които съставляват текущия проект. Структурата на проекта т.е. файловете, които го изграждат могат да се видят и да се управляват чрез т.нар. *Solution Explorer* (фиг. 2.3).



Фиг. 2.3. *Solution Explorer*

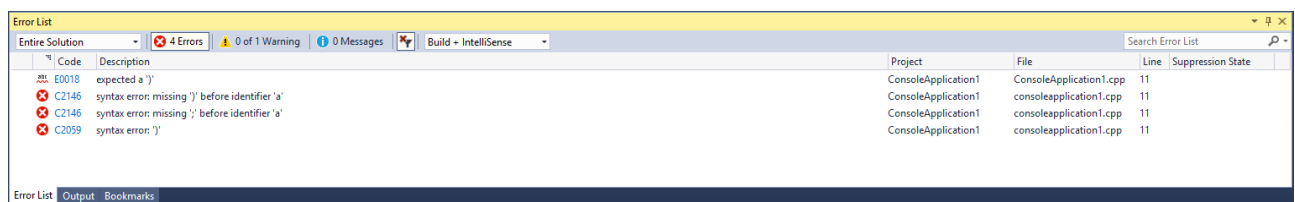
3.2. Компилиране и свързване на проекта

За да се компилиране и свързване на програмния код се използват група команди от основното меню **Build** (фиг.2.4). Чрез команда **Build/Compile** се компилира програмния код. Свързването е чрез команда **Build/Build (име на проект)** или чрез **Build/Build Solution**. Първата свързва само файловете в текущия проект, докато втората свързва всички проекти в текущия контейнер (solution). Ако програмният код не е компилиран, последните две команди стартират първо компилатора, а след това свързващия редактор.



Фиг. 2.4. Команди за компилиране и свързване на програмния код

Ако компилаторът открие синтактични грешки, програмният код не може да бъде компилиран. Съответно се извеждат откритите при компилацията грешки (фиг. 2.5). Необходимо е програмистът да редактира програмния код и отново да компилира. Това се повтаря докато не бъде създаден обектен и в последствие изпълним код.



Фиг. 2.5. Списък на откритите при компилиране грешки

3.3. Стартиране

За да се стартира изпълнението на програмния код се натиска клавишна комбинация Ctrl +F5. Така след изпълнението на програмния код екранът на конзолата не се затваря, а се изчаква да се натисне произволен клавиш. На екрана се изписва съобщението "Press any key to continue...". Ако изпълнението на програмния код се стартира чрез клавиш F5, след приключване на изпълнението, конзолата се затваря автоматично.

1. Аритметични типове данни

Основни типове данни в C/C++ са *аритметичните типове*: **int**, **char**, **float**, **double**.

Тип **char** служи за представяне на символни данни (константи и променливи). Размерът на типа е 1 байт (8 бита), което означава, че стойността на дадена променлива или константа от тип char е в границите на [-128, +127], тъй като типът е за знакови числа. При представяне на знаково число с 8 бита, най-старшият бит се използва като бит за знак: 0 за знак + и 1 за знак -, а останалите 7 бита са за стойността на числото. При

беззнаковите числа, за стойността на числото се използват всичките 8 бита. Числената стойност на променлива от тип *char* е *ASCII кода* на символа.

Тип *int* е предназначен за представяне на цели числа със знак. Обхвата на типа зависи от компилатора: за 16 бита (1 дума) е $[-2^{15}, +2^{15}-1]$, докато при 32 бита (1 двойна дума) е $[-2^{31}, +2^{31}]$.

Тип *float* служи за представяне на реални числа. Обхвата на променлива от тип *float* при 32-битово кодиране е $[\pm 10E-37, \pm 10E+37]$.

Чрез тип *double* се представят реални числа с двойна точност. При кодиране с 64 бита, обхвата на променлива от тип *double* е $[\pm 10E-307, \pm 10E+307]$.

Аритметичните типове могат да бъдат използвани със следните **модификатори**:

- *signed* - за знакови числа;
- *unsigned* - за беззнакови числа;
- *short* - за числа с единична точност;
- *long* - за числа с двойна точност.

Модификаторите променят (модифицират) типа на числото.

2. Променливи и константи. Дефиниции

Основна характеристика на всеки език за програмиране са типовете данни, които поддържа. Типът на константа или на променлива определя нейното съдържание т.е. от какъв вид е информацията: числов, символен, цели или реални числа и т.н.

Променливите са данни, чиито стойности се изменят по време на изпълнение на програмата, докато *константите* са данни, които не се променят по време на изпълнение на програмата.

Общият вид на дефиницията на променлива е следният:

тип име [=стойност][, име [=стойност],...];

Задължителни елементи на дефиницията са името и типа на променливата. Едновременно, дефинирането на променлива може да бъде съпроводено и с инициализация т.е. със задаване на начална стойност на променливата.

На един ред могат да бъдат дефинирани повече от една променливи, които имат един и същи тип.

Примери за дефиниране на променливи:

```
unsigned j;           // дефиниране на променлива j от тип unsigned int
int x,y;              // дефиниране на променливи x,y от тип int
char ch='Y';          // дефиниране на променлива ch с тип char и начална
                      // стойност 'Y'
float pi=3.14;        // дефиниране на променлива pi с тип float и начална
                      // стойност 3.14
unsigned char h;       // дефиниране на променлива h от тип unsigned char
int z=2,k;             // дефиниране на променливи z с начална стойност 2 и k
                      // от тип int
```

Подобно на променливите, константите имат тип, обозначават се чрез символни имена (идентификатори), но стойностите им са постоянни.

След дефиниране на константа, при всяко нейно срещане в програмата, компилаторът заменя символното име със стойността ѝ. Ако, в процеса на разработка на програмата се наложи промяна на параметър, определен чрез константа, единствената корекция в тази програма е замяна стойността на една константа. Ако, не

е използвана константа, а само стойност, в този случай, ще се наложат корекции навсякъде в програмата, където се среща параметърът.

В език C++ съществува начин за дефиниране на константи чрез използване на ключова дума **const**. Общият вид на дефиницията е следния:

const [тип] име=стойност;

Примери:

const int N=100;

const float PI=3.14;

Ако типът на константата не се зададе явно, използва се стойността ѝ за да се зададе тип по подразбиране:

const y=3; // типът на константата е int

const z=0.8 // типът на константата е float

В език C не съществува ключова дума **const**. За дефиниране на константи в C се използва директива на предпроцесора **#define**.

Примери:

#define N 100

#define PI 3.14

#define Y 3

3. Въвеждане и извеждане на данни

3.1. Функции printf() и scanf()

В C/C++ не са предвидени оператори за вход и изход, тъй като по правило те са машиннозависими. Вместо тях, са използвани функции за вход и изход, разпространявани като стандартна библиотека, заедно с компилатора на езика. Най-често използваните функции за въвеждане и извеждане на данни са: **scanf()** и **printf()**. Тъй като тези функции се считат за ненадеждни, в последните версии на Visual Studio се препоръчва вместо тях да се използват функции **scanf_s()** и **printf_s()**.

Функциите **scanf()** и **printf()** са декларирани в **stdio.h**, което означава, че за да бъдат използвани е необходимо да включване на заглавния файл:

#include <stdio.h>

Функция **printf()** служи за извеждане на данни и има следната декларация:

int printf("форматиращи параметри", списък_аргументи);

Форматиращите параметри са комбинация от: *низова константа*, която се извежда на екрана във вида, в който е записана и *низ форматни спецификации*, които не се показват на екрана, но служат за управление начина на извеждане на аргументите. В низовата константа могат да са включени и управляващи символи. Например '\n' - символ за нов ред.

Аргументите са променливи, константи или изрази. Функцията printf() извежда на екрана стойностите на аргументите, според зададените форматни спецификации. Всеки аргумент има свой форматиращ параметър т.е. броя на форматиращите параметри е равен на броя на аргументите.

Форматиращите параметри започват със символ %, последвани от модификатори и спецификации. Видовете форматни спецификации са:

%d – за десетично цяло число със знак;

%u – за десетично цяло число без знак;
%o – за цяло осмично число;
%x – за цяло шестнадесетично число, като се използват символи a, b, c, d, e, f;
%X – за цяло шестнадесетично число, като се използват символи A, B, C, D, E, F;
%f – за дробно-десетично число от тип float в F формат; (за дробно-десетично число от тип double в F формат е необходимо да се добави модификатор **l** т.е. видът на спецификацията е **%lf**)

%e – за дробно-десетично число в експоненциален (E) формат, като се използва символ **e**;

%E – за дробно-десетично число в експоненциален (E) формат, като се използва символ **E**;

%g – за дробно-десетично число в E или в F формат, като се използва символ **e**;

%G – за дробно-десетично число в E или в F формат, като се използва символ **E**;

%c – за единичен символ;

%s – за низ.

Заедно с форматните спецификации, във форматиращите параметри могат да бъдат добавени и различни модификатори. По този начин, общият вид на форматиращия параметър може да се представи като:

% [флагове][поле][.точност] [F|N|h|l|L] форматна_спецификация

където:

флагове са следните символи:

- задава ляво изравняване;

+ задава знак + пред положителните числа;

интервал положителните стойности започват с интервал, вместо с знак +;

определя добавяне на символ 0 пред осмичните числа и 0x или 0X пред шестнадесетичните.

поле – определя минималния брой на символите за извеждане;

точност – определя минималния брой символи след десетичната точка; използва се само за дробно-десетични числа;

F|N|h|l|L – определят дължината на аргумента, като:

N – близък указател;

F – далечен указател;

h – тип *short int*;

l – тип *long*;

L – тип *long double*.

Примери:

```
printf("I like C and C++."); //отпечатва низ
```

```
float x,y;
```

```
...
```

```
printf("Резултатът е %f", y); //отпечатва стойността на y
```



```
int a=3,b=8;
```

```
...
```

```
printf ("a+b=%d", (a+b)); //отпечатва резултата от сбора на a и b
```

Въвеждането на данни е чрез функция *scanf()*. Функцията има следната декларация:
int scanf("форматиращи параметри", списък_аргументи);

Форматните спецификации са същите, както при функция *printf()*.

Като аргументи се използват адресите на променливите, чиито стойности се въвеждат. Съответно, за целта се използва адресна операция &. Функцията *scanf()* връща броя на успешно обработени входни символи.

Примери:

```
int k;
```

```
printf("input value of k=");
```

```
scanf("%d", &k);
```

```
float r;
```

```
printf(" r=");
```

```
scanf("%f", &r);
```

II. Задачи за изпълнение

1. Да се създаде конзолно приложение, с което да се представят възможностите на функциите *printf_s()* и *scanf_s()* за въвеждане и извеждане на данни.
2. Да се създаде конзолно приложение за преобразуване на цяло число от десетична в шестнадесетична и осмична бройна система и обратно, като се използват форматните спецификации %d, %X, %x, %o.