

---

# C Programming for Engineers

---

2<sup>nd</sup> Edition

---

Mohamed Sobh

---

C Programming for Engineers, 2<sup>nd</sup> Edition

Copyright © 2011 by Dr. Mohamed Ali Sobh

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means without permission of the author.

## Preface

This book introduces C programming language from engineering point of view. The book covers basic and advanced C programming topics. The book covers following topics, data types, input output, control statements, basic mathematic and logic calculations, functions, structures, arrays, strings, pointers, memory management, file management, c projects and c libraries. Many illustrative examples are offered with each topic to clarify the idea and to show the engineering applications. Examples are provided with a complete code samples. The code samples are ready to compile and run.

This book introduces two chapters “Solving Engineering Problems Part 1” and “Part 2”. Those chapters provide many engineering problems then discuss the available programming techniques then provide a complete solution with source code.

Dr. Mohamed Ali Sobh  
Computer and System Department  
Faculty of Engineering  
Ain Shams University, Cairo, Egypt

<b>CHAPTER 1: INTRODUCTION.....</b>	<b>13</b>
<b>CHAPTER 2: CONTROLLING PROGRAM FLOW .....</b>	<b>43</b>
<b>CHAPTER 3: ADVANCED DATA TYPES .....</b>	<b>63</b>
<b>CHAPTER 4: FUNCTIONS .....</b>	<b>82</b>
<b>CHAPTER 5: SOLVING ENGINEERING PROBLEMS, PART 1 .....</b>	<b>104</b>
<b>CHAPTER 6: STRUCTURES .....</b>	<b>122</b>
<b>CHAPTER 7: POINTERS.....</b>	<b>146</b>
<b>CHAPTER 8: MEMORY AND FILES .....</b>	<b>185</b>
<b>CHAPTER 9: PROJECTS AND LIBRARIES .....</b>	<b>215</b>
<b>CHAPTER 10: SOLVING ENGINEERING PROBLEMS, PART 2 .....</b>	<b>233</b>
<b>CHAPTER 11: GUI/GDI PROGRAMMING FOR WINDOWS.....</b>	<b>271</b>
<b>APPENDIX A .....</b>	<b>317</b>

## Table of Contents

<b>CHAPTER 1: INTRODUCTION.....</b>	<b>13</b>
1.1. FIRST C PROGRAM “HELLO PROGRAM” .....	14
1.2. MAKING YOUR FIRST PROGRAM USING VISUAL STUDIO 2010.....	14
1.3. UNDERSTANDING “HELLO PROGRAM” .....	17
1.4. MAKING CALCULATIONS .....	19
1.5. VARIABLE NAME .....	24
1.6. COMMENTS .....	24
1.7. DATA TYPES .....	25
<i>Example 1: Printing the Number of Days in 10 Weeks .....</i>	26
<i>Example 2: Printing Decimal and Hexadecimal Numbers .....</i>	27
<i>Example 3: Calculate Circle Area.....</i>	27
<i>Example 4: Calculating Number of Days in n Weeks.....</i>	28
<i>Example 5: Printing ASCII Letters .....</i>	28
1.8. FORMATTING NUMBERS WHILE PRINTING AND SCANNING.....	30
1.9. DATA CONVERSION AND TYPE CASTING .....	31
1.10. MATHEMATICAL AND LOGICAL EXPRESSIONS .....	32
1.11. ADVANCED MATHEMATICAL EXPRESSIONS .....	35
<i>Example 1: Using Trigonometric Functions .....</i>	35
<i>Example 2: Calculate Triangle Side Length .....</i>	35
1.12. CODING CONVENTION.....	37
1.12.1. Indentation and Code Arrangement .....	37
1.12.2. Variable Naming .....	39
1.13. EXERCISES.....	40
<b>CHAPTER 2: CONTROLLING PROGRAM FLOW .....</b>	<b>43</b>
2.1. PROGRAM EXECUTION FLOW.....	44
2.2. CONDITIONS.....	45
<i>Example 1: Using Conditions .....</i>	45
2.3. IF STATEMENT.....	46
<i>Example 1: Calculate Circle Area or Circumference .....</i>	46
<i>Example 2: Comparing Three Numbers.....</i>	47
2.4. INLINE CONDITION .....	48
<i>Example 1: Calculate the Minimum of Two Numbers .....</i>	49
2.5. SWITCH STATEMENT .....	49
<i>Example 1: Calculate Circle Area or Circumference .....</i>	50
2.6. FOR STATEMENT.....	52
<i>Example 1: Printing Hello World in a Loop.....</i>	52
<i>Example 2: Calculate the Summation of values between 1 and 99.....</i>	53
<i>Example 3: Calculate the Average Students Degrees.....</i>	54
2.7. WHILE STATEMENT .....	55
<i>Example 1: Calculate the Summation of odd values between 1 and 99.....</i>	55
<i>Example 2: Calculate the Average Students Degrees.....</i>	56
2.8. DO...WHILE STATEMENT.....	56

<i>Example 1: Calculate Polynomial Value .....</i>	57
2.9. GOTO STATEMENT .....	57
2.10. CODING CONVENTION.....	58
<i>2.10.1. Indentation and Code Arrangement .....</i>	58
2.11. EXERCISES.....	59
<b>CHAPTER 3: ADVANCED DATA TYPES .....</b>	<b>63</b>
3.1. ARRAYS .....	64
<i>Example 1: Store and Print 10 Students Degrees .....</i>	65
<i>Example 2: Calculate Polynomial Value for a Set of Inputs .....</i>	66
<i>Example 3: Calculate the Summation of Array.....</i>	67
<i>Example 4: Calculate the Maximum of the Array.....</i>	68
3.2. 2D ARRAYS.....	68
<i>Example 1: Scan 3x3 Matrix .....</i>	69
<i>Example 2: Calculate and Print the Transpose of 3x3 Matrix .....</i>	70
3.3. STRINGS .....	71
<i>Example 1: Printing Array of Strings .....</i>	74
<i>Example 2: Copy String to String .....</i>	74
<i>Example 3: Adding String to String.....</i>	75
<i>Example 4: Changing String Case .....</i>	75
<i>Example 5: Finding the String Length.....</i>	76
<i>Example 6: Comparing Two Strings.....</i>	76
<i>Example 7: Converting String to Integer Value .....</i>	77
3.4 EXERCISES.....	79
<b>CHAPTER 4: FUNCTIONS .....</b>	<b>82</b>
4.1. INTRODUCTION .....	83
4.2. FUNCTION DEFINITION .....	85
4.3. PARAMETERS TYPES.....	87
<i>Example 1: Calculate the Factorial.....</i>	87
<i>Example 2: Calculate the Minimum Value of any Given Array.....</i>	88
<i>Example 3: Finding a Name in a Set of Names.....</i>	89
<i>Difference between Passing Single Values and Arrays.....</i>	89
4.4. VARIABLES SCOPE.....	91
4.4.1. Local Variables .....	91
4.4.2. Global Variables .....	92
<i>Example 4: Using Global Variables.....</i>	92
4.4.3. Static Variables.....	93
<i>Example 5: Using Static Variables .....</i>	93
4.5. CALLING MECHANISM .....	94
4.5. RECURSION .....	95
<i>Example 6: Infinite Printing Loop .....</i>	96
<i>Example 7: Finite Printing Loop.....</i>	97
4.5. MACROS .....	97
<i>Example 8: Percentage Macro .....</i>	97
<i>Example 9: Exchange Macros.....</i>	98
4.9. CODING CONVENTION.....	99
<i>4.9.1. Local, Global, Static Variables Naming .....</i>	99

<i>4.9.2. Function Naming</i> .....	99
4.8. EXERCISES.....	100
<b>CHAPTER 5: SOLVING ENGINEERING PROBLEMS, PART 1 .....</b>	<b>104</b>
5.1. INTRODUCTION .....	105
5.2. SORTING A LIST OF VALUES .....	105
<i>5.2.1. Problem Statement</i> .....	105
<i>5.2.2. Understand the Solution</i> .....	105
<i>5.2.3. Design the Program</i> .....	107
<i>5.2.4. Write the Program</i> .....	108
<i>5.2.5. Put the Solution in a Function</i> .....	108
<i>5.2.6. Extending the Solution to Sort String Values.</i> .....	109
5.3. CALCULATING POLYNOMIAL VALUE.....	110
<i>5.3.1. Problem Statement</i> .....	110
<i>5.3.2. Design the Program</i> .....	110
<i>5.3.3. Write the Program</i> .....	111
<i>5.3.4. Optimize the Solution</i> .....	111
<i>5.3.5. Apply the Optimization to the Program</i> .....	112
5.4. EVALUATING SERIES.....	113
<i>5.4.1. Problem Statement</i> .....	113
<i>5.4.2. Design the Program</i> .....	113
<i>5.4.3. Write the Finite Series Program</i> .....	114
<i>5.4.4. Optimize the Finite Series Program</i> .....	114
<i>5.4.5. Write the Infinite Series Program</i> .....	115
5.5. PERFORMING MATRIX CALCULATIONS.....	117
<i>5.5.1. Problem Statement</i> .....	117
<i>5.5.2. Understand the Solution</i> .....	117
<i>5.5.3. Write the Matrix Program</i> .....	118
5.6. EXERCISE .....	120
<b>CHAPTER 6: STRUCTURES .....</b>	<b>122</b>
6.1. INTRODUCTION .....	123
<i>Example 1: Employee Sort without Structures</i> .....	123
6.2. DEFINING AND USING STRUCTURE VARIABLES.....	126
<i>Example 2: Define and Use Structure Variable</i> .....	126
<i>Example 3: Copying Structure Variable Contents to another Variable</i> .....	127
<i>Example 4: Employee Sort with Structures</i> .....	128
6.3. INITIALIZING STRUCTURE VARIABLES.....	131
6.4. NESTED STRUCTURE DEFINITION .....	131
<i>Example 5: Employee Sort with Nested Structures</i> .....	132
6.5. USING STRUCTURES WITH FUNCTIONS.....	134
<i>Example 6: Read and Print Employee and Date Values using Functions</i> .....	134
<i>Example 7: Comparing Two Date Values using Functions</i> .....	136
<i>Example 8: Simplified Employee Sort Program using Functions</i> .....	136
6.6. SUPPORTING COMPLEX NUMBERS .....	137
<i>Example 9: Adding Two Complex Numbers</i> .....	137
<i>Example 10: Adding Two Complex Numbers with Functions</i> .....	138
6.7. ENUM .....	139

<i>Example 11: Personal Data .....</i>	139
<b>6.8. UNIONS.....</b>	<b>140</b>
<i>Example 12: Using Union .....</i>	140
<b>6.8. CODING CONVENTION.....</b>	<b>142</b>
<i>6.8.1. Structure, Union Naming.....</i>	142
<i>6.8.2. Structure, Union Members Naming .....</i>	142
<i>6.8.2. enum Members Naming.....</i>	142
<b>6.10. EXERCISE .....</b>	<b>143</b>
<b>CHAPTER 7: POINTERS.....</b>	<b>146</b>
<b>7.1. INTRODUCTION .....</b>	<b>147</b>
<i>Example 1: Printing Employee Data without Pointers .....</i>	147
<i>Example 2: Printing Employee Data with Pointers.....</i>	148
<b>7.2. ADDRESS OF VARIABLE .....</b>	<b>149</b>
<i>Example 3: Printing Variables Addresses and Values.....</i>	150
<b>7.3. ADDRESS OF ARRAY .....</b>	<b>151</b>
<i>Example 4: Printing Array of Elements Addresses and Values .....</i>	152
<b>7.4. TOTAL MEMORY USED BY PROGRAM VARIABLES.....</b>	<b>153</b>
<i>Example 5: Calculating Structure Size using sizeof() directive.....</i>	155
<i>Example 6: Calculating Number of Array Elements using sizeof() directive .....</i>	155
<b>7.5. POINTER TO VARIABLE.....</b>	<b>156</b>
<i>Example 7: Using Pointer to Variable.....</i>	156
<b>7.6. POINTER TO ARRAY.....</b>	<b>158</b>
<i>Example 8: Using Pointer to Array .....</i>	158
<i>Example 9: Average of Weights .....</i>	159
<b>7.7. POINTER TO STRUCTURE .....</b>	<b>161</b>
<i>Example 10: Using Pointer to Structure .....</i>	161
<b>7.8. POINTERS AND FUNCTIONS.....</b>	<b>163</b>
<i>Example 11: Fast Data Transfer Using Pointers .....</i>	163
<i>Example 12: Definition of Several Outputs for Function using Pointers.....</i>	166
<b>7.9. PASSING ARRAYS AND POINTERS TO FUNCTIONS.....</b>	<b>168</b>
<b>7.10. FUNCTION PARAMETERS TYPES.....</b>	<b>169</b>
<b>7.11. POINTER DATA TYPE .....</b>	<b>170</b>
<b>7.12. DEALING WITH POINTER ADDRESS VALUE .....</b>	<b>171</b>
<b>7.13. POINTER WITH UNKNOWN TYPE (VOID*) .....</b>	<b>172</b>
<i>Example 13: Universal Compare with void Pointers .....</i>	173
<b>7.14. POINTER TO POINTER.....</b>	<b>174</b>
<i>Example 14: Using Pointer to Pointer .....</i>	175
<i>Example 15: Finding Employees with Maximum and Minimum Salaries.....</i>	175
<b>7.15. NULL AND UNASSIGNED POINTERS .....</b>	<b>177</b>
<b>7.16. POINTER TO FUNCTION.....</b>	<b>177</b>
<i>Example 16: Using Pointer to Function .....</i>	177
<b>7.17. ADVANCED STRING MANIPULATION WITH POINTERS.....</b>	<b>179</b>
<i>Example 17: Playing with Text .....</i>	179
<i>Example 18: Tokenizing Strings.....</i>	180
<b>7.18. CODING CONVENTION.....</b>	<b>181</b>
<i>7.18.1. Pointer Naming .....</i>	181
<b>7.18 EXERCISE .....</b>	<b>182</b>

<b>CHAPTER 8: MEMORY AND FILES .....</b>	<b>185</b>
8.1. MEMORY.....	186
8.2. STATIC MEMORY.....	189
<i>Example 1: Average Temperature using Static Memory.....</i>	189
8.3. DYNAMIC MEMORY.....	190
<i>Example 2: Average Temperature using Dynamic Memory.....</i>	190
<i>Example 3: Simple String Replace Implementation.....</i>	191
8.4. MEMORY FUNCTIONS .....	192
<i>Example 4: Comparing and Exchanging Two Arrays.....</i>	193
<i>Example 5: Converting Student Record to/from Text.....</i>	193
8.5. FILES.....	195
<i>Example 6: Creating Folders and Files.....</i>	195
<i>Example 7: Writing and Reading Characters .....</i>	196
<i>Example 8: Writing and Reading Line of Strings .....</i>	198
<i>Example 9: Writing and Reading Formatted Strings and Numbers .....</i>	200
<i>Example 10: Writing and Reading Binary Data .....</i>	202
<i>Example 11: Writing and Reading Structures .....</i>	205
<i>Example 12: Random Access Files .....</i>	206
<i>Example 13: Student Database Example.....</i>	208
8.6. EXERCISE .....	213
<b>CHAPTER 9: PROJECTS AND LIBRARIES .....</b>	<b>215</b>
9.1. INTRODUCTION .....	216
9.2. C PROJECT .....	216
<i>Example 1: Student Database Project using Visual Studio 2008 .....</i>	222
9.3. LIBRARIES .....	223
<i>Example 2: Building and Using Static Library using Visual Studio 2008.....</i>	224
<i>Example 3: Building and Using Dynamic Library using Visual Studio 2008.....</i>	226
9.4. COMMAND LINE .....	229
<i>Example 4: Printing Default Parameters of Command Line .....</i>	229
<i>Example 5: Command Line Sort.....</i>	229
9.5. EXERCISE .....	231
<b>CHAPTER 10: SOLVING ENGINEERING PROBLEMS, PART 2 .....</b>	<b>233</b>
10.1. INTRODUCTION .....	234
10.2. DYNAMIC LINKED LISTS.....	234
10.2.1. <i>Problem Statement .....</i>	234
10.2.2. <i>Understanding the Solution .....</i>	235
10.2.3. <i>Write the Program .....</i>	236
10.3. MATRIX ALGEBRA.....	241
10.3.1. <i>Problem Statement .....</i>	241
10.3.2. <i>Understand the Solution .....</i>	242
10.3.3. <i>Write the Program .....</i>	242
10.4. FINDING ROOTS .....	247
10.4.1. <i>Problem Statement .....</i>	247
10.4.2. <i>Understanding the Solution .....</i>	247
10.4.3. <i>Write the Program .....</i>	248

10.5. ADVANCED ROOT FINDING SOLUTIONS.....	250
10.5.1. Complex Roots.....	250
10.5.2. Finding All Roots.....	250
10.5.3. Bisection Method .....	250
10.5.4. False Position Method .....	253
10.5.5. Newton Raphson .....	254
10.6. NUMERICAL DIFFERENTIATION .....	256
10.6.1. Problem Statement .....	256
10.6.2. Understanding the Solution .....	256
10.6.3. Write the Program .....	256
10.7. NUMERICAL INTEGRATION .....	258
10.7.1. Problem Statement .....	258
10.7.2. Understanding the Solution .....	258
10.7.3. Write the Program .....	259
10.8. SOLVING FIRST ORDER DIFFERENTIAL EQUATIONS .....	261
10.8.1. Problem Statement .....	261
10.8.2. Understanding the Solution .....	261
10.8.3. Write the Program .....	264
10.9. ADVANCED ODE SOLUTIONS.....	266
10.9.1. Runge-Kutta 3 Method.....	266
10.9.2. Runge-Kutta 4 Method.....	266
10.9.3. Solving Higher Order Differential Equations .....	267
10.10. NUMERICAL ERRORS .....	268
10.10.1. Number Truncation .....	268
10.10.2. Mathematical Approximation .....	268
10.11. EXERCISE .....	269
<b>CHAPTER 11: GUI/GDI PROGRAMMING FOR WINDOWS .....</b>	<b>271</b>
11.1. BASIC GUI/GDI PROGRAMMING .....	272
11.1.1. Making Empty Window Program.....	272
11.1.2. Understanding the WINGUI Code .....	274
11.1.3. Windows APIs data types.....	275
11.1.4. Modify Window Title .....	276
11.1.5. Modify Window Size.....	276
11.1.6. Make a Full Screen Window .....	277
11.1.7. Getting Help on Windows APIs.....	278
11.1.8. Draw Simple Graphics .....	278
11.1.9. Draw More Graphics .....	279
11.1.10. Changing the Brush, the Pen and the Font .....	280
11.1.11. Draw Formatted Text .....	285
11.1.12. Capturing Mouse Events .....	286
11.1.13. Capturing Keyboard Events .....	288
11.1.14. Making Animation.....	290
11.2 GDI PROGRAMMING .....	294
11.2.1. Bouncing Balls Screen Saver.....	294
11.2.2. Simple Drawing .....	296
11.2.3. Dealing with Bitmaps .....	299
11.2.4. Working with Regions .....	300

11.3 GUI PROGRAMMING .....	302
11.3.1. <i>Showing Windows Messages</i> .....	302
11.3.2. <i>Working with Controls</i> .....	303
11.3.3. <i>Basic Calculator</i> .....	306
11.3.4. <i>Standard Calculator</i> .....	308
11.4 ADVANCED GUI/GDI PROGRAMMING TOPICS .....	312
11.4.1. <i>Drawing Polylines and Polygons</i> .....	312
11.4.2. <i>Create Applications with Several Windows</i> .....	313
11.5 EXERCISES.....	316
<b>APPENDIX A .....</b>	<b>317</b>



## Chapter 1: Introduction

## 1.1. First C Program “Hello Program”

C programs consist of lines of text “Code”. Those lines contain statements/instructions telling the computer what to do. The order of the lines in the program is the same order in which the lines are read and executed by the computer.

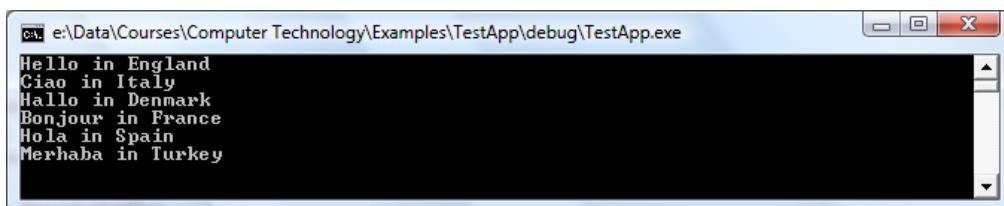
Following example shows a very simple C program that prints the word “hello” on the screen.

```
#include "stdio.h"

void main()
{
    printf("Hello in England\r\n");
    printf("Ciao in Italy\r\n");
    printf("Hallo in Denmark\r\n");
    printf("Bonjour in France\r\n");
    printf("Hola in Spain\r\n");
    printf("Merhaba in Turkey\r\n");
}
```

To run above program you must place the code in a file with .c or .cpp extension, and then use any development environment (Microsoft Visual Studio in Windows, cc/gcc commands in UNIX) to compile and run the program. Compilation means understanding the program then identifying the writing errors (Syntax Errors), then producing the final program using the machine language. Know that there is no way to run the C code directly, because the computer understands only programs written in machine language.

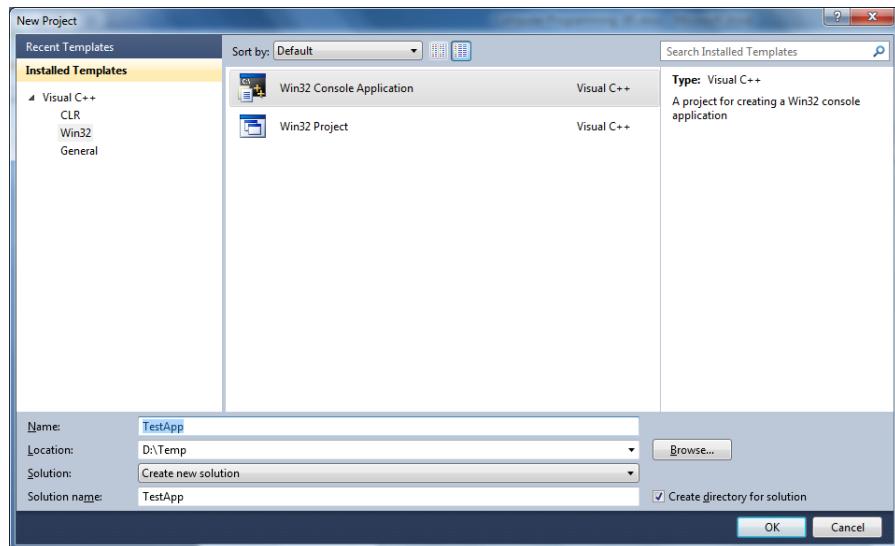
Hello program should run as shown in the following figure. The black screen is called the “**Consol Window**”. Programs uses consol window called **Consol Applications**. Consol application works on text mode there is no graphics support, also it interacts with user using keyboard there is no mouse support.



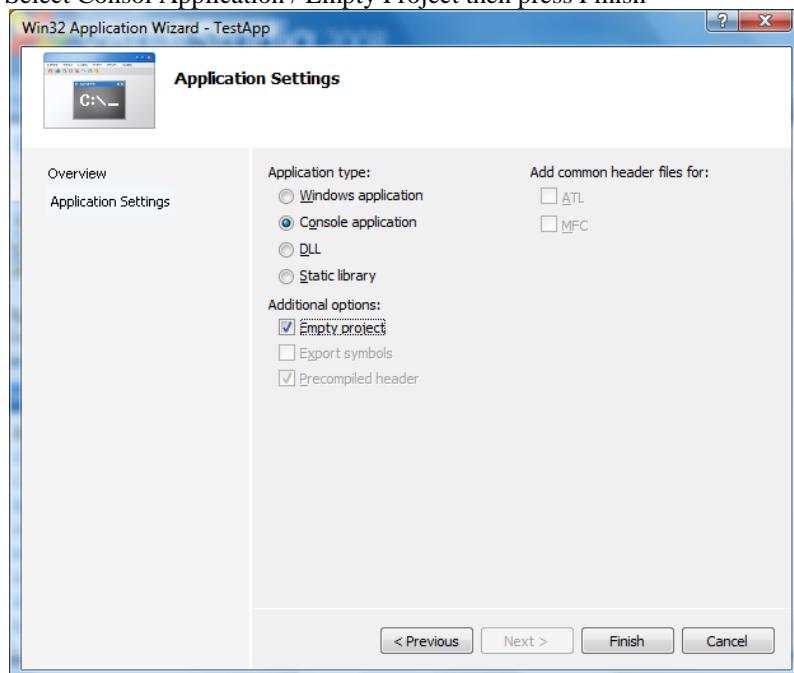
## 1.2. Making Your First Program Using Visual Studio 2010

Make sure you are installing Microsoft Visual Studio 2010 correctly.

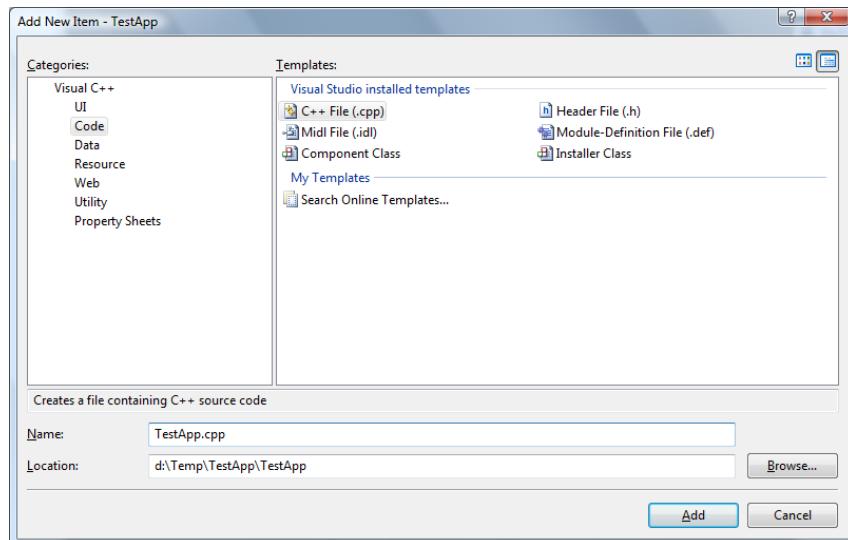
1. Start Microsoft Visual Studio 2010
2. Open menu item File/New/Project
3. Select Visual C++/Win32/ Win32 Consol Application
4. Select program path and name “TestApp” then press OK



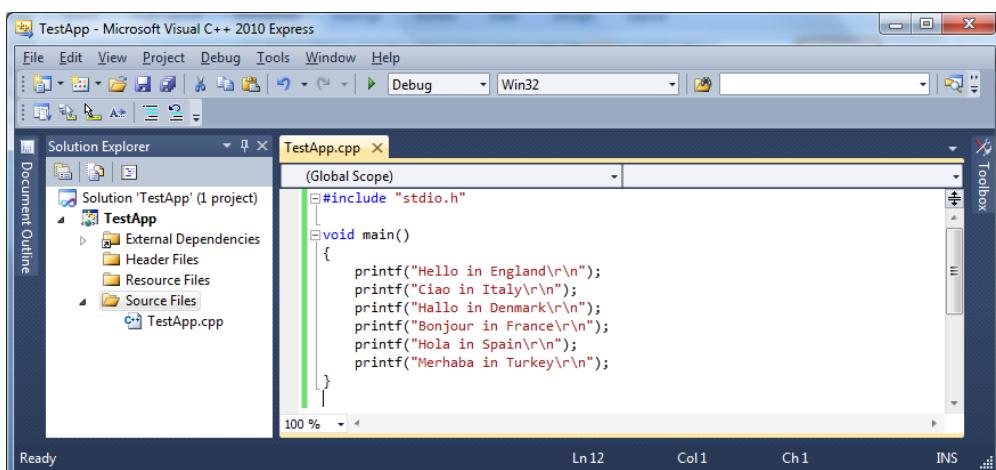
5. Press Next
6. Select Consol Application / Empty Project then press Finish



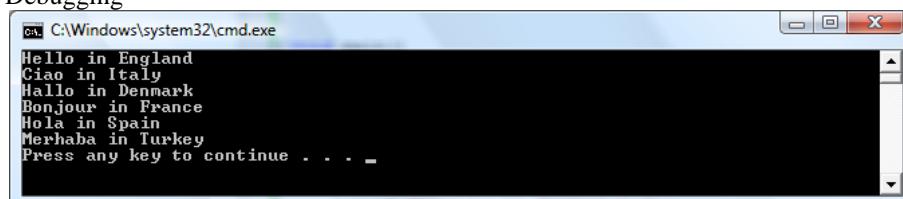
7. Select from menu Project/Add New Item
8. Select Visual C++/Code/C++ File (.cpp)
9. Enter the file name "TestApp.cpp", and then press Add



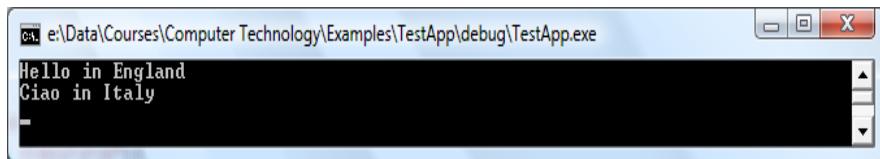
10. Now the file is TestApp.cpp file is opened
11. Write hello program code in the opened file



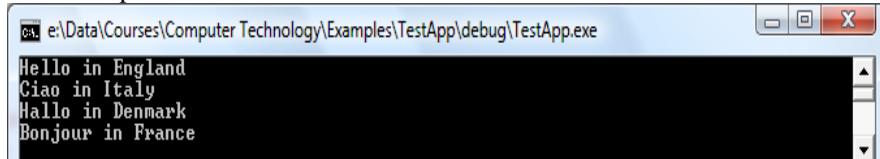
12. To run the program press Ctrl+F5 or select from menu Debug/Start without Debugging



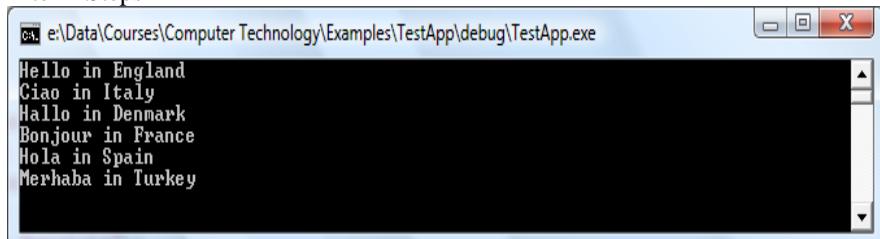
13. To run the program step by step press F10 or select from menu Debug/Start Debugging
14. To execute one step forward press F10 or select from menu Debug/Step Over
15. After each step you can watch the consol window to see the changes



After 2 Steps



After 4 Steps



After 6 Steps

### 1.3. Understanding “Hello Program”

Hello program consists of the following sections:

#### Header Section

```
#include "stdio.h"
```

This section tells the compiler that the header file “**stdio.h**” contains all required C functions by the program. Simply the “**printf**” function appears in the next section is defined in “**stdio.h**” file. Without the definition of the header file “**stdio.h**” the compiler is unable to know the meaning of the “**printf**” function.

In this section specify all required header files. Header file must end with (.h) extension. You can add one or more lines to include one or more header files.

Important, you are not free to write the header section in the way you want. You must follow the C style (**syntax**) to write this section. In another way you must, place the # letter followed directly by the word **include**, then leave an empty space, then write the name of the header “**stdio.h**” between two double quotes. Following examples are wrong representations of the include line.

<b>include "stdio.h"</b>	Wrong	missing #
<b>#Include "stdio.h"</b>	Wrong	Wrong case (I is capital), C is case sensitive

#include "stdio.h"	Wrong	Missing double quote
--------------------	-------	----------------------

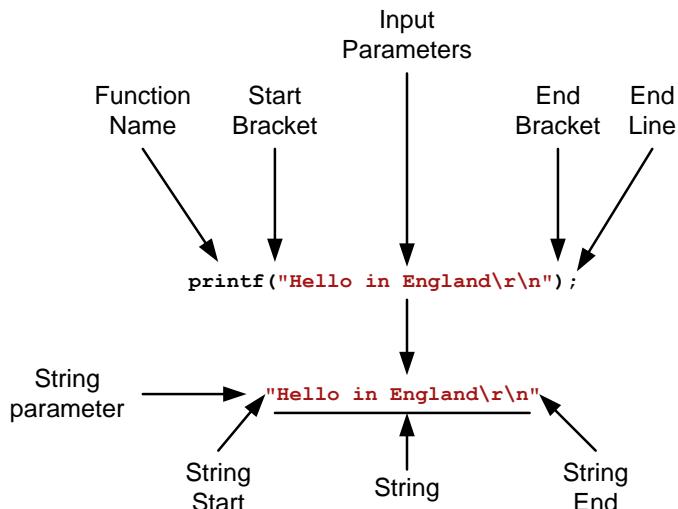
## Main Function

```
void main()
{
    printf("Hello in England\r\n");
    printf("Ciao in Italy\r\n");
    printf("Hallo in Denmark\r\n");
    printf("Bonjour in France\r\n");
    printf("Hola in Spain\r\n");
    printf("Merhaba in Turkey\r\n");
}
```

**main** function is the core part of any C program. **main** function consists of a set of lines each ends with a semicolon character “;”. Each line performs a single operation. In the above example the main consists of a several lines performing different print operations. The operations should be executed with the same order. Which means the computer should prints “Hello in England” first then prints “Ciao in Italy”.

**main** function should be written exactly as shown in the above program. Its body should starts with a brace “{” and ends with a brace “}”, any line between those braces belongs to the **main** function.

## Print Function



`printf` is a built in C function, it prints the provided string at the consol screen. As shown in the above figure the function starts and ends with a rounded braces ‘(’ and ‘)’, also it ends

with a semicolon ‘;’. Between the braces you should provide the string to be printed. The printed string should start and end with a double quotation ““”. The characters ‘\r\n’ at the end of the string order the computer to print new text values starting from the next line. ‘\r’ represents the carriage return and the ‘\n’ represents the new line. Alternatively ‘\n’ can be used alone; however ‘\r\n’ is commonly used, especially by windows programs.

## 1.4. Making Calculations

Following C example performs some simple calculations then prints the result. The program calculates the area of a rectangle then prints the final result.

```
#include "stdio.h"

void main()
{
    int width = 100;
    int height = 50;
    int area;

    area = width * height;

    printf("Rectangle area equals %d", area);
}
```

Above program should display the following output.



At the beginning of the **main** function, three names (**variables**) are defined, the **width**, the **height** and the **area**. This means that the computer will allocate three places in the computer memory and assign those names to those places. The variable is used to store numeric values. You can alter the value by many ways, for that reason it is called a variable.

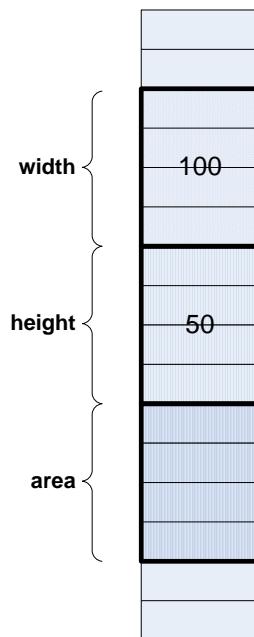
You can **initialize** the variable with an initial value at the beginning of the program (like **width** = 100, **height** = 50), otherwise you can leave it uninitialized (like **area**). Know that **area** variable take its value using the expression (**area** = **width** \* **height**).

```
#include "stdio.h"

void main()
{
    int width = 100;
    int height = 50;
    int area;

    area = width * height;

    printf("Rectanngle area equals %d\r\n", area);
}
```

Computer  
Memory

The program works as follows:

1. Define **width** variable and initialize it with 100
2. Define **height** variable and initialize it with 50
3. Define **area** variable leave it uninitialized
4. Calculate the value of the mathematical expression (**width \* height**)
5. Assign the result to the **area** variable
6. Read the string “**Rectangle area equals %d\r\n**” then replace the “**%d**” with the value in the variable **area**
7. Prints the final string “**Rectangle area equals 5000**” then moves to the next line.

C language is flexible; you can write the same program in many different ways. Following programs are similar to the above program and provides the same output.

Alternative 1:

```
#include "stdio.h"

void main()
{
    int w;
    int h;
    int a;
```

```
w = 100;
h = 50;
a = w * h;

printf("Rectangle area equals %d\r\n", a);
}
```

Explanation:

- If you did not initialize the variable at definition line, you can assign it with a value later.
- You can use other names for the variables.

Alternative 2:

```
#include "stdio.h"

void main()
{
    int width;
    int height;

    width = 100;
    height = 50;

    printf("Rectangle area equals %d\r\n", width * height);
}
```

Explanation: You can supply directly the multiplication result to the **printf** function without using the **area** variable as a temporary storage for the expression result.

Alternative 3:

```
#include "stdio.h"

void main()
{
    printf("Rectangle area equals %d\r\n", 100 * 50);
}
```

Explanation: You can directly supply the values of the width and the height then multiply and provide the result to the **printf** function.

Think? if you need a program that takes the width and the height from the user instead of using fixed values.

It is clear that each time the program starts, it should ask the user to supply the width and the height values. User will use the keyboard to provide the values. Following C program illustrates this idea.

```
#include "stdio.h"

void main()
{
    int width;
    int height;
    int area;

    printf("Please Enter the Rectangle width:");
    scanf("%d", &width);

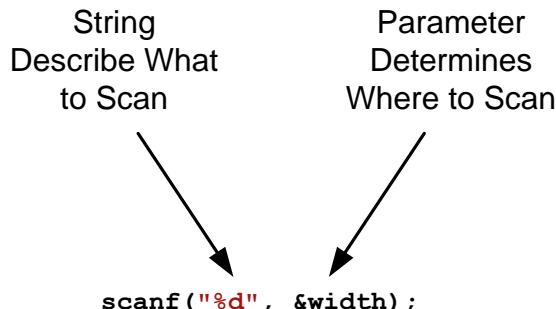
    printf("Please Enter the Rectangle height:");
    scanf("%d", &height);

    area = width * height;

    printf("Rectangle area equals %d\r\n", area);
}
```

The **scanf** function used to capture user input using the keyboard. Following steps illustrates how the statement “**scanf(“%d”, &height)**” works:

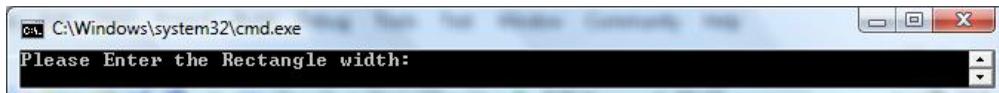
1. Stops the program and wait for user to press the keyboard buttons.
2. When the user presses any printable button it displays the associate letter on the screen.
3. It repeats (Step 2) until the user press the **Carriage Return** button
4. Converts the supplied letters into a number and store it in the **width** variable



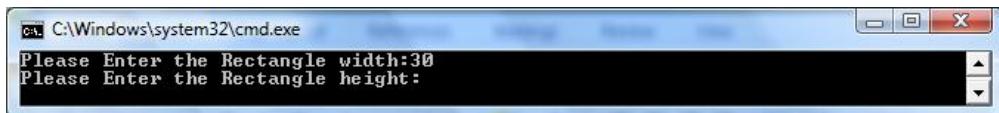
The “**%d**” string indicates that **scanf** will take a numeric value from user. The numeric value should be integer (no fractions) and have no letters. If the user supplies a letter by mistake the **scanf** will produce an error and closes the program. If the user supplies a real value the **scanf** will take the integer part and ignore the fraction part.

The & operator in “**&width**” statement provides the **scanf** function with the location or the address of the variable width inside the computer memory. **scanf** fills this address with the supplied value.

The final program should work as follows:



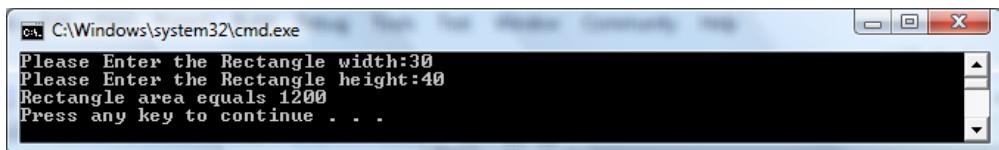
The program asks for the width



The user press following keyboard buttons “3” then “0” then “Carriage Return”

The program stores the “30” value in the Width variable

The program asks again for the height



The user press following keyboard buttons “4” then “0” then “Carriage Return”

The program stores “40” value in the Height variable

The program calculates the area

The program prints the result

Above program can be written using different ways, following programs shows some alternatives.

Alternative 1:

```
#include "stdio.h"

void main()
{
    int width;
    int height;
    int area;

    printf("Please Enter the Rectangle width and height:");
    scanf("%d %d", &width, &height);

    area = width * height;

    printf("Rectangle area equals %d\r\n", area);
}
```

Explanation: You can scan 2 numbers using a single scanf command.

Alternative 2:

```
#include "stdio.h"

void main()
{
    int width;
    int height;

    printf("Please Enter the Rectangle width and height:");
    scanf("%d %d", &width, &height);

    printf("Rectangle area equals %d\r\n", width * height);
}
```

Explanation: You can supply the multiplication result directly to the printf function; there is no need to define the area variable.

It appears that width and height variables are required by above program, because they are used to store the supplied user values.

## 1.5. Variable Name

Variable name can be any set of letters and numbers of a length up to 256 characters. Following constraints must be respected:

- Do not use any reserved keyword in C like (void, include, int)
- Do not use space or any special character inside variable name except “\_”.
- Do not start with a number

Correct variable names	M n Values	m_name counter name1	name2 min_value
Wrong variables names	Min value max>name void	5names printf	min-value

## 1.6. Comments

Sometimes programmers need to add some notes beside their code. Those notes are very important to describe the code and to clarify complex operation. Notes or comments can be added in two ways as shown in the following example.

```
#include "stdio.h"

void main()
{
```

```

double temprature;

//Supply the temprature in Fahrenheit
printf("Enter the temprature in Fahrenheit : \r\n");
scanf("%lf", &temprature);

/*Convert temprature from
Fahrenheit to Celsius */
temprature = (temprature - 32.0) * 5.0/9.0;

//prints the
//result
printf("The temprature in Celsius is %lf\r\n",
       temprature);
}

```

## 1.7. Data Types

Computer programs can process only numeric values. Letters and graphics are represented by numbers ('A' letter has a numeric code of 65).

When the program prints "Hello" on the screen it means that it sends the associated numeric codes for each letter to the operating system. The operating system draws the associated letter shape.

There are two basic numeric values:

- **Integer Values:** numeric values without the fraction part (Ex: 1, 2987 and -3893). Integer values are divided into:
  - **Signed Values:** numeric values that support positive and negative values.
  - **Unsigned Values:** numeric values that support positive values only.
- **Real Values:** numeric values with the fraction part (Ex: 23.938 and -1.0073).

Computer performs Integer calculation faster than Real calculations. Integer values are commonly used if there is no need for fraction parts. If the fraction part is important Real numbers must be used. Real numbers are commonly used to perform scientific calculation.

Either Integer or Real values have different representations. Each representation has different size and range. Numeric types with large range occupy larger space in memory. Programmer should use the suitable variable type which provides the maximum calculation speed and uses the minimum memory storage.

For example if the **Unsigned Integer** value uses one byte (8 bits) to hold the numeric value:

- The minimum value is  $(00000000)_2 = (0)_{10}$ .
- The maximum value is  $(11111111)_2 = (2^8 - 1)_{10} = (255)_{10}$ .

Another example if the **Signed Integer** value uses one byte (8 bits) to hold the numeric value. If Tow's Complement method is used to represent the negative values:

- The minimum value is  $(10000000)_2 = -(10000000)_2 = -(2^7)_{10} = -(128)_{10}$ .
- The maximum value is  $(01111111)_2 = (2^7 - 1)_{10} = (127)_{10}$ .

Another example if the **Unsigned Integer** value uses two byte (16 bits) to hold the numeric value:

- The minimum value is  $(0000000000000000)_2 = (0)_{10}$ .
- The maximum value is  $(1111111111111111)_2 = (65535)_{10}$ .

Another example if the **Signed Integer** value uses four byte (32 bits) to hold the numeric value. If Tow's Complement method is used to represent the negative values:

- The minimum value is  $(80000000)_{16} = - (7FFFFFFF)_{16} = - (2^{31})_{10} = - (2147483648)_{10}$ .
- The maximum value is  $(FFFFFFFF)_{16} = (2^{31}-1)_{10} = (2147483647)_{10}$ .

Following data types are supported by C language.

Data Type	Major Type	Size (Bytes)	Precision	Range
Char	Integer	1	1	-128 to 127
unsigned char	Integer	1	1	0 to 255
Short	Integer	2	1	-32,768 to 32,767
unsigned short	Integer	2	1	0 to 65,535
*int	Integer	4	1	-2,147,483,648 to 2,147,483,647
*unsigned int	Integer	4	1	0 to 4,294,967,295
Long	Integer	4	1	-2,147,483,648 to 2,147,483,647
unsigned long	Integer	4	1	0 to 4,294,967,295
Float	Real	4	3.4E-38	+/- 3.4E38
Double	Real	8	1.7E-308	+/- 1.7E308

\* Precision means; the smallest fraction value

\*\* int, this data type called the machine dependent data type, which means its size and range vary from a machine type to another machine type (EX: in 8 bit computers **int** is 1 byte, in 16 bit computers **int** is 2 bytes, in 32 bit computers **int** is 4 bytes, in 64 bit computers **int** is 8 bytes).

Know that (32 Bits computers) means the principle data unit size in those computers are 32 bit, which mean the computer is designed and optimized to process 32 bit values.

Following examples illustrate how to use different data types:

### Example 1: Printing the Number of Days in 10 Weeks

```
#include "stdio.h"

void main()
{
    int nWeeks = 10;
    int nDays;

    nDays = 7 * nWeeks;
```

```

    printf("Number of days in %d weeks is %d\r\n",
           nWeeks, nDays);
}

```

“%d” tells the **printf** it will print an **int** variable.  
The first “%d” is associated with the **nWeeks** variable.  
The second “%d” is associated with the **nDays** variable.

### Example 2: Printing Decimal and Hexadecimal Numbers

```

#include "stdio.h"

void main()
{
    int x = 160;
    int y = 0x2FCA;

    printf("x y in decimal x = %d, y = %d\r\n", x, y);
    printf("x y in hexdecimal x = %x, y = %X\r\n", x, y);
}

```

This example shows how to assign hexadecimal values using the modifier (0x) before the hexadecimal number.

“%d” tells the **printf** it will print an **int** variable in a decimal format.  
“%x” tells the **printf** it will print an **int** variable in a hexadecimal format.

### Example 3: Calculate Circle Area

```

#include "stdio.h"

void main()
{
    float radius = 2.0f;
    float area;

    area = (3.1416f) * radius * radius;

    printf("Circle area equals %f\r\n", area);
}

```

“%f” tells the **printf** it will print a **float** variable.  
The ‘f’ letter after value “2.0f” is used to force the **float** representation for this value. If the ‘f’ modifier is not used the **double** representation is used by default.

**Example 4: Calculating Number of Days in n Weeks**

```
#include "stdio.h"

void main()
{
    char nDaysPerWeek = 7;
    long nWeeks;
    long nDays;
    printf("Enter the number of weeks");
    scanf("%d", &nWeeks);
    nDays = nWeeks * nDaysPerWeek;
    printf("Number of days in %d weeks is %d\r\n",
           nWeeks, nDays);
}
```

**Example 5: Printing ASCII Letters**

```
#include "stdio.h"

void main()
{
    char letter1 = 'H';
    char letter2 = 'e';
    char letter3 = 108;
    char letter4 = 108;
    char letter5 = 0x6F;

    printf("%c%c%c%c%c\r\n",
           letter1, letter2, letter3, letter4, letter5);
}
```

“%c” tells the **printf** it will print a letter having the code supplied by the **char** variable.

Following table shows the ASCII table which is used to code the letters.

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20	!	64	40	@	96	60	'
^A	1	01		SOH	33	21	"	65	41	À	97	61	a
^B	2	02		STX	34	22	#	66	42	È	98	62	b
^C	3	03		ETX	35	23	\$	67	43	Ç	99	63	c
^D	4	04		EOT	36	24	%	68	44	Ð	100	64	d
^E	5	05		ENQ	37	25	&	69	45	È	101	65	e
^F	6	06		ACK	38	26	*	70	46	Ð	102	66	f
^G	7	07		BEL	39	27	,	71	47	G	103	67	g
^H	8	08		BS	40	28	(	72	48	H	104	68	h
^I	9	09		HT	41	29	)	73	49	I	105	69	i
^J	10	0A		LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	'	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	0	80	50	P	112	70	p
^Q	17	11		DC1	49	31	1	81	51	Q	113	71	q
^R	18	12		DC2	50	32	2	82	52	R	114	72	r
^S	19	13		DC3	51	33	3	83	53	S	115	73	s
^T	20	14		DC4	52	34	4	84	54	T	116	74	t
^U	21	15		NAK	53	35	5	85	55	U	117	75	u
^V	22	16		SYN	54	36	6	86	56	V	118	76	v
^W	23	17		ETB	55	37	7	87	57	W	119	77	w
^X	24	18		CAN	56	38	8	88	58	X	120	78	x
^Y	25	19		EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B		ESC	59	3B	;	91	5B	[	123	7B	{
^`	28	1C		FS	60	3C	<	92	5C	\	124	7C	-
^]	29	1D		GS	61	3D	=	93	5D	]	125	7D	}
^~	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^-	31	1F	▼	US	63	3F	?	95	5F	—	127	7F	Ø

\* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS).  
The DEL code can be generated by the CTRL + BKSP key.

## 1.8. Formatting Numbers while Printing and Scanning

Both **printf** and **scanf** uses formatting directive to control the shape of the output or the input text, or to insert numeric values within text. Following table shows a list of all directives:

'\r\n'	Makes a newline
'\t'	Inserts a tab
'\\'	Prints '\'
'\"'	Prints "
'%d'	Prints or scans an integer (int) value
'%x' or '%X'	Prints or scans an integer value in small or capital hexadecimal format
'%f'	Prints or scans a real (float) value
'%lf'	Prints or scans a real (double) value
'%u'	Prints or scans an (unsigned int) value
'%c'	Prints or scans a single character (char)

Directives Example:

Code	Output	Description
<code>printf("A\r\nB\r\nC");</code>	A B C	"\r\n" makes a line break where '\r' is the carriage return and '\n' is the newline command.
<code>printf("A\tB\tC\r\n"); printf("D\tE\tF\r\n"); printf("N\tO\tP\r\n");</code>	A     B     C D     E     F N     O     P	'\t' makes a tab separator.
<code>printf("A\\B\\C ");</code>	A\B\C	To print the '\' letter you must place '\\' instead.
<code>printf("Say \"Hello\"");</code>	Say "Hello"	To print the '\"' letter you must place '\"' instead.
<code>int a = 20*30; printf("Area is %d",a);</code>	Area is 600	The directive '%d' is replaced with an integer value (a).
<code>printf("If the width is %d and the height is %d then the area is %d",20, 30, 20*30);</code>	If the width is 20 and the height is 30 then the area is 600	There are three '%d' directives and three integer values each value is printed instead of one of the '%d' directives, Number of '%d' directives must equals to the number of numeric values.
<code>int W, H; printf("Enter the width and the height:");</code>	Enter the width and the height:20/80 Area is 1600	The directive '%d' is used to scan an

<code>scanf("%d/%d", &amp;W, &amp;H); printf("\r\nArea is %d ", W*H);</code>		integer value. The combination ' <code>%d/%d</code> ' is used to scan two integer value separated by ' <code>/</code> '.
<code>int x = 172; printf("X equals %x ", x);</code>	<code>X equals ac</code>	The directive ' <code>%x</code> ' prints the integer value in small hexadecimal format.
<code>int X = 172; printf("X equals %X ", X);</code>	<code>X equals AC</code>	The directive ' <code>%X</code> ' prints the integer value in capital hexadecimal format.
<code>int X; printf("Enter X in hexadecimal format:"); scanf("%X", &amp;X); printf("\r\nX equals %d ", X);</code>	<code>Enter X in hexadecimal format: AC X equals 172</code>	The directive ' <code>%X</code> ' also used to scan values in hexadecimal format.
<code>float R = 2.5; printf("R equals %f ", R);</code>	<code>R equals 2.5</code>	The directive ' <code>%f</code> ' prints a real (float) value.
<code>int X = 6235; printf("X equals %10d", X);</code>	<code>X equals 6235</code> ----- -----	Prints the number in 10 digits including the ' <code>.</code> ' and 2 digits in the fraction part.
<code>float R = 8372.5675365; printf("R equals %10.2f ", R);</code>	<code>R equals 8372.56</code> ----- -----	Prints the number in 10 digits including the ' <code>.</code> ' and 2 digits in the fraction part.
<code>int X = 15; printf("X equals %05d", X);</code>	<code>X equals 00015</code> ----- -----	Prints the number in 5 digits and pad it with zeros.

## 1.9. Data Conversion and Type Casting

Computer programs usually use several data types. Sometimes it is required to copy data between different data types.

```
int radius = 5;
double area;
area = 3.14159 * radius * radius;
```

In this example the radius of type (`int`) is assigned in expression to area of type (`double`). The conversion from smaller to larger data types does not cause any problem.

```

float x = 1.2;
int y = 5;
int z = 0x878925;
int a;
char b;
a = x;           //Problem, Fraction part is lost
b = y;           //No problem, 5 < char range(-128 to 127)
b = z; //Problem, 0x878925 >> char range(-128 to 127)

```

The conversion from larger to smaller data types may lead to some data losses. For that reason the compiler warns you against this type of operation. However sometimes the conversion does not affect the data like the second expression ( $b = y$ ). Compiler is not able to differentiate between safe or unsafe situation, for that reason you must use the **type casting** to force the conversion if you decide that it is safe.

```

float x = 1.2;
int y = 5;
int z = 0x878925;
int a;
char b;
a = (int)x;    //x = 1
b = (char)y;   //y = 5
b = (char)z;   //z = 0x25

```

## 1.10. Mathematical and Logical Expressions

C language supports following expression operators:

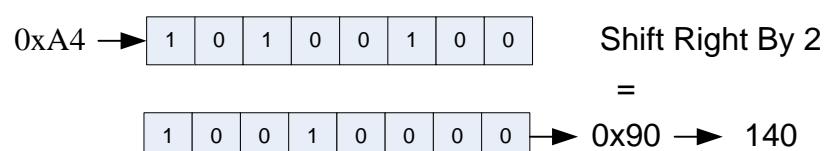
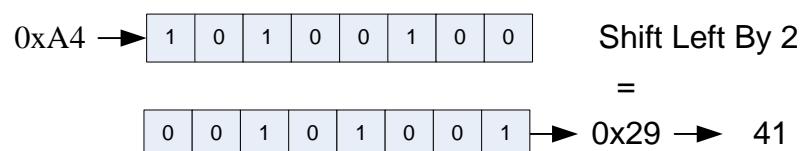
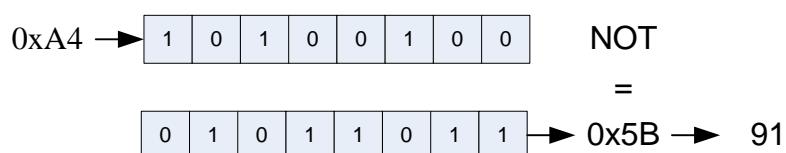
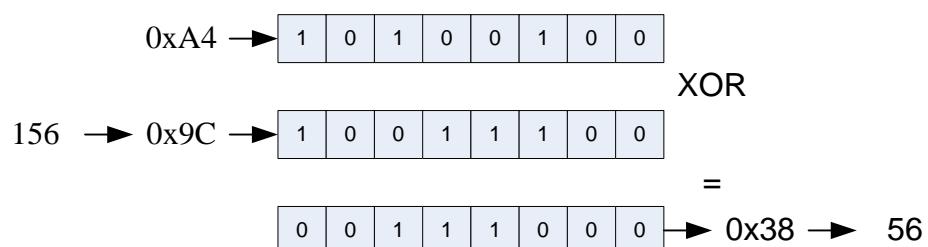
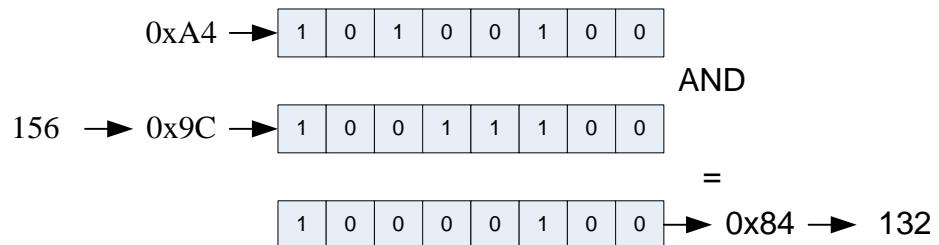
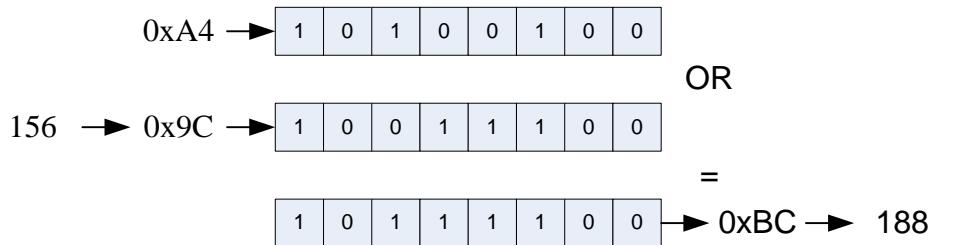
=	Equal operator. X = Y; means copy the value of Y into X
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Division Reminder (Mod) (EX: 15%4 → 3)
( )	Mathematical Parenthesis
++	Increment by one operator
--	Decrement by one operator
	Logical OR operator
&	Logical AND operator
^	Logical XOR operator
~	Logical NOT operator
>>	Shift Right Operator
<<	Shift Left Operator
+=	Self addition operator. Ex: X+=2 means X = X + 2
Other self operators -=, *=, /=, %=,  =, &=, ^=, >>=, <<=	

Following examples show how to use those operators to calculate different mathematical e:

Mathematical Function	C Representation
$X = \left( (A \times B) + \left( \frac{A}{B} \right) \right) \times (A - B)$	<b>X = ((A*B)+(A/B))*(A-B);</b>
$X = \frac{A \times (B + C)}{\frac{A}{C} - B}$	<b>X = A*(B+C) / ((A/C)-B);</b>
$X = A^5$	<b>X = A*A*A*A*A;</b>

Following examples provides some specific C expressions:

C Expression	Meaning
<b>X = X + 9;</b>	Calculate X+9 then stores the result in X
<b>X++;</b>	Add one to X
<b>X--;</b>	Subtract one from X
<b>X = 10;</b> <b>Y = 5;</b> <b>X = X + Y++;</b>	Add X+Y → 15 then increment Y → 6 Store 15 in X
<b>X = 10;</b> <b>Y = 5;</b> <b>X = X + ++Y;</b>	Increment Y → 6 Add X+Y → 16 Store 16 in X
<b>unsigned char X = 0xA4;</b> <b>unsigned char Y = 156;</b> <b>unsigned char Z = X Y;</b>	Calculate the X OR Y → 0xBC (188)
<b>unsigned char X = 0xA4;</b> <b>unsigned char Y = 156;</b> <b>unsigned char Z = X&amp;Y;</b>	Calculate the X AND Y → 0x84 (132)
<b>unsigned char X = 0xA4;</b> <b>unsigned char Y = 156;</b> <b>unsigned char Z = X^Y;</b>	Calculate the X XOR Y → 0x38 (56)
<b>unsigned char X = 0xA4;</b> <b>unsigned char Z = ~X;</b>	Calculate the (NOT X) → 0x5B (91)
<b>unsigned char X = 0xA4;</b> <b>unsigned char Z = X&gt;&gt;2;</b>	Calculate the X shifted by two bits to right → 0x29 (41)
<b>unsigned char X = 0xA4;</b> <b>unsigned char Z = X&lt;&lt;2;</b>	Calculate the X shifted by two bits to right → 0x90 (140)



Parenthesis is important to determine the order of calculation and remove any confusion.

C Expression	Meaning
<code>X = X + Y * Z;</code>	Multiply Y by Z then Add X
<code>X = (X + Y) * Z;</code>	Add X to Y then multiply by Z

## 1.11. Advanced Mathematical Expressions

Mathematical operators can be used to perform simple calculations; however it cannot be used to perform advanced calculations like square root or logarithm or trigonometric calculations. Advanced mathematic operations are provided by the math library “math.h”. You must include this library in the include header. Math library performs its calculations using real numbers, which means that you must use float or double variables. Following examples demonstrate how to use the math library:

### Example 1: Using Trigonometric Functions

```
#include "stdio.h"
#include "math.h"

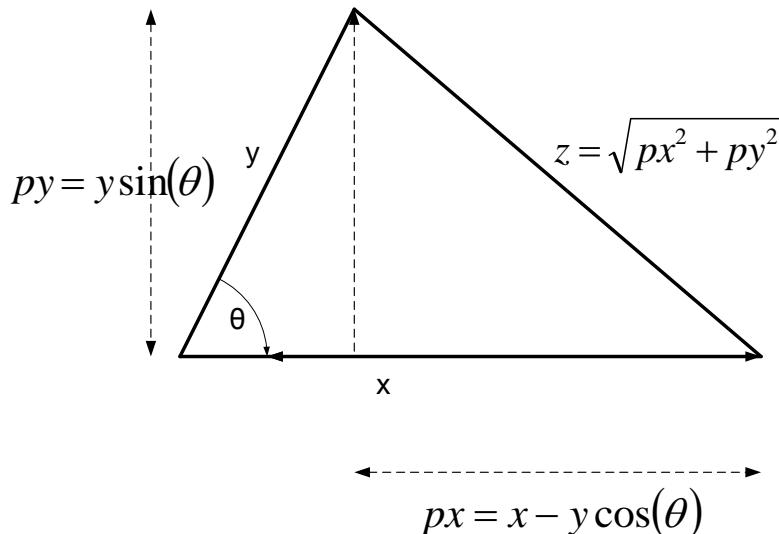
void main()
{
    double lineLength = 200.0;
    double slope = 0.35;
    //0.35 radian --> 0.1 * 180/PI --> 20 degree

    printf("The horizontal projection length is %lf\r\n",
           lineLength * cos(slope));

    printf("The vertical projection length is %lf\r\n",
           lineLength * sin(slope));
}
```

### Example 2: Calculate Triangle Side Length

This example calculates the third triangle side length given the other side's lengths and the angle between them in degrees. The geometric idea is illustrated in the next figure.



```
#include "stdio.h"
#include "math.h"

#define PI 3.141592
void main()
{
    float x, y, theta, px, py, z;

    printf("Enter two side length and the angle between them
           in degrees:\r\n");
    scanf("%f %f %f", &x, &y, &theta);

    //Convert theta from Degrees to Radian
    theta = theta * PI / 180;

    px = x - y * cos(theta);
    py = y * sin(theta);
    z = sqrt( pow(px, 2) + pow(py, 2));

    printf("The third side length is %f\r\n", z);
}
```

Math library provides following mathematical operations:

Mathematical Operation Let x = 2.1 and z = 1.2	C Example
$y = \sin(x)$	<code>y = sin(x);</code>
$y = \cos(x)$	<code>y = cos(x);</code>
$y = \tan(x)$	<code>y = tan(x);</code>

$y = \sin^{-1}(x)$	<code>y = asin(x);</code>
$y = \cos^{-1}(x)$	<code>y = acos(x);</code>
$y = \tan^{-1}(x)$	<code>y = atan(x);</code>
$y = \tan^{-1}(x/z)$	<code>y = atan2(x, z);</code> <i>//to avoid dividing by 0 if z = 0</i>
$y = \sqrt{x}$	<code>y = sqrt(x);</code>
$y = (x)^{3.22}$	<code>y = pow(x, 3.22);</code>
$y = \ln(x)$	<code>y = log(x);</code>
$y = \log_{10}(x)$	<code>y = log10(x);</code>
$y =  x  = 2.1$	<code>y = abs(x); //if x is integer</code> <code>y = fabs(x); //if x is real</code>
$y = [x] = 3$	<code>y = ceil(x);</code>
$y = [x] = 2$	<code>y = floor(x);</code>

## 1.12. Coding Convention

Coding convention is a set of rules that enhance the readability and the understandability of the code. At the end of each chapter a list of standard and related coding convention is mentioned.

### 1.12.1. Indentation and Code Arrangement

Indentation means arranging the code inside the brackets. Following example shows a non-arranged code.

```
#include      "stdio.h"

void main()
{
    int x      =6,      y=7;
    int z;
z = x+y      ;
printf(      "z = %d", z);
}
```

It appears that the above code is unreadable.

**First Mistake:** New programmers usually use spaces to push some lines forward.

```
int x      =6,      y=7;
int z;
```

**Second Mistake:** More than one space is used to separate code sections.

```
x      =6
```

**Third Mistake:** The code in different lines is not aligned.

Following example shows an arranged code.

```
#include "stdio.h"

void main()
{
    int x = 6, y = 7;
    int z;
    z = x + y;
    printf("z = %d", z);
}
```

It appears that above code is more readable. Generally following roles must be obeyed to write an arranged code.

**First:** Use One Tab to push the whole code between any two packets

```
{
    int x = 6, y = 7;
    int z;
    z = x + y;
    printf("z = %d", z);
}
```

**Second:** Use One Space to separate small code segments. For example:

```
int x = 6, y = 7;
```

**Third:** Use One Line Only to separate code blocks. For example:

```
#include "stdio.h"

void main()
{
    ...
}
```

**Fourth:** Do not leave lines between the head of the block and the body of it. For example, do not leave a line between (**void main()**) and its body:

```
void main()
{
```

### 1.12.2. Variable Naming

Standard rules for the naming of the variables:

**First:** local variable must start with small letter unless the name is known shortcut, for example:

```
int x, value, studentDegree, ID;
```

**Second:** if variable name consists of separate sections, let each section begin with Capital letter, for Example:

```
int tempratureValue, studentMathDegree;
```

**Third:** Short cuts must use Capital Letters, for example:

```
int ID, STD;
```

### 1.13. Exercises

Q1.

Following program has many syntax errors. Enumerate the errors then re-write the program after correction.

```
#include stdio.h

Void Main()
{
    float w
    printf "Enter your weight in pounds ";
    scanf[%d" weight];
    printf('Your weight in kilograms is %d',
           weight * 0.45359237);
}
```

Q2.

Write a program that takes the following polynomial coefficients then evaluates and prints the polynomial value at x equals 10.

$$y(x) = ax^2 + bx + c$$

Q3.

Write a program that takes the distance in feet then converts and prints it in meters.

Q4.

Write a program that takes the ASCII code then prints the equivalent character. What is the equivalent character to 37, 84, 116, 0x6E, 0x55, 0x28, 32

Q5.

Write a program that takes a character from user then prints the equivalent ASCII code. What is the equivalent ASCII code to 'a', 'A', '2', '%', 'c' and the space.

Q6.

Write a program that evaluates the following expression at given x, y and z values.

$$f(x, y, z) = \frac{x + \sqrt{e^{x^{0.2}} + (x^2 + y^3)}}{\sin\left(\frac{x}{(z * 10.5)}\right) + z^{5.5}}$$

Q7.

Write the expected output of the following program.

```
#include "stdio.h"

void main()
{
    int x = 11;
    int y = 23;
```

```

int z = -42;

printf("x = %d (%08X)\r\n", x, x);
printf("y = %d (%08X)\r\n", y, y);
printf("z = %d (%08X)\r\n", z, z);
printf("x or y = %d (%08X)\r\n", x|y, x|y);
printf("x and y = %d (%08X)\r\n", x&y, x&y);
printf("x xor y = %d (%08X)\r\n", x^y, x^y);
printf("not z = %d (%08X)\r\n", ~z, ~z);
}

```

Q8.

Write the expected output of the following program. (Justify your answer?)

```

#include "stdio.h"

void main()
{
    int x = 15;
    int y = 2;
    int z;
    float r;

    z = x/y;
    printf("x/y = %d\r\n", z);

    r = x/y;
    printf("x/y = %f\r\n", r);

    r = x/float(y);
    printf("x/y = %f\r\n", r);

    r = x/(y * 1.0);
    printf("x/y = %f\r\n", r);
}

```

Q9.

Write the expected output of the following program.

```

#include "stdio.h"
#include "math.h"

void main()
{
    float x;

    x = 1.7;
}

```

```
printf("float(x) = %f\r\n", x);

x = int(1.3);
printf("int(x) = %f\r\n", x);

x = ceil(1.3);
printf("ceil(x) = %f\r\n", x);

x = floor(1.3);
printf("floor(x) = %f\r\n", x);
}
```

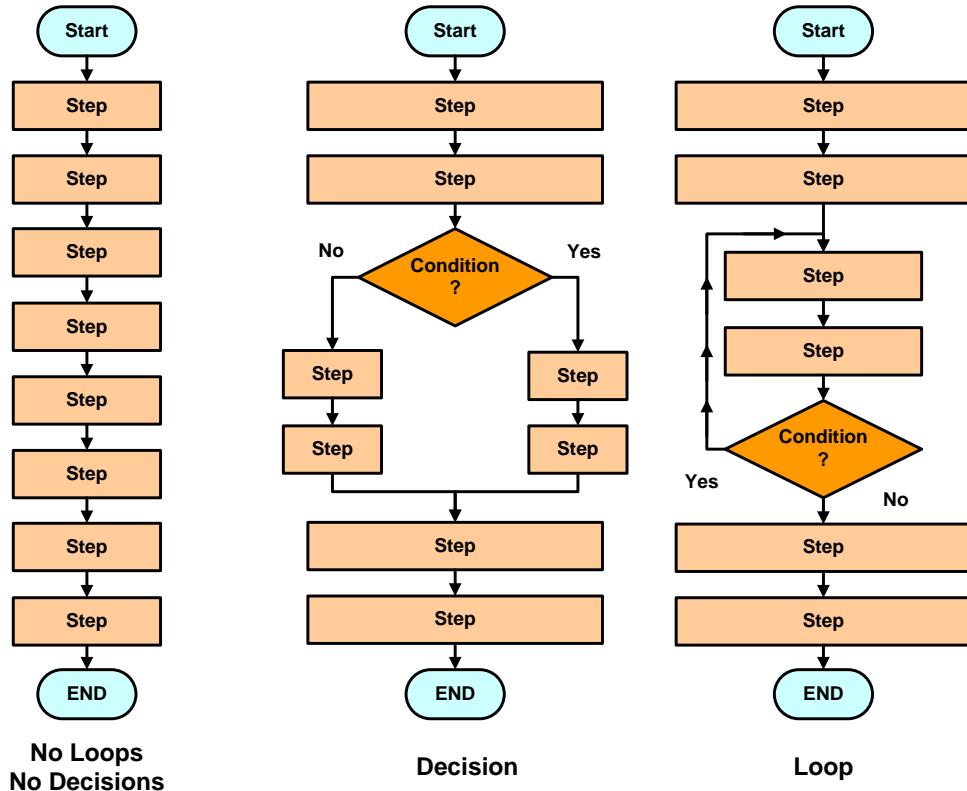
## Chapter 2: Controlling Program Flow

## 2.1. Program Execution Flow

Computer executes programs step by step starting from the **program entry point** “main” up to the end. However sometimes it is required to change the sequence of the execution.

Sometimes for certain condition some steps must be skipped. This called a **Decision**.

Sometimes it is required to repeat some steps for a number of times or until certain condition. This called a **Loop**.



**Condition** is an expression that produces a numeric result to indicate either a success or failure. Successful condition denoted by “Non Zero Value” or “Yes” or “True”. Failed conditions denoted by “0” or “No” or “False”.

C language provides several statements to change the execution flow, those statements are:

- **if** statement
- **switch** statement
- **for** statement
- **while** statement
- **do...while** statement
- **goto** statement

## 2.2. Conditions

Condition statement is an expression that produces (zero – treated as FALSE) or (non zero - treated as TRUE). Usually conditional operators are used to write the condition statement. Conditional operator are different from mathematical operators, it produces directly (0 value to indicate FALSE condition) and (1 value to indicate TRUE condition). Following table shows the conditional operators:

Operator	Meaning
>	Greater
>=	Greater or equal
<	Less
<=	Less than or equal
==	Equal
!=	Not equal
!	Not If the input is true the output is false if the input is false the output is true
&&	And Example: A>B && C>D If both sides are true the output is true, otherwise it gives false
	Or Example: A>B    C>D If either sides is true the output is true, otherwise it gives false

Following examples illustrate the usage of the condition.

### Example 1: Using Conditions

```
#include "stdio.h"
#include "math.h"

void main()
{
    int a = 9;
    int b = 8;
    int c = 12;
    printf("%d\r\n", a>b); //prints 1
    printf("%d\r\n", b>c); //prints 0
    printf("%d\r\n", a<=9); //prints 1
    printf("%d\r\n", a!=9); //prints 0
    printf("%d\r\n", (a-b)>(c-b)); //prints 0
    printf("%d\r\n", a>b && c>b); //prints 1
    printf("%d\r\n", a>b && c<b); //prints 0
    printf("%d\r\n", a>b || c<b); //prints 1
    printf("%d\r\n", !(a<b)); //prints 1
    printf("%d\r\n", 3 && 0); //prints 0
    printf("%d\r\n", -15 || 0); //prints 1
    printf("%d\r\n", !(-15)); //prints 0
}
```

```
}
```

### 2.3. if Statement

Syntax:

```

if(/*if condition*/)
{
    //if body
}
else if(/*else if condition*/)
{
    //else if body
}
else if(/*else if condition*/)
{
    //else if body
}
else
{
    //else body
}

```

If statement works as follows; the computer evaluate the (**if condition**), if the result is true it executes the (**if body**), if the result is false it proceeds to the (**else if**) statement if present. Again, the computer evaluate the (**else if condition**), if the result is true it executes the (**else if body**), if the result is false it proceeds to the next (**else if**) statement if present. Finally if all (**if..else if..else if**) blocks give false result the computer proceeds to the (**else**) statement if present and execute the (**else body**). The usage of (**else if**) and (**else**) statements is optional, you do not have to use them if it is not required. Following examples illustrate how to use the (**if**) statement.

#### Example 1: Calculate Circle Area or Circumference

In this program the user has to choose between calculating circle area or circle circumference. The choice comes by taking a character from the keyboard using the (`getche`) function. If the user presses ‘a’ character it proceeds with area calculation and printing. If the user presses ‘c’ character it proceeds with circumference calculation and printing. If the user presses other letters the program prints an error message.

```

#include "stdio.h"
#include "conio.h" //for getch

#define PI 3.14159

void main()

```

```
{
    char choice;
    float radius;
    float area, circumference;

    printf("Enter circle radius : ");
    scanf("%f", &radius);

    printf("Enter your choice (a to print the area,
           c to print the circumference) : ");
    choice = getche();
    if(choice=='a')
    {
        area = PI * radius * radius;
        printf("\r\narea is %f\r\n", area);
    }
    else if(choice=='c')
    {
        circumference = 2 * PI * radius;
        printf("\r\ncircumference is %f\r\n",
               circumference);
    }
    else
        printf("\r\nwrong choice\r\n");
}
```

**Important:** There is no need to use brackets if the body block consists of one statement as in the **(else)** case. However if the body contains more than one statement it is a must to use the brackets.

Question? What if the user enters capital letters like ‘A’ or ‘C’.

### Example 2: Comparing Three Numbers

This program finds the largest value of the three given values.

```
#include "stdio.h"

void main()
{
    int a, b, c;
    printf("Enter three values : ");
    scanf("%d %d %d", &a, &b, &c);

    if(a>b)
    {
        if(a>c)
            printf("the largest value is %d\r\n", a);
```

```

        else
            printf("the largest value is %d\r\n", c);
    }
    else
    {
        if(b>c)
            printf("the largest value is %d\r\n", b);
        else
            printf("the largest value is %d\r\n", c);
    }
}

```

**Important:** In this example there are three (**if**) statements, two of them inside the first one. It is applicable to place (**if**) statement inside other statements as if you use the brackets correctly to avoid confusing the compiler. Putting any statement inside any other statements called **statements nesting**.

## 2.4. *Inline condition*

Sometimes it is required to take a fast decision inside your statements; this is called the inline condition. Following examples illustrate the idea.

```

#include "stdio.h"

void main()
{
    int a, b, minimum;
    printf("Enter tow numbers : ");
    scanf("%d %d", &a, &b);
    minimum = (a<b) ? a : b;
    printf("The minimum is %d\r\n", minimum);
}

```

minimum = ( a<b ) ? a : b;	Equivalent To	$\text{if}(a < b)$ minimum = a; $\text{else}$ minimum = b;
----------------------------	---------------	---

```
minimum = ( a<b ) ? a : b ;
```

Condition                            Assigned value in case of true                    Assigned value in case of false

It is clear that inline condition is used to reduce the code size, furthermore above code can have more reduction as shown in the following program.

### Example 1: Calculate the Minimum of Two Numbers

```
#include "stdio.h"

void main()
{
    int a, b;
    printf("Enter tow numbers : ");
    scanf("%d %d", &a, &b);
    printf("The minimum is %d\r\n", (a<b)?a:b);
}
```

## 2.5. switch Statement

Syntax:

```
switch(/*switch expression*/)
{
    case /*case value*/:
        {
            //case body
        }
        break;
    ...
    ...
    ...
    case /* case value*/:
        {
            //case body
        }
        break;
    default:
        {
```

```
//case body
}
break;
}
```

**switch** statement works as follows; the computer evaluates the (**switch expression**), then it jumps to the appropriate **case** statement with (**case value**) equals to the calculated integer result, if none of the **case** statements are applicable it jumps to the **default** statement, if default statement is not present it jumps out of the whole **switch** statement.

When the computer jumps to the case or default statement it executes the (**case body**). After executing the (**case body**) if it is followed by the **break** statement, the computer jumps outside the whole switch statement, otherwise it moves directly to the next (**case body**) until it finds the break statement or finishes the whole switch statement.

It is not allowed to have two cases with the same (**case value**).

### Example 1: Calculate Circle Area or Circumference

```
#include "stdio.h"
#include "conio.h"

#define PI 3.14159

void main()
{
    float radius, area, circumference;

    printf("Enter circle radius : ");
    scanf("%f", &radius);

    printf("Enter your choice (a to print the area,
          c to print the circumference) : ");
    switch(getche())
    {
        case 'a':
        case 'A':
        {
            area = PI * radius * radius;
            printf("\r\narea is %f\r\n", area);
        }
        break;
        case 'c':
        case 'C':
        {
            circumference = 2 * PI * radius;
            printf("\r\ncircumference is %f\r\n",
                  circumference);
        }
    }
}
```

```

        }
        break;
    default:
    {
        printf("\r\nwrong choice\r\n");
    }
    break;
}
}

```

Important: this program will behave correctly if the user supplies small or capital letters (EX: 'a' or 'A').

In the above program we can eliminate the use of (**area** and **circumference**) variables.

```

#include "stdio.h"
#include "conio.h"

#define PI 3.14159

void main()
{
    float radius;

    printf("Enter circle radius : ");
    scanf("%f", &radius);
    printf("Enter your choice (a to print the area,
           c to print the circumference) : ");
    switch(getche())
    {
    case 'a':
    case 'A':
        printf("\r\narea is %f\r\n",
               PI * radius * radius);
        break;
    case 'c':
    case 'C':
        printf("\r\ncircumference is %f\r\n",
               2 * PI * radius);
        break;
    default:
        printf("\r\nwrong choice\r\n");
        break;
    }
}

```

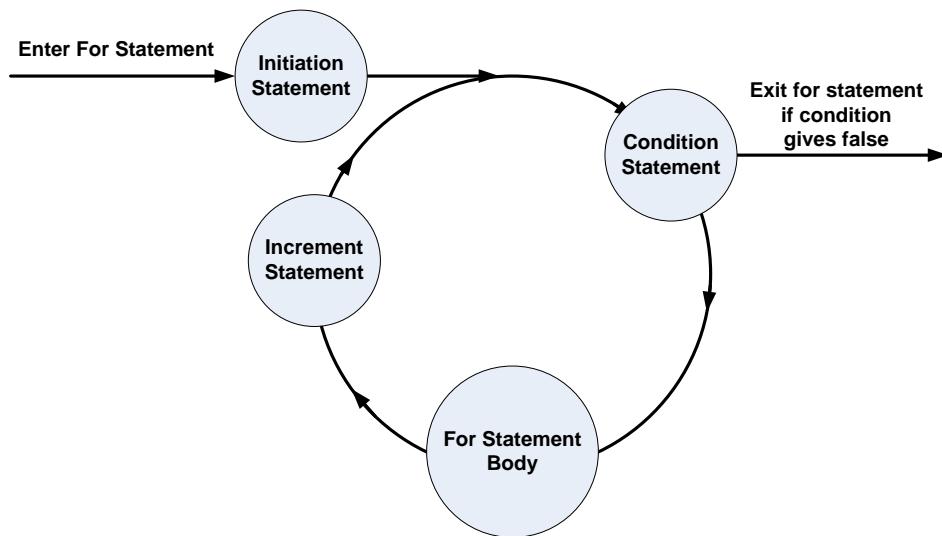
## 2.6. for Statement

Syntax:

```
for(/*intiation*/; /*condition*/; /*increment*/)
{
    //for body
}
```

**for** statement repeats the execution of the (**for body**) until the (**condition**) statement is not succeeded any more. Computer processes **for** statement as follows:

1. Execute the (**intiation**) statement to assign an initiate value to some variable if it is required.
2. Execute the (**condition**) statement, if false, go out of the for statement, otherwise, proceed to the next step
3. Execute the (**for body**)
4. Execute the (**increment**) statement to update some variables
5. Go back to (Step 2)



### Example 1: Printing Hello World in a Loop

```
#include "stdio.h"

void main()
{
    int i;
    for(i=0;i<10;i++)
    {
        printf("%d : Hello World\r\n", i);
    }
}
```

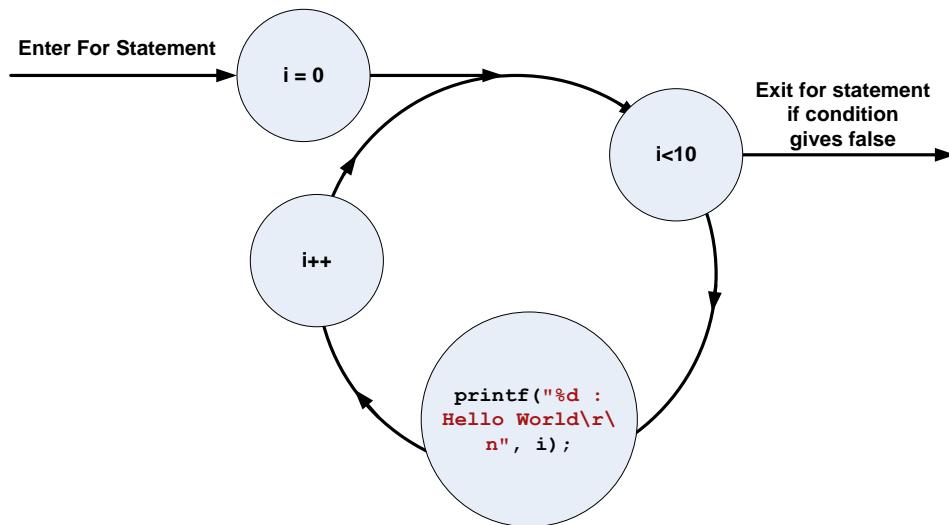
```
}
```

The screenshot shows a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The window displays the following text:  
0 : Hello World  
1 : Hello World  
2 : Hello World  
3 : Hello World  
4 : Hello World  
5 : Hello World  
6 : Hello World  
7 : Hello World  
8 : Hello World  
9 : Hello World  
Press any key to continue . . .

Above example prints “Hello World” 10 times. The program works as following:

1. Execute the initiation statement ( $i=0$ )
2. Execute the condition statement ( $i < 10$ ), if false, exit from the for statement
3. Execute the body of the for statement
4. Execute the increment statement ( $i++$ )
5. Go back to (Step 2)

Initially ( $i$ ) is loaded with (0), after each loop it is incremented by (1). If ( $i$ ) value reaches (10) the condition statement fails and the computer exits from the loop. It is clear that ( $i$ ) variable takes the values {0,1,2,3,4,5,6,7,8,9} during the execution.



### Example 2: Calculate the Summation of values between 1 and 99

Following program calculates the summation of numbers between 1 and 99.

```
#include "stdio.h"

void main()
```

```
{
    int i, sum = 0;

    for(i=1;i<=99;i++)
    {
        sum += i;
    }

    printf("Summation of values between (1 and 99) is : %d",
    sum);
}
```

**Example 3: Calculate the Average Students Degrees**

Following program calculates the average students degree for 10 students.

```
#include "stdio.h"

void main()
{
    int i;
    float degree, sum;

    for(i=1, sum=0; i<=10; i++)
    {
        printf("Enter student (%d) degree:", i);
        scanf("%f", &degree);
        sum += degree;
    }

    printf("Average students degree is : %f\r\n", sum/10);
}
```

What if the number of the students is unknown for the programmer? Following program calculates the average students degree for any given students number.

```
#include "stdio.h"

void main()
{
    int i, nStudents;
    float degree, sum;

    printf("Enter the number of the students:");
    scanf("%d", &nStudents);
```

```

for(i=1, sum=0; i<=nStudents; i++)
{
    printf("Enter student (%d) degree:", i);
    scanf("%f", &degree);
    sum += degree;
}

printf("Average students degree is : %f\r\n",
       sum/nStudents);
}

```

## 2.7. while Statement

Syntax:

```

while(/*condition*/)
{
    //while body
}

```

**while** statement is similar to the **for** statement, however it is more simple, there is no initiation or increment statements, you have to choose where to initiate and where to increment your variables if you need this. The computer executes the while statement as follows:

1. Execute the (**condition**) statement, if false, go out of the **while** statement, otherwise, proceed to the next step
2. Execute the (**while body**)
3. Repeat (Step 1)

### Example 1: Calculate the Summation of odd values between 1 and 99

```

#include "stdio.h"

void main()
{
    int i = 1, sum = 0; // Initiation is placed here

    while(i<=99)
    {
        sum += i;
        i+=2; // Increment is placed here
    }
    printf("Summation of odd values between
           (1 and 99) is : %d", sum);
}

```

### Example 2: Calculate the Average Students Degrees

Following program calculates the average students degree. The difference that it does not ask for the number of the students first, it takes the degrees directly and exits if a negative value is supplied.

```
#include "stdio.h"

void main()
{
    int nStudents = 0;
    float degree, sum = 0;

    printf("Enter negative value to exit:\r\n");
    while(1)
    {
        printf("Enter student (%d) degree:", nStudents + 1);
        scanf("%f", &degree);

        if(degree<0)break; //force exit from while loop

        sum += degree;

        nStudents++;
    }

    printf("Average students degree is : %f\r\n",
           sum/nStudents);
}
```

**Important:** **break** statement is used to exit from any loop type. In this example if the degree value is negative the break statement is executed and the program exits from the while loop and execute the next statement.

### 2.8. do...while Statement

Syntax:

```
do
{
    //do...while body
}
while(/*condition*/);
```

**do ... while** statement is similar to while statement, except that the condition is checked after executing each loop, which means that, the first loop is performed without a check. The computer executes the while statement as follows:

1. Execute the (**do...while body**)
2. Execute the (**condition**) statement, if false, go out of the **do...while** statement, otherwise go to (Step 1)

### Example 1: Calculate Polynomial Value

Following program evaluates the polynomial  $f(x) = 5X^2 + 3X + 2$

```
#include "stdio.h"
#include "conio.h"

void main()
{
    float x, y;

    do
    {
        printf("\r\nEnter x value:");
        scanf("%f", &x);
        y = 5*x*x + 3*x + 2;
        printf("\r\ny(%f) = %f", x, y);

        printf("\r\nDo you want to evaluate
                           again (y/n):");
    }
    while(getche()=='y');
}
```

## 2.9. goto Statement

Syntax:

<pre>// C Statement labelname: // C Statement // C Statement goto labelname; // C Statement</pre>	<pre>// C Statement goto labelname; // C Statement // C Statement labelname: // C Statement</pre>
---	---

Simply **goto** statement tells the program where to jumps, it can jump forward or backward. Following example illustrates the idea.

```
#include "stdio.h"
#include "conio.h"

void main()
{
    float x, y;

evaluate_again:

    printf("\r\nEnter x value:");
    scanf("%f", &x);
    y = 5*x*x + 3*x + 2;
    printf("\r\ny(%f) = %f", x, y);

    printf("\r\nDo you want to evaluate again (y/n):");

    if(getche()=='y')
        goto evaluate_again;
}
```

**Important:** It is not recommended to use **goto** statement extensively, because it allows programmers to jump anywhere in their program and this lead to unorganized and unreadable codes.

## 2.10. Coding Convention

### 2.10.1. Indentation and Code Arrangement

First: Brackets of control statements must start at the next line, also the inside code must be pushed by one Tab.

```
if( x==y )
{
    print("..."); 
    print("...");
}
else
{
    print("..."); 
    print("...");
}
```

Second: If the control statement has a body of one statement, you can ignore the brackets. You can either place the statement at the same line or in the next line pushed by one Tab.

<pre>if(x==y)     print("...");</pre>	<pre>if(x==y)print("...");</pre>
---------------------------------------	----------------------------------

## 2.11. Exercises

Q1.

Write the expected output of the following program:

```
#include "stdio.h"

void main()
{
    int a = 12;
    int b = 3;
    int c = 52;
    printf("%d\r\n", a>b && b<c);
    printf("%d\r\n", !(b>c && (a+200)>c));
    printf("%d\r\n", (a-3)==9 && 1 );
    printf("%d\r\n", 0 || 111);
    printf("%d\r\n", 1 && -1);
    printf("%d\r\n", !(-9) && !0);
}
```

Q2.

Write the expected output of the following program:

```
#include "stdio.h"
void main()
{
    int a = 12;
    int b = 3;
    int c = 52;

    if(a>b)
    {
        if(b>c)
        {
            printf("R1");
        }
        else if(a<c)
        {
            if(b<c && 0)
                printf("R2");
            else if(a<c && !0)
                printf("R3");
        }
        else
        {
            if(a>b)
                printf("R4");
            else
                printf("R5");
        }
    }
}
```

```

    }
else
{
    printf("R6");
}
}

```

Q3.

Using inline condition to write a program that takes 3 numbers a, b and c then finds and prints the maximum value.

Q4.

Using if condition to write a program that takes 4 numbers a, b, c and d, then find and prints the maximum value.

Q5.

Write a program that takes two numbers x, y and a choice letter c to select between different mathematical operations. c may take (+, -, \* and /) values, otherwise the program should prints an error and repeat the request. If the user supplies the right choice the program should performs the operation and prints the result.

Q6.

Write a program that produces the following output:

```
C:\Windows\system32\cmd.exe
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9
3 4 5 6 7 8 9
4 5 6 7 8 9
5 6 7 8 9
6 7 8 9
7 8 9
8 9
9
```

Q7.

Write a program that prints the multiplication table as shown in the following figure:

*	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	2	4	6	8	10	12	14
3	3	6	9	12	15	18	21
4	4	8	12	16	20	24	28
5	5	10	15	20	25	30	35
6	6	12	18	24	30	36	42
7	7	14	21	28	35	42	49

Q8.

Write a program that calculates the summation of the numbers starting from 1 to 1000.

**Q9.**

Using getche(), write a program that keep taking letters from user until the user type the word ‘byby’.

```
C:\Windows\system32\cmd.exe
Isadkjfhadskjhf jsadhf jkhsadjkhyby
Good By
```

**Q10. (Report)**

Using getch(), write a program that asks the user to enter a password if the password is correct the program prints “correct password” if not it repeat the trial again. The password should be fixed “asueng”. The supplied password letters should be printed as “\*\*\*\*\*”. The false trial should be limited to 3 trials.

```
C:\Windows\system32\cmd.exe
Enter the Password:*****
Wrong Password.

Enter the Password:*****
Wrong Password.

Enter the Password:*****
$$$$$$ Correct Password. $$$$

```

**Q11.**

Write a program that takes a date (EX: 17 6 2009) then calculates and prints the number of days starting from the beginning of the given year.

**Q12. (Report)**

Write a program that takes two dates (EX: 1 5 1997 and 17 7 2009) then calculates and prints the number of days between them.

**Q13.**

Write a program that takes any integer value then prints the equivalent binary value.

**Q14.**

Write the expected output of the following program:

```
#include "stdio.h"

void main()
{
    int i, j, s = 1;
    for(i=1;i!=0;i+=s)
    {
        for(j=i;j<2*i;j++)
        {
```

```
        printf("%d ", j);
    }
    printf("\r\n");
    if(i==5)s*=-1;
}
}
```

Q15.

Write the expected output of the following program:

```
#include "stdio.h"

void main()
{
    int i, j, s = 1;
    for(i=2;i<10;i++)
    {
        j = 1;
        while(j<i)
        {
            if((i%j)==0)
                printf("%d ", j);
            j++;
        }
        printf("\r\n");
    }
}
```

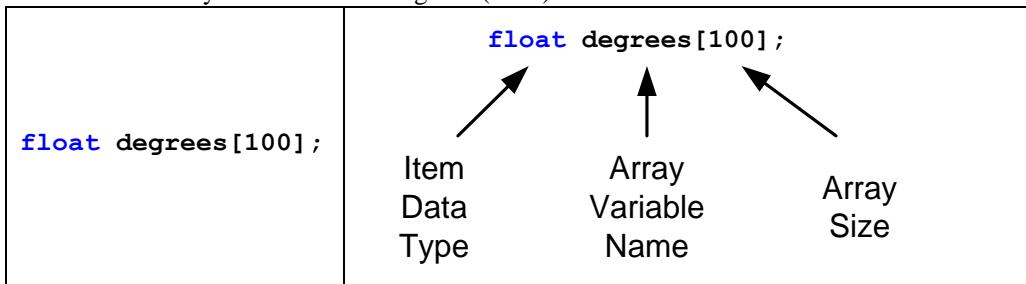
## Chapter 3: Advanced Data Types

### 3.1. Arrays

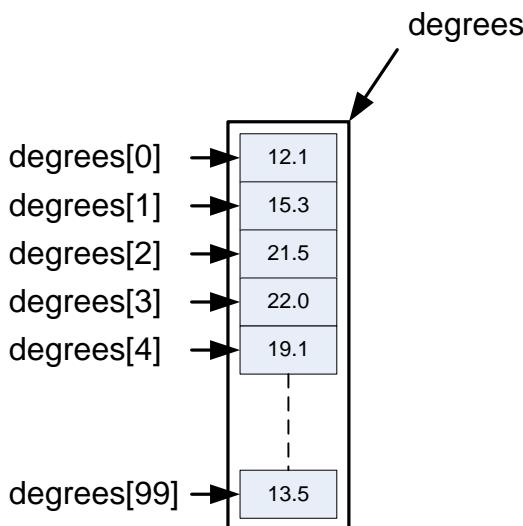
What if you need to store 100 student's degrees in order to calculate the average degree. Using variables, the solution is to define 100 variables {degree1, degree2... degree100} to store and process all student degrees. Using huge number of variables increases program size and increase the probability of human mistakes.

Array is a single variable that able to hold several values. Programmer can set or get specific value in the array.

To define an array variable containing 100 (**float**) values:



Following figure shows how the array is represented in computer memory:



The position of each value is called the index. The index of the first value is 0, which means if the array contains 10 items the index range from 0 to 9.

Dealing with array elements is very similar to normal variables; you can assign a value to any item or assign any item to other program variables. Following code illustrates this idea:

```
float degrees[100];
float x = 9.9;
```

```

float y;

//assign x value to the fifth item
degrees[4] = x;

//assign fifth item to y
y = degrees[4];

//scan eighth item from user
scanf("%f", &degrees[7]);

//print eighth item from user
printf("The eighth item value is %f", degrees[7]);

```

### Example 1: Store and Print 10 Students Degrees

Following example shows how to scan 10 students degree from user, stores them in a single area, and then prints them.

```

#include "stdio.h"

void main()
{
    int i;
    float degrees[10];

    //Scanning students degrees and storing them in array
    for(i=0; i<10; i++)
    {
        printf("\r\nEnter student %d degree : ", i+1);
        scanf("%f", &degrees[i]);
    }

    //Printing all students degrees
    for(i=0; i<10; i++)
    {
        printf("\r\nStudent %d degree is %f",
               i+1, degrees[i]);
    }
}

```

Initializing the array:

When the array is defined, there is no data assigned to their elements. Assigning initial values for array elements is called initializing the array. Following code section shows how to initialize the array:

```
float degrees[10] = {75.5, 88.0, 89.5, 23.5, 72.0, 63.5, 57.5,
62.0, 13.5, 46.5};
```

It is applicable in C to not to mention the array size, and let the compiler calculates it from the number of the initial values.

```
float degrees[] = {75.5, 88.0, 89.5, 23.5, 72.0, 63.5, 57.5,
62.0, 13.5, 46.5};
```

### Example 2: Calculate Polynomial Value for a Set of Inputs

It is required to evaluate the polynomial  $f(x) = 5X^2 + 3X + 2$  for the following values {5, 16, 22, 3.5, 15}.

```
#include "stdio.h"

void main()
{
    float x, y;

    x = 5;
    y = 5 * x * x + 3 * x + 2;
    printf("y(%f) = %f\r\n", x, y);

    x = 16;
    y = 5 * x * x + 3 * x + 2;
    printf("y(%f) = %f\r\n", x, y);

    x = 22;
    y = 5 * x * x + 3 * x + 2;
    printf("y(%f) = %f\r\n", x, y);

    x = 3.5;
    y = 5 * x * x + 3 * x + 2;
    printf("y(%f) = %f\r\n", x, y);

    x = 15;
    y = 5 * x * x + 3 * x + 2;
    printf("y(%f) = %f\r\n", x, y);
}
```

```
C:\Windows\system32\cmd.exe
y<5.000000> = 142.000000
y<16.000000> = 1330.000000
y<22.000000> = 2488.000000
y<3.500000> = 73.750000
y<15.000000> = 1172.000000
Press any key to continue . . .
```

**Comment:** the program works correctly, however it appears that code size contains many repeated sections, each section for each value. If the number of values increases code size increases.

Alternative Solution using Arrays and Loops

```
#include "stdio.h"

void main()
{
    float x[5] = {5, 16, 22, 3.5, 15};
    float y;
    int i;

    for(i=0;i<5;i++)
    {
        y = 5 * x[i] * x[i] + 3 * x[i] + 2;
        printf("y(%f) = %f\r\n", x[i], y);
    }
}
```

**Comment:** array and loops succeeds to reduce the code size.

### Example 3: Calculate the Summation of Array

Following example calculates and prints the average of an array containing 10 values.

```
#include "stdio.h"

void main()
{
    int i;
    float values[10] = {75.5, 88.0, 89.5, 23.5, 72.0,
                        63.5, 57.5, 62.0, 13.5, 46.5};
    float sum;

    //Calculate the sum
    for(i=0, sum=0; i<10; i++)
        sum += values [i];
```

```

    //Prints the average
    printf("Average value is %f\r\n", sum/100);
}

```

#### Example 4: Calculate the Maximum of the Array

Following example finds the maximum value in array of size 10.

```

#include "stdio.h"

void main()
{
    int i;
    float degrees[] = {75.5, 88.0, 89.5, 23.5, 72.0,
                        63.5, 57.5, 62.0, 13.5, 46.5};
    float max;

    max = degrees[0];
    for(i=1; i<10; i++)
    {
        if(max<degrees[i])
        {
            max = degrees[i];
        }
    }

    printf("Maximum degree is %f\r\n", max);
}

```

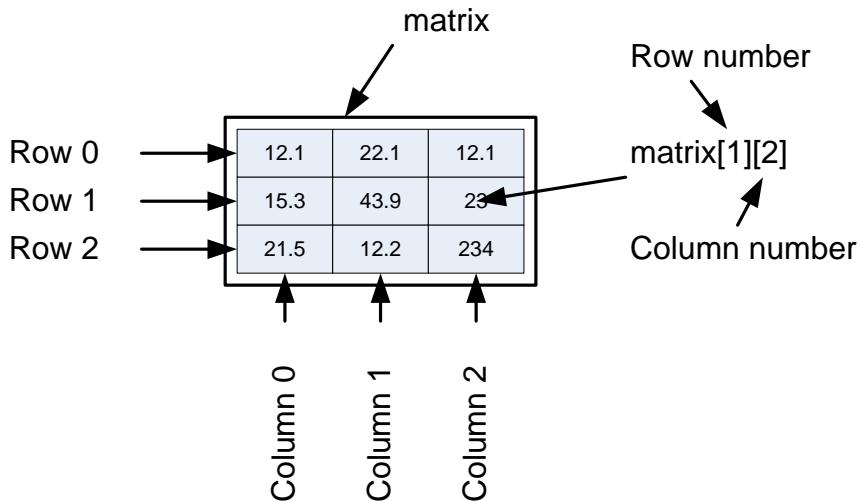
### 3.2. 2D Arrays

2D array is suitable to hold tabular information for example, it can be used to store 4 students degrees in 6 subjects or to represent  $(m \times n)$  matrix.

To define an array variable containing 100 (**float**) values:

<code>float matrix[3][3];</code>	<p style="text-align: center;">2D Array Variable Name</p> <p style="text-align: center;">↓</p> <p style="text-align: center;"><code>float matrix[3][3];</code></p> <p style="text-align: center;">Item data type      Number of Rows      Number of Columns</p>
----------------------------------	---

Following figure shows how to represent the 2D array in computer memory:



Following code section shows how to use 2D arrays:

```
//define and initialize the 2D array
int degrees[4][6] = {
    {36, 28, 76, 47, 82, 33},
    {75, 49, 38, 98, 59, 83},
    {82, 65, 10, 21, 86, 22},
    {25, 63, 65, 76, 37, 21}};

int x = 9;
int y;

//assign x value to the item (2, 3)
degrees[2][3] = x;

//assign item (2,3) to y
y = degrees[2][3];

//scan item (1, 4) from user
scanf("%f", &degrees[1][4]);

//print item (1, 4) from user
printf("The provided item value is %f", degrees[1][4]);
```

### Example 1: Scan 3x3 Matrix

Following example shows how to scan 3x3 matrix then calculate and prints its determinant.

m11	m12	m13
m21	m22	m23
m31	m32	m33

```

Determinant =
m11 * (m22*m33-m32*m23) -
m12 * (m21*m33-m31*m23) +
m13 * (m21*m32-m31*m22)

```

```

#include "stdio.h"

void main()
{
    float a[3][3];
    int r, c;
    float determinant;

    for(r=0; r<3; r++)
    {
        for(c=0; c<3; c++)
        {
            printf("Enter the item(%d, %d) : ", r, c);
            scanf("%f", &a[r][c]);
        }
    }

    determinant =
        a[0][0] * (a[1][1] * a[2][2] - a[2][1] * a[1][2]) -
        a[0][1] * (a[1][0] * a[2][2] - a[2][0] * a[1][2]) +
        a[0][2] * (a[1][0] * a[2][1] - a[2][0] * a[1][1]);

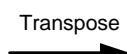
    printf("matrix determinant is %f\r\n", determinant);
}

```

### Example 2: Calculate and Print the Transpose of 3x3 Matrix

Following example calculates and prints the transpose of a 3x3 matrix.

m11	m12	m13
m21	m22	m23
m31	m32	m33



m11	m21	m31
m12	m22	m32
m13	m23	m33

```

#include "stdio.h"

void main()
{

```

```

float a[3][3] = { {2, 3, 4},
                  {4, 8, 9},
                  {8, 5, 2}};
float t[3][3];
int r, c;

for(r=0; r<3; r++)
{
    for(c=0; c<3; c++)
    {
        t[r][c] = a[c][r];
    }
}

for(r=0; r<3; r++)
{
    for(c=0; c<3; c++)
    {
        printf("%2.2f\t", t[r][c]);
    }
    printf("\r\n");
}
}
}

```

```

C:\Windows\system32\cmd.exe
2.00 4.00 8.00
3.00 8.00 5.00
4.00 9.00 2.00
Press any key to continue . . .

```

### 3.3. Strings

String is a set of several consecutive characters; each character can be represented in C language by a (char) data type. This means that string value can be represented by an array of (char).

```

char Text[] = {'h', 'e', 'l', 'l', 'o', 0};

```

Above code shows how to store “hello” string letters in array. The last letter is sited with zero value (null termination), this value is important to tell the computer that the string is terminated before this item.

As shown string is an ordinary array with (char) values. Due to the importance of string values in any computer program, C language provides special support for string values.

To simplify string manipulation you can assign normal text values to arrays, you do not need to divide string value into characters, also you do not have to supply the null termination value; it is implicitly supplied.

```
char text1[] = {"hello"};
char text2[] = "hello";
```

### Printing String Value

```
#include "stdio.h"

void main()
{
    char text[100] = {'h', 'e', 'l', 'l', 'o', 0};
    printf("%s\r\n", text);
}
```

“**text**” variable contains 100 (char) value, only first 5 places is used to hold the word “hello” and the sixth place is used to hold the null termination.

In **printf**, “%s” is used to inform the program that it will print a string value.

**Important:** **printf** uses the null termination to end the printing operation. If the null termination is not used, the program continues printing the following memory contents until it reaches a zero.

### Scanning String Value

```
#include "stdio.h"

void main()
{
    char text[100];
    printf("Enter your first name\r\n");
    scanf("%s", text);
    printf("Your Name is %s\r\n", text);
}
```

**scanf** is used with “%s” to scan string input from keyboard. However there is a problem? **scanf** takes only the first word in the input text and leave the rest.

```
C:\Windows\system32\cmd.exe
Enter your first name
Ahmed Mostafa
Your Name is Ahmed
Press any key to continue . . .
```

C language provides a solution for above problem using **gets** function.

```
#include "stdio.h"
#include "conio.h"

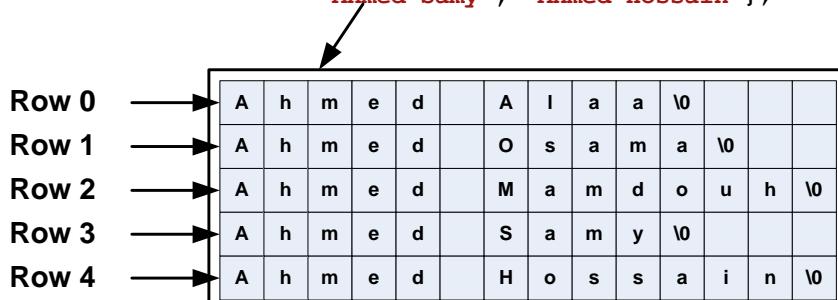
void main()
{
    char Text[100];
    printf("Enter your first name\r\n");
    gets(Text);
    printf("Your Name is %s\r\n", Text);
}
```

```
C:\Windows\system32\cmd.exe
Enter your first name
Ahmed Mostafa
Your Name is Ahmed Mostafa
Press any key to continue . . .
```

## Array of Strings

In order to store several names, it is required to use 2D arrays of type (char). Each row is used to store one string value. The number of columns must be greater than or equal to the (maximum string length + 1). Following figure shows how to store five names in 2D array.

```
char names[5][14] = {"Ahmed Alaa", "Ahmed Osama", "Ahmed Mamdouh",
                     "Ahmed Samy", "Ahmed Hossain"};
```

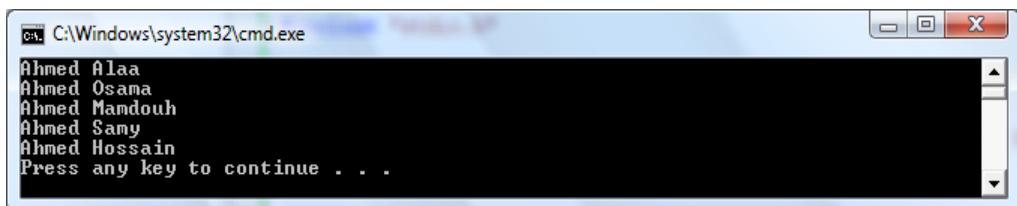


### Example 1: Printing Array of Strings

```
#include "stdio.h"

void main()
{
    char names[5][14] = {"Ahmed Alaa", "Ahmed Osama", "Ahmed
    Mamdouh", "Ahmed Samy", "Ahmed Hossain"};
    int i;

    for(i=0; i<5; i++)
        printf("%s\r\n", names[i]);
}
```



### Example 2: Copy String to String

What if you need to copy string variable to another variable? It is required to copy each character until the null termination is reached. Following example illustrate this idea:

```
#include "stdio.h"

void main()
{
    char a[20] = "Alaa Ezzat";
    char b[20];
    int i = 0;

    while(a[i]!=0)
    {
        b[i] = a[i];
        i++;
    }
    b[i] = 0; //Add null termination to the end of B

    printf("%s\r\n", b);
}
```

There is another solution to above problem using **strcpy** function. **strcpy** takes both the destination and the source strings and performs the coping operation internally.

```
#include "stdio.h"
#include "string.h"

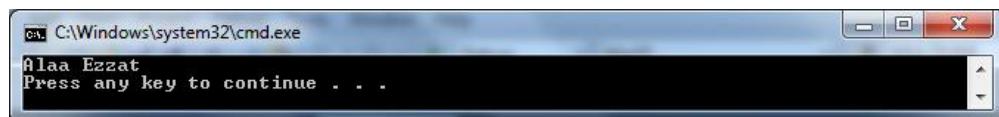
void main()
{
    char a[20] = "Alaa Ezzat";
    char b[20];
    strcpy(b, a);
    printf("%s\r\n", b);
}
```

### Example 3: Adding String to String

String addition means concatenating the second string characters at the end of the first string.  
This can be made using **strcat** function.

```
#include "stdio.h"
#include "string.h"

void main()
{
    char a[20] = "Alaa";
    char b[20] = "Ezzat";
    strcat(a, " ");
    strcat(a, b);
    printf("%s\r\n", a);
}
```



### Example 4: Changing String Case

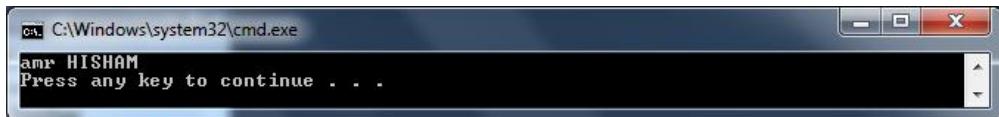
**strlwr** function changes all string letters to the lower case.  
Ex: “AhMed” → “ahmed”

**strupr** function change all string letters to the upper case.  
Ex: “aHmeD” → “AHMED”

```
#include "stdio.h"
#include "string.h"

void main()
```

```
{
    char a[20] = "Amr";
    char b[20] = "Hisham";
    strlwr(a);
   strupr(b);
    printf("%s %s\r\n", a, b);
}
```



### Example 5: Finding the String Length

**strlen** function calculates the given string length, it simply counts the number of letters until the null termination.

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

void main()
{
    char name[] = "Ahmed Said";
    printf("Name: %s, Length: %d\r\n", name, strlen(name));
}
```

### Example 6: Comparing Two Strings

**strcmp** function compares two strings and produces one of three results:

- if the two string are identical it gives 0
- if the first string is lower in the alphabetic order it gives -1
- if the second string is higher in the alphabetic order it gives 1

“ahmed” and “ahmed” → 0

“ahmed” and “amgad” → -1 because the second letter ‘h’ is less than ‘m’

“maged” and “aya” → 1 because the first letter ‘m’ is greater than ‘a’

**strcmp** differentiate between capital and small letters which means “MAGED” is less than “ahmed”, because the ASCII code of the capital letters is lower than the ASCII code of the small letters. To solve this problem you can change the case of both strings to the same case then use **strcmp** function. Alternatively you can use **strcmp** function which performs the comparison independent on the string case.

Following program scans user name and searches for his name in an internal list.

```

#include "stdio.h"
#include "string.h"

void main()
{
    char names[5][14] = {"Alaa", "Osama", "Mamdouh", "Samy",
    "Hossain"};
    char name[14];
    int i;

    printf("Enter your name : ");
    scanf("%s", name);

    for(i=0;i<5;i++)
    {
        if(strcmp(name, names[i])==0)
        {
            printf("Congratulation,
                    your name is in the list");
            break;
        }
    }

    if(i==5)
        printf("We are sorry, your name is not listed");
}

```

### Example 7: Converting String to Integer Value

What if string variable contains some numeric information and it is required to find its numeric value?

```

#include "stdio.h"
#include "string.h"

void main()
{
    char text[20] = "1025";
    int x = 200;
    int y;

    y = x + text; //wrong
}

```

The variable text contains 4 ASCII letters ‘1’, ‘0’, ‘2’, ‘5’. The statement (x + text) is completely wrong, because text is string and not a number, computer cannot understand string contents directly.

**atoi** function helps computer to convert string to a number of type (**int**) if it is applicable otherwise it gives zero, for example:

“1025” → 1025

“120KG” → 120

“The Cost is 30” → 0

```
#include "stdio.h"
#include "stdlib.h"

void main()
{
    char text[20];
    int x;
    int y;
    int z;

    printf("Enter x : ");
    gets(text);
    x = atoi(text);

    printf("Enter y : ");
    gets(text);
    y = atoi(text);

    z = x + y;
    printf("x + y = %d\r\n", z);
}
```

Converting String to Real Value

**atof** function helps computer to convert string to a number of type (float) if it is applicable otherwise it gives zero, for example:

“10.25” → 10.25

“1.2KG” → 1.2

“The Cost is 3.5” → 0

### 3.4 Exercises

#### Q1.

It is required to evaluate the polynomial  $f(x)$  at  $x=3$ . The polynomial coefficients are:  $\{a_0=29, a_1=-81, a_2=27, a_3=-98, a_4=21, a_5=-83, a_6=78, a_7=-93, a_8=72, a_9=8\}$ . Write a program to evaluate the polynomial without using the arrays.

#### Q2.

Solve Q1 using arrays to store the polynomial coefficients.

#### Q3.

Giving the following numbers, write a program that finds the maximum, the minimum, the mean ( $\mu$ ), the variance ( $V$ ), and the standard deviation ( $\sigma$ ) values.

The numbers are:  $\{84, 63, 21, 78, 82, 19, 83, 47, 23, 78, 54, 60, 91, 23, 29, 48, 37, 26\}$

Know that:

$$\mu = \frac{\sum_{i=1}^n a_i}{n}$$

$$V = \frac{\sum_{i=1}^n (a_i - \mu)^2}{n}$$

$$\sigma = \sqrt{V}$$

#### Q4.

Giving the following numbers, write a program that calculates the average, then counts and prints the number of values above and below the average value.

The numbers are:  $\{84, 63, 21, 78, 82, 19, 83, 47, 23, 78, 54, 60, 91, 23, 29, 48, 37, 26\}$

#### Q5.

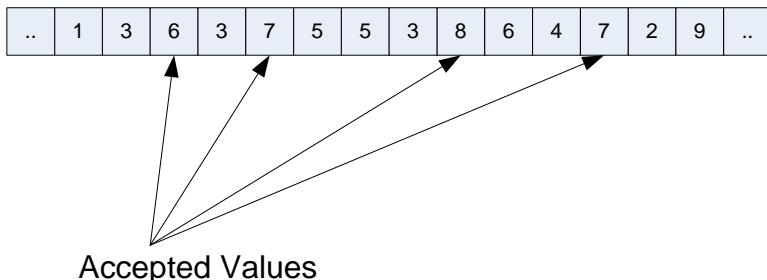
Given the following array, write a program that reverses all array elements then prints them.

```
int x[] = {1,2,3,4,5,6,7,8,9,10};
```

#### Q6.

Define a 1D array of 1000 element and fill it with random values (using rand() function) between 1 and 1000, then searches and counts the number of values that satisfy the following condition:

The previous and the next values is lower than the center value



#### Q7.

Define a 100x100 2D array and fill it with random values between 1 and 1000, then search and count the number of values that satisfy following condition:

All 8 neighbors' values are less than the center value

..	..	..	..	..
..	1	3	6	..
..	3	7	5	..
..	3	3	4	..
..	..	..	..	..

Accepted

..	..	..	..	..
..	2	9	9	..
..	5	3	8	..
..	6	4	7	..
..	..	..	..	..

Rejected

### Q8.

For the given paragraph write a program that computes the frequency of all alphabetic letters (how many each letter from a to z is repeated). Your program must be case insensitive.

C is a general purpose computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system. Although C was designed for implementing system software, it is also widely used for developing portable application software. C is one of the most popular programming languages. It is widely used on many different software platforms, and there are few computer architectures for which a C compiler does not exist. C has greatly influenced many other popular programming languages, most notably C++, which originally began as an extension to C.

### Q9.

Write and discuss the expected output of the following programs:

```
#include "stdio.h"
#include "string.h"

void main()
{
    char text1[] = "Welcome";
    char text2[100] = "Welcome";
    char text3[50] = {'W','e','l','c','o','m','e'};

    printf("1. size = %d, length = %d\r\n",
           sizeof(text1), strlen(text1));
    printf("2. size = %d, length = %d\r\n",
           sizeof(text2), strlen(text2));
    printf("3. size = %d, length = %d\r\n",
           sizeof(text3), strlen(text3));
}
```

```
#include "stdio.h"
#include "string.h"

void main()
{
```

```

char text[] = "Welcome to computer world";

printf("%s\r\n", text);
printf("size = %d, length = %d\r\n",
       sizeof(text), strlen(text));

text[10] = 0;
printf("%s\r\n", text);
printf("size = %d, length = %d\r\n",
       sizeof(text), strlen(text));
}

```

**Q10. (Report)**

Write a program that counts the number of full words in the paragraph of Q8.

**Q11.**

Write a program that replaces the word “computer” with the word “programming” in the following paragraph. Hint: Use another bigger array to produce the new string.  
Welcome to computer world. Welcome to computer world.

**Q12.**

Write a program that counts the number of words “programming” in the paragraph of Q8.

**Q13.**

Write a program that computes the summation of the following numeric values. Consider that they are provided in a string format.

```

char values[10][3] = {"93", "28", "80", "93", "28", "40",
"98", "32", "45", "25"};

```

## Chapter 4: Functions

## 4.1. Introduction

Consider the following problem? it is required to evaluate the equation:

$$z = \frac{m(3,2) + \sqrt{m(6,1.5)^{3.2} + m(5,3.4)}}{m(13,1.2)} \text{ where } m(x,y) = 5(x+y)^2 + 3x - 2y + 2.$$

Solution:

```
#include "stdio.h"
#include "math.h"

void main()
{
    float x;
    float y;
    float z;
    float m1, m2, m3, m4;

    //Evaluate m1 = m(3,2);
    x = 3;
    y = 2;
    m1 = 5 * (x+y)*(x+y) + 3*x + 2*y + 2;

    //Evaluate m2 = m(6,1.5);
    x = 6;
    y = 1.5;
    m2 = 5 * (x+y)*(x+y) + 3*x + 2*y + 2;

    //Evaluate m3 = m(5,3.4);
    x = 5;
    y = 3.4;
    m3 = 5 * (x+y)*(x+y) + 3*x + 2*y + 2;

    //Evaluate m4 = m(13,1.2);
    x = 13;
    y = 1.2;
    m4 = 5 * (x+y)*(x+y) + 3*x + 2*y + 2;

    //Finaly Calculate z
    z = (m1 + sqrt(pow(m2, 3.2f) + m3))/m4;
    printf("z = %f\r\n", z);
}
```

**Comment:** the program works correctly, however it appears that the code size contains many repeated sections, the repetition comes from the need of calculating  $m(x, y)$  several times while calculating the  $(z)$  value.

## Alternative Solution using Function

```

#include "stdio.h"
#include "math.h"

float calcm(float x, float y) //Function
{
    float m;
    m = 5 * (x+y)*(x+y) + 3*x + 2*y + 2;
    return m;
}

void main()
{
    float z;

    z = (calcm(3,2) +
        sqrt(pow(calcm(6,1.5), 3.2f) +
              calcm(5,3.4)))/calcm(13,1.2);

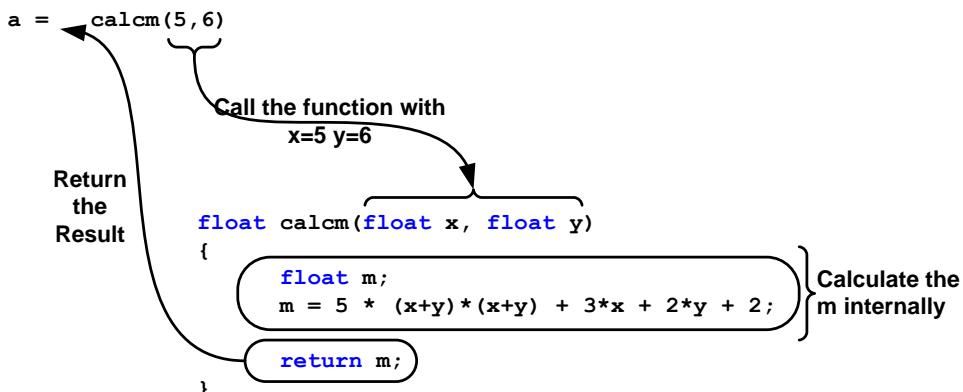
    printf("z = %f\r\n", z);
}

```

**Comment:** using function (**calcm**) reduces the code size and makes it more readable.

#### Understanding **calcm** function:

**calcm** function is a small program, simply it takes the values of x and y, calculating m internally, then return the calculated value.



The call **a = calcm(5, 6)** works as follows:

1. Copies the values 5 and 6 to the variables x and y
2. Performs the internal calculations to calculate m

3. When executing the line (**return m**), the computer copies the value inside m and return it to the line (**a = calcm(5, 6)**)
4. Copies the return value to (**a**) variable

Whenever you need to calculate m at any x and y just call **calcm(x,y)**, for example:

```
a = calcm(5, 6);

b = calcm(5, 6)/calcm(2.2,1.2);

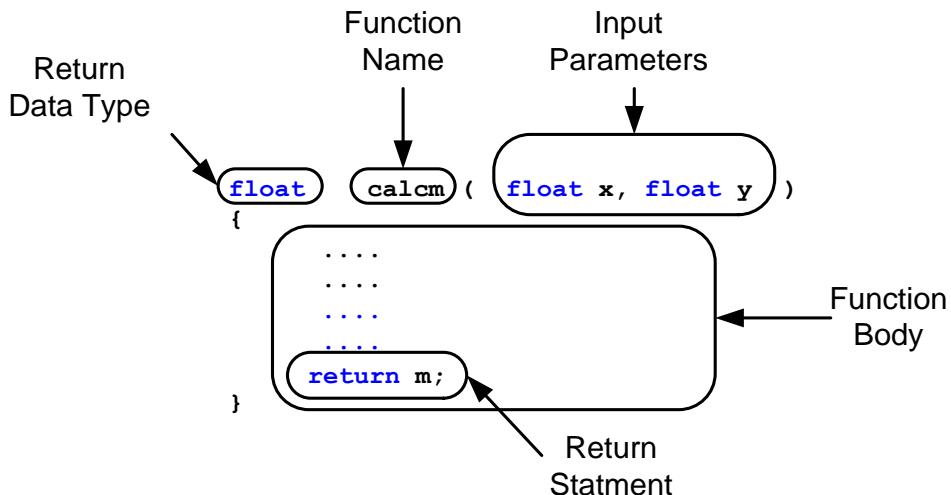
c = sin( calcm(5,6) );

printf("result = %f", calcm(2.1, 2))
```

## 4.2. Function Definition

Function is a small program, you can call it from anywhere in your program. Calling function means using a function to do its operation.

The function consists of the following parts:



**Function Name**: like variable, must have no spaces and no special characters and must starts with letters

**Input Parameters**: supplied parameters types and names. You can define any number of inputs; also you can define zero number of inputs.

**Return Type**: the data type of the function output, if the function has no output use (**void**) keyword.

**Function Body:** performs specific function operation.

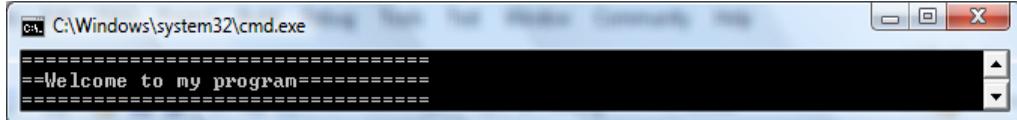
**Return Statement:** this statement tells the computer that the function execution is completed and the required function output is ready for the caller. The computer takes the returned value and supplies it to the caller.

**Important:** sometimes functions does not take any input and does not return an output, this means that the function performs only an internal operation. Following example clarifies this idea.

```
#include "stdio.h"

void printWelcome()
{
    printf("=====\\r\\n");
    printf("=====Welcome to my program=====\\r\\n");
    printf("=====\\r\\n");
}

void main()
{
    printWelcome();
}
```



The function **printWelcome** prints a decorated welcome message; it does not take any input parameter or return any result.

Where to define the function? To answer these questions try the following program.

```
#include "stdio.h"

void main()
{
    printWelcome();
}

void printWelcome()
{
    printf("=====\\r\\n");
    printf("=====Welcome to my program=====\\r\\n");
    printf("=====\\r\\n");
}
```

The compiler gives an error at the line `printWelcome()`; in the main function, the error state that “the function `printWelcome` is undefined”. Which means that the compiler cannot locate the function before the main, even if it is located after the main?

To solve this problem, you have to move the function before the main, or to move only the function prototype, as shown in the following program.

```
#include "stdio.h"

//Function Prototype
void printWelcome();

void main()
{
    printWelcome();
}

void printWelcome()
{
    printf("=====\\r\\n");
    printf("=====Welcome to my program=====\\r\\n");
    printf("=====\\r\\n");
}
```

### 4.3. Parameters Types

Function parameters can take any data type like `char`, `int` and `float`, also it can be single values or arrays. Following examples illustrates how to pass different parameter types.

Example 1 shows how to send a single value. Example 2 shows how to send a 1D array. Example 3 shows how to send a 2D array.

#### Example 1: Calculate the Factorial

Following program uses a function to calculate the factorial of any positive number.

Factorial of  $x$  means  $x! = (x) \times (x - 1) \times (x - 2) \dots \times (3) \times (2) \times (1)$

For example:  $5! = 5 \times 4 \times 3 \times 2 \times 1$

Specially  $0! = 1$

```
#include "stdio.h"

int factorial(int x)
{
    int f = 1;
    for(;x>0;x--)
```

```

        f *= x;
    return f;
}

void main()
{
    printf("Factorial(%d) = %d\r\n", 10, factorial(10));
    printf("Factorial(%d) = %d\r\n", 0, factorial(0));
    printf("Factorial(%d) = %d\r\n", 5, factorial(5));
}

```

**Example 2: Calculate the Minimum Value of any Given Array**

Following program uses a function to calculate the minimum value of any given array.

```

#include "stdio.h"

int calcMin(int values[], int n)
{
    int i, minValue = values[0];
    for(i=0; i<n; i++)
    {
        if(values[i]<minValue)
            minValue = values[i];
    }
    return minValue;
}

void main()
{
    int xvalues[10] = {35, 67, 27, 54, 76,
                      44, 59, 32, 43, 25};
    int yvalues[5] = {28, 71, 67, 83, 62};
    int zvalues[13] = {87, 21, 74, 36, 27,
                      64, 87, 63, 27, 86, 48, 32, 76};

    printf("The minimum of x, y, z values is
          (%d, %d, %d)\r\n",
          calcMin(xvalues, 10),
          calcMin(yvalues, 5), calcMin(zvalues, 13));
}

```

**Important:** `calcMin` function takes two parameters an array and (`int`) value containing the array size. Know that function could not expect the given array size, you must supply it by yourself.

### Example 3: Finding a Name in a Set of Names

Following program uses a function to search for a specified name in a set of names.

```
#include "stdio.h"
#include "string.h"

int findName(char names[][14], int n, char name[])
{
    int i = 0;
    for(i=0;i<n;i++)
    {
        if(strcmp(names[i], name)==0)
            return 1;
    }
    return 0;
}

void main()
{
    char name[14];
    char names[5][14] = {"Alaa", "Osama", "Mamdouh",
                         "Samy", "Hossain"};
    puts("Enter your first name:");
    gets(name);
    if(findName(names, 5, name)==1)
        puts("Welcome");
    else
        puts("Goodby");
}
```

### Difference between Passing Single Values and Arrays

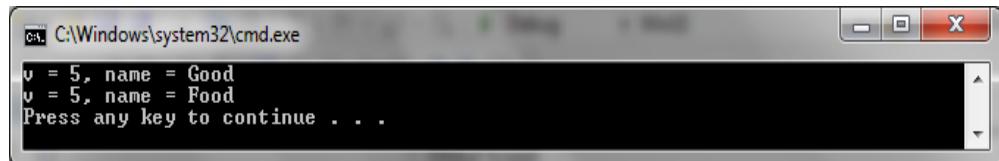
Both methods leads to carry the value of the input parameter to be used inside the function, however there is a major difference between single values and arrays. In **single values case** only the value of the parameter is passed. In **arrays case** only the address of the array is passed to avoid coping all array data, which means that if the function tries to modify the input array the original array will be modified also. Following example illustrates this idea:

```
#include "stdio.h"

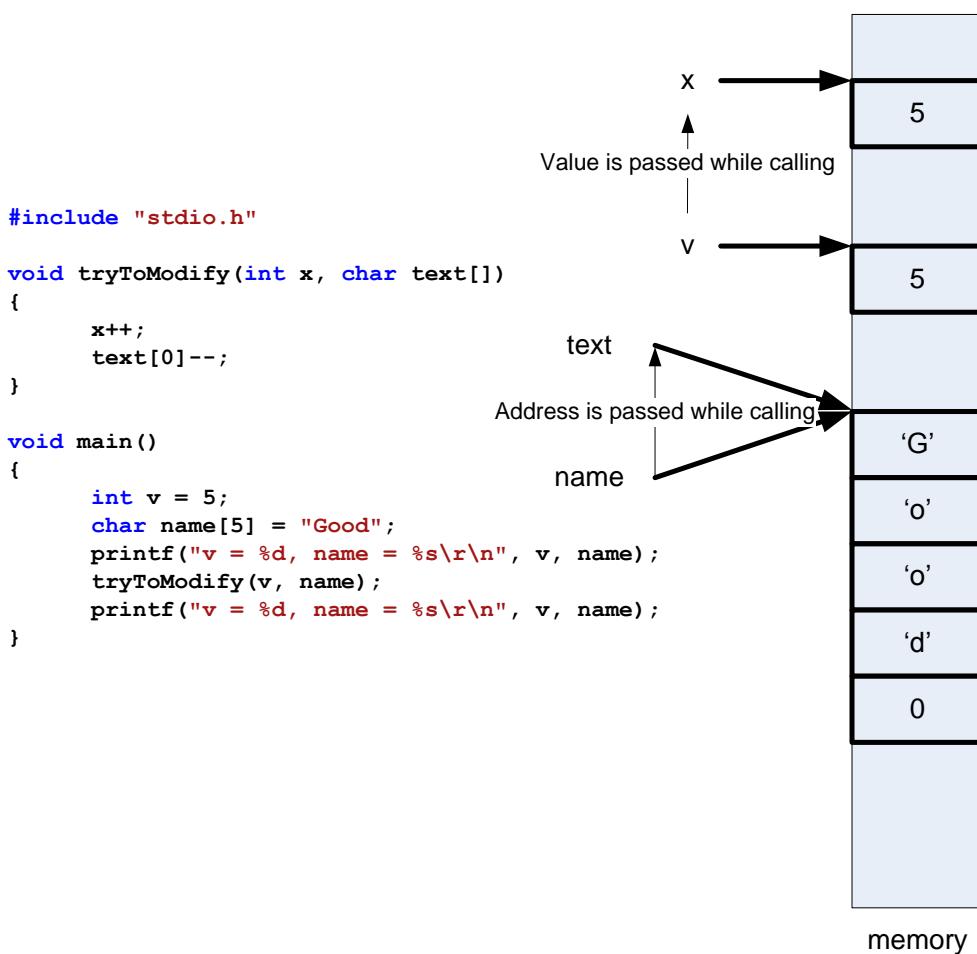
void tryToModify(int x, char text[])
{
    x++;
    text[0]--;
}

void main()
```

```
{  
    int v = 5;  
    char name[5] = "Good";  
    printf("v = %d, name = %s\r\n", v, name);  
    tryToModify(v, name);  
    printf("v = %d, name = %s\r\n", v, name);  
}
```



It appears that the (**tryToModify**) function failed to modify the single value (v) however it succeeded to modify the array (name) because both name and text will point to the same address when the function (**tryToModify**) is called. Following figure illustrates this idea:



## 4.4. Variables Scope

### 4.4.1. Local Variables

Any program consists of a set of functions; each one has its own local variable including the parameters. Other functions variables are not accessible. Following figure illustrate this idea:

```
#include "stdio.h"

int myMull(int x, int y)
{
    int z;
    z = x * y;
    return z;
}

int myAdd(int x, int y)
{
    int z;
    z = x + y;
    return z;
}

void main()
{
    int a = 5, b = 6;
    printf("a + b = %d\r\n", myAdd(a,b));
    printf("a * b = %d\r\n", myMull(a,b));
}
```

The diagram illustrates the scope of variables in the provided C program. It uses curly braces to group the code into three distinct scopes:

- myMull Scope**: Contains local variables `x`, `y`, and `z`.
- myAdd Scope**: Contains local variables `x`, `y`, and `z`.
- main Scope**: Contains local variables `a`, `b`, and `c`.

Above program have three functions each with different scopes; each scope holds different local variables. The variable **a**, **b** and **c** of main function are inaccessible through **myAdd** or **myMull** functions. The variables **x**, **y** and **z** of **myMull** function are inaccessible through **myAdd** or **main** functions, even if **myAdd** function has the same variables names.

#### 4.4.2. Global Variables

Global variables can be accessed anywhere, any function can read or write to those variables. Global variables are defined outside all function bodies, which mean that it does not belong to any function. Following program illustrates this idea:

#### Example 4: Using Global Variables

In the following program the variable name is defined as a global variable, all program function can access this variable.

```
#include "stdio.h"

char gName[50];

void welcome()
{
    printf("Welcome %s\r\n", gName);
```

```

void goodby()
{
    printf("Goodby %s\r\n", gName);
}

void main()
{
    puts("Enter your name:");
    gets(gName);
    welcome();
    goodby();
}

```

#### 4.4.3. Static Variables

Static variables are defined by the modifier **static**. Static variables are initialized once in the program life. For example if the variable (**x**) is defined inside a function, the variable is initialized only at first function call, further function calls do not perform the initialization step, this means that if the variable is modified by any call the modification result remains for the next call. Following example illustrate this idea:

#### Example 5: Using Static Variables

In the following program the function **myprint** contains two variables **x** and **y**. **x** is defined as **static** variable. Both variables are initialized with zero then incremented.

The **myprint** function is called 10 times. For non-static variable **y** each time the variable is initialized by (0) then incremented by one, for that reason the printed value is always (1). For static variable **x** the initialization is performed only at first call, for that reason the printed value is incremented by one starting form (1) to (10).

```

#include "stdio.h"

void myprint()
{
    static int x = 0;
    int y = 0;
    x++;
    y++;
    printf("x = %d, y = %d\r\n", x, y);
}

void main()
{
    int i;
    for(i=0;i<10;i++)
        myprint();
}

```

```
}
```

```
C:\Windows\system32\cmd.exe
x = 1, y = 1
x = 2, y = 1
x = 3, y = 1
x = 4, y = 1
x = 5, y = 1
x = 6, y = 1
x = 7, y = 1
x = 8, y = 1
x = 9, y = 1
x = 10, y = 1
Press any key to continue . . . -
```

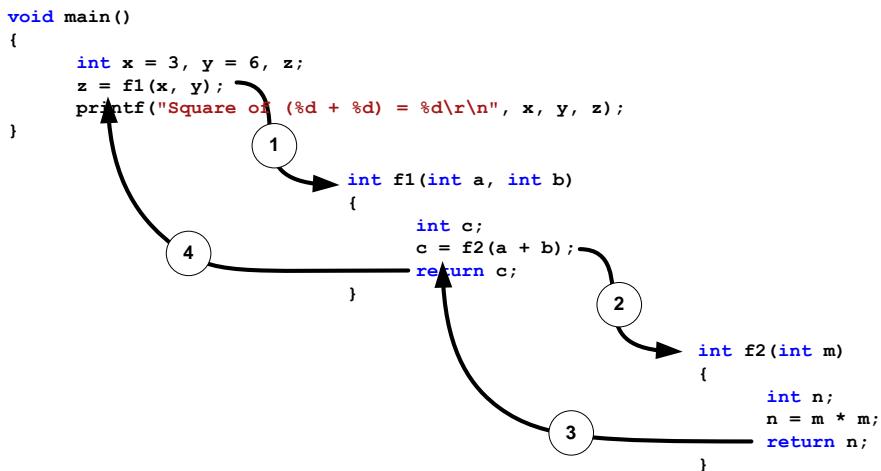
#### 4.5. Calling Mechanism

When the function is called all local variables are created in a private memory section called **Stack Memory**. After returning from the function all local variables are destroyed and their locations in the Stack Memory are freed.

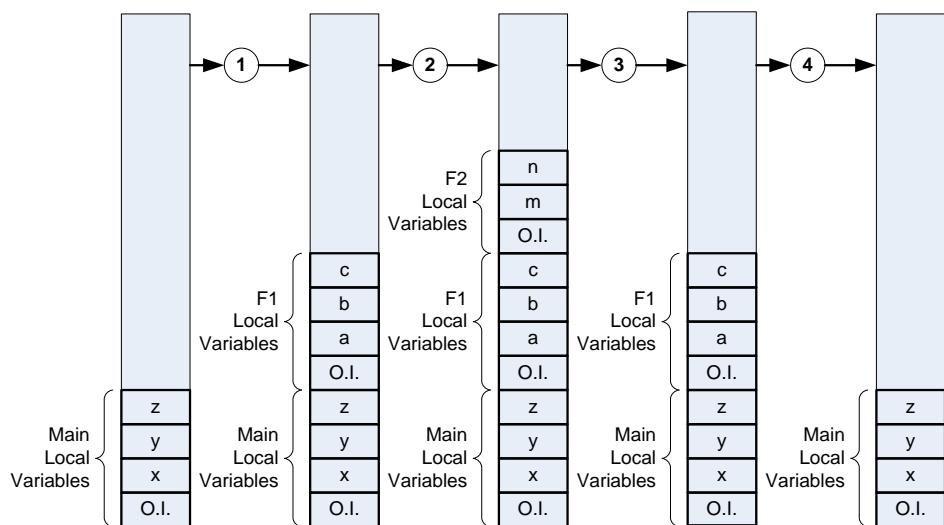
If the function calls another function the new function takes another area from the Stack Memory to place its local variables, if further nested calls are made further areas are taken.

The maximum nested call depth is limited by the size of the local variables used by each function and the total Stack Memory Size. Stack Memory size value can be configured from the project settings.

Following figure explains the calling mechanism for three nested calls. When the **main** routine is called by the system, all local variables (x, y and z) are placed at the top of the Stack Memory. When the **(f1)** function is called the local variables (a, b and c) are placed at the top of the Stack Memory. Finally when the **(f2)** function is called the local variables (m and n) are placed at the top of the Stack Memory. After returning from **f2** function the (m and n) variable are destroyed. After returning from **f1** function the (a, b and c) variable are destroyed.

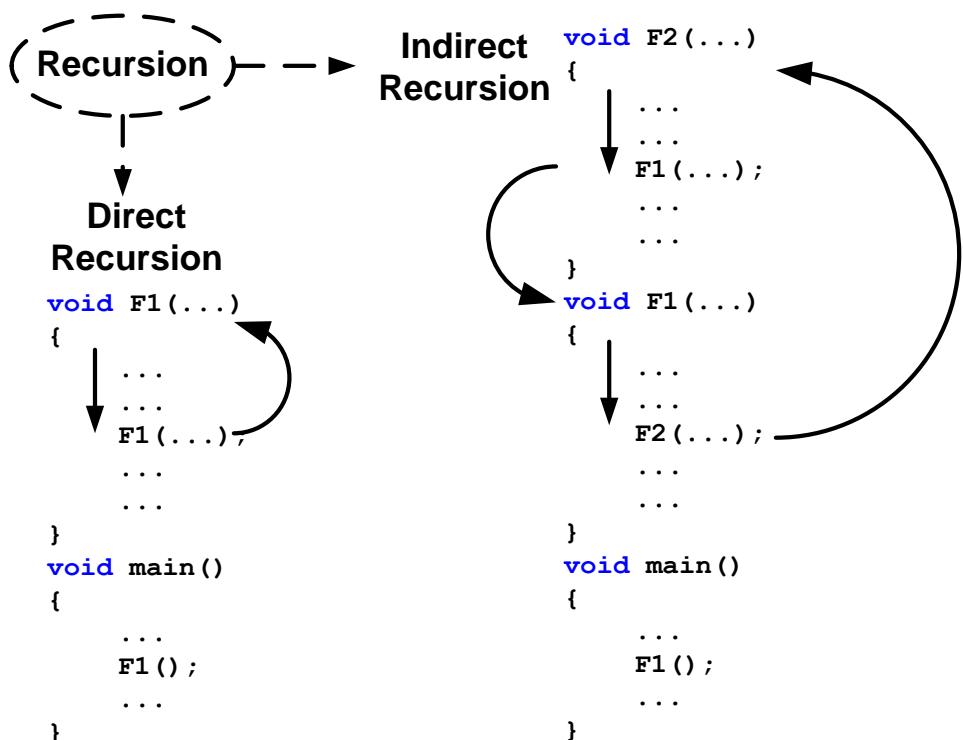


O.I. : Other Information like ... caller address



## 4.5. Recursion

Recursion is a situation happens when a function calls itself directly or indirectly. Following figure illustrates this situation:



Is recursion useful?

Recursion is an alternative way to repeat the function execution with the same or variant parameters values. In another way recursion is an indirect way to make a loop; simply you can convert any recursion to a normal loop. Following example illustrates this idea:

### Example 6: Infinite Printing Loop

This example shows how to implement a loop that prints a word “Hello” using either normal looping or recursion.

Normal Loop	Recursive Loop
<pre>#include &lt;stdio.h&gt;  void printheello() {     printf("hello\r\n"); }  void main() {     while(1)         printheello(); }</pre>	<pre>#include &lt;stdio.h&gt;  void printheello() {     printf("hello\r\n");     printheello(); }  void main() {     printheello(); }</pre>

**Note:** Above programs will not terminate, what is the required change to terminate them?

How to stop recursive function call?

Simply you can use one of the function parameters as a guide to determine the termination situation. For example if it is required to stop the printing operation in above example after 10 repetitions.

### Example 7: Finite Printing Loop

Finite Normal Loop	Finite Recursive Loop
<pre>#include "stdio.h"  void printheHello() {     printf("hello\r\n"); }  void main() {     int i = 0;     while((i++)&lt;10)         printheHello(); }</pre>	<pre>#include "stdio.h"  void printheHello(int downCounter) {     printf("hello\r\n");     downCounter--;     if(downCounter&gt;0)         printheHello(downCounter); }  void main() {     printheHello(10); }</pre>

### 4.5. Macros

Sometimes programmers repeat the writing of the same code segment all over the program. Sometimes the code segment is repeated exactly another times it repeated with some changes.

If the repeated **code size is large**, it is preferable to define a function for this code. If the repeated **code size is small**, Macros may be the best choice. Macros also do not affect the execution speed because there is no parameter passing in Macros.

### Example 8: Percentage Macro

Following program defines and uses a macro for calculating the percentage, it can be used anywhere in the program.

```
#include "stdio.h"

#define PR(value, maxValue) (100.0 * value/double(maxValue))

void main()
```

```

{
    int degree, maxDegree;

    printf("Enter subject degree followed by the
           maximum degree:\n\n");

    printf("Arabic:");
    scanf("%d %d", &degree, &maxDegree);
    printf("Arabic degree is %lf\n", PR(degree,maxDegree));

    printf("Math:");
    scanf("%d %d", &degree, &maxDegree);
    printf("Math degree is %lf\n", PR(degree,maxDegree));

    printf("Physics:");
    scanf("%d %d", &degree, &maxDegree);
    printf("Physics degree is %lf\n", PR(degree,maxDegree));

    printf("English:");
    scanf("%d %d", &degree, &maxDegree);
    printf("English degree is %lf\n", PR(degree,maxDegree));
}

```

Know that macros also save writing time; the compiler replaces macros with its code before performing the final compilation.

### Example 9: Exchange Macros

```

#include "stdio.h"

#define Exchange(value1, value2, temp) \
    temp = value1; \
    value1 = value2; \
    value2 = temp;

void main()
{
    int A = 10, B = 44, C;
    float X = 10.1, Y = 12.2, Z;
    printf("A = %d, B = %d, X = %f, Y = %f\n", A, B, X, Y);
    Exchange(A, B, C);
    Exchange(X, Y, Z);
    printf("A = %d, B = %d, X = %f, Y = %f\n", A, B, X, Y);
}

```

## 4.9. Coding Convention

### 4.9.1. Local, Global, Static Variables Naming

**First:** All variable naming rules specified in chapter 1 is used for local variable.

**Second:** Global variables must start with ‘g’ letter, for example:

```
int gCount;
```

**Third:** Static variables must start with ‘s’ letter, for example:

```
static int sCount;
```

### 4.9.2. Function Naming

Function name uses exactly the same naming convention of local variable naming, for example:

```
void sort(...)
```

## 4.8. Exercises

Q1.

Write the expected output of the following program:

```
#include "stdio.h"

void F1(int x, char text[])
{
    if(x>0)
    {
        printf("%s %d\r\n", text, x);
        F1(--x, text);
    }
}

void main()
{
    F1(5, "Welcome");
}
```

Q2.

Without using functions, write a program that calculates and prints the values of y for all x values starting from 0 to 10 with step 0.1.

$$y(x) = \frac{m(x^2) + m(5x)}{m(\sqrt{x})^{0.2}} \quad \text{where } m(x) = 7x^3 - 5x^2 + 2x + 11$$

Q3.

Using functions, write a program that calculates and prints the values of y for all x values starting from 0 to 10 with step 0.1.

$$y(x) = \frac{m(x^2) + m(5x)}{m(\sqrt{x})^{0.2}} \quad \text{where } m(x) = 7x^3 - 5x^2 + 2x + 11$$

Q4.

Write a function that check if the given integer value is prime or not. The function should return 1 if the number is prime and 0 if not. The function should have the following prototype:

```
void isPrime(int x);
```

Q5.

Write separate functions to calculate following quantities for any given double array:

- Minimum value
- Maximum value
- Average
- Variance
- Standard Deviation

Demonstrate the usage of your functions in a program.

Q6.

Write a function that takes any polynomial coefficients and evaluate the polynomial at certain value. The function should have the following prototype:

```
double polyEvaluate(double coefficients[], int order, double x);
```

Q7.

Write a function that takes any text and prints it in a reverse order.

Example: Welcome → emocleW

The function should have the following prototype:

```
void reversePrint(char text[]);
```

Q8.

Write a function that takes any text and prints each separate word in a line.

Example:

Welcome to computer World



Welcome  
to  
computer  
World

The function should have the following prototype:

```
void printLines(char text[]);
```

Q9.

Write a function that takes any text and returns the number of words inside the text.

```
void countWords(char text[]);
```

#### Q10. (Report)

Write a function that performs string replacement operation. The function takes two strings, the main text, the old text and new the new text. The function should search the main text for each occurrence of old text and replaces it with the new text. The function should return the number of replacements or -1 if the main text buffer size is not enough. The function should have the following prototype:

```
int replace(char text[], int maxTextSize,
           char oldText[], char newText[])
{ }
```

**Usage Example:**

```
char text[100] = "Welcome to computer world,
                  Welcome to computer world";

int n = replace("Welcome to computer world,
                  Welcome to computer world", sizeof(text),
                  "computer", "programming");
if(n>=0)
    printf("The replace succeeded,
           number of replacement = %d", n);
else
    printf("The replace failed,
           use larger text buffer");
```



## Chapter 5: Solving Engineering Problems, Part 1

## **5.1. Introduction**

C Programming language is commonly used to solve engineering problems, famous computer software like windows operating system, Linux operating system, Microsoft Office and 3D computer games are programmed in C.

Following sections demonstrate how to solve basic engineering problems using C.

## **5.2. Sorting a List of Values**

### **5.2.1. Problem Statement**

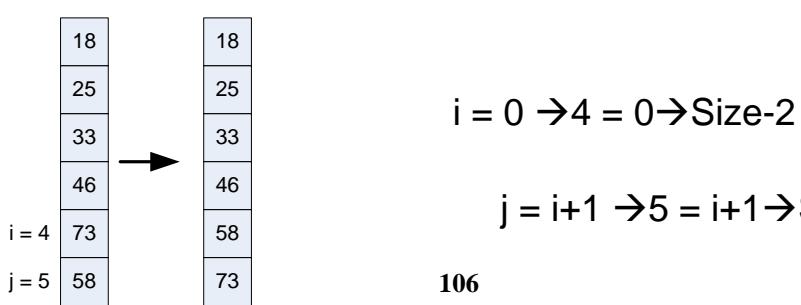
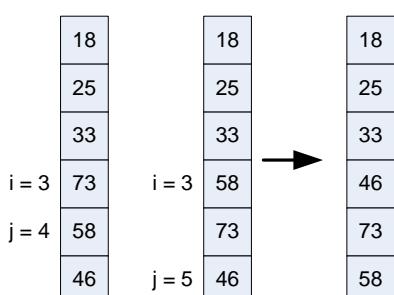
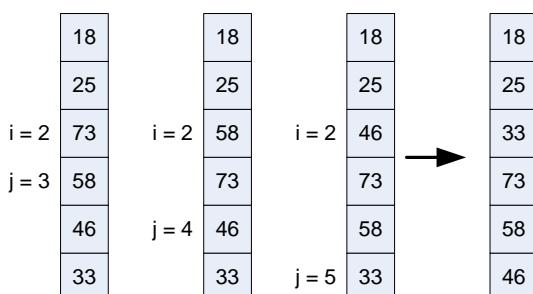
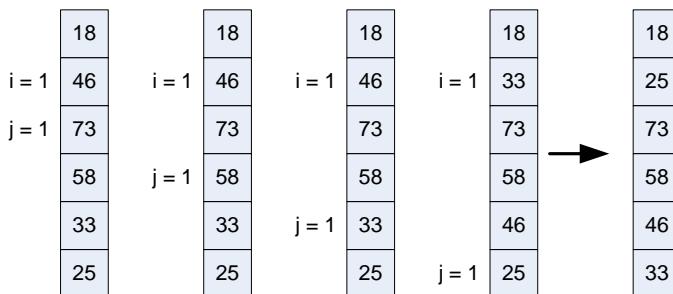
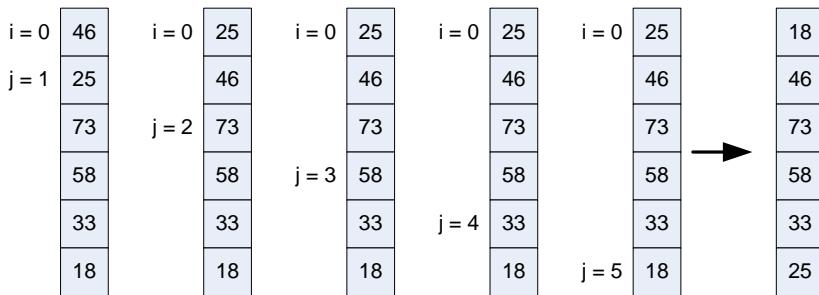
Given an array of numeric values, it is required to sort those values in ascending order. There are many algorithms to perform the sorting like quick sort, heap sort, binary tree sort and bubble sort. In this section we will use bubble sort algorithm to perform the sort.

### **5.2.2. Understand the Solution**

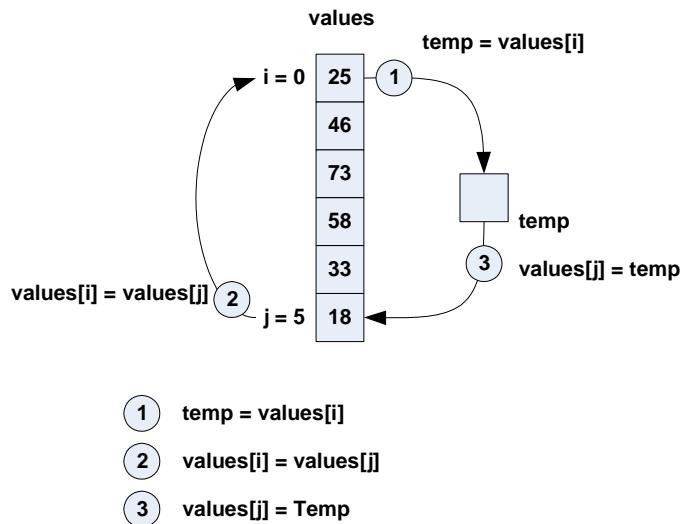
The bubble sort algorithm can be described as follows:

Given an array of size (n), for each position (i) in the array starting from 0 to n-2, if we compare this position contents (i) with all trailing position contents ( $i+1 \rightarrow n-1$ ) and exchange only if smaller contents is detected, the whole array will be sorted.

Following numeric example illustrate this idea.

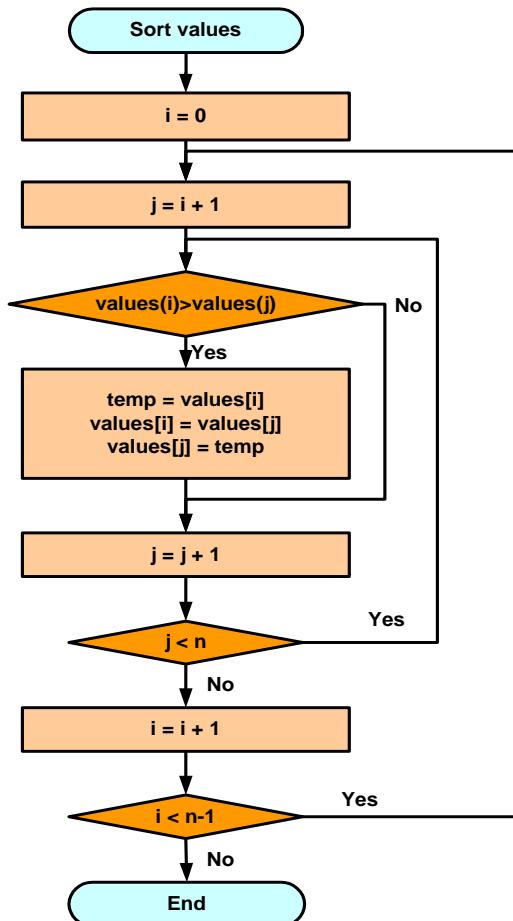


To exchange two values it is required to have a temporary third place. The exchange algorithm is illustrated in the following figure.



### 5.2.3. Design the Program

It is required to loop over array elements using the index (i) starting from (0) to ( $n-2$ ). For each (i) value it is required to compare it with item (j) starting from ( $i+1$ ) up to ( $n-1$ ). If the compared item (j) is less than item (i) the two items must be exchanged. Expert programmers can convert above ideas directly to a program, for complex problems it is required to illustrate the solution first using the flow chart.



#### 5.2.4. Write the Program

```
#include "stdio.h"

void main()
{
    int values[6] = {46, 25, 73, 58, 33, 18};
    int i, j, n = 6, temp;

    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(values[j]<values[i])
            {
                temp = values[i];
                values[i] = values[j];
                values[j] = temp;
            }
        }
    }

    //printing the sorted array
    for(i=0;i<n;i++)
        printf("%d\r\n", values[i]);
}
```

#### 5.2.5. Put the Solution in a Function

Placing computer algorithms in a function allows programmers to reuse their codes in any future programs.

```
#include "stdio.h"

void sort(int values[], int n)
{
    int i, j, temp;

    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(values[j]<values[i])
            {
                temp = values[i];
                values[i] = values[j];
                values[j] = temp;
            }
        }
    }
}
```

```

        values[j] = temp;
    }
}
}

void main()
{
    int values[6] = {46, 25, 73, 58, 33, 18};
    int i;

    sort(values, 6);

    //printing the sorted array
    for(i=0;i<6;i++)
        printf("%d\r\n", values[i]);
}

```

#### 5.2.6. Extending the Solution to Sort String Values

```

#include "stdio.h"
#include "string.h"

#define MAX_NAME_SIZE 50

void sort(char values[][MAX_NAME_SIZE], int n)
{
    int i, j;
    char temp[MAX_NAME_SIZE];

    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(strcmp(values[j], values[i])<0)
            {
                strcpy(temp, values[i]);
                strcpy(values[i], values[j]);
                strcpy(values[j], temp);
            }
        }
    }
}

void main()
{
    char names[5][MAX_NAME_SIZE] = {"Ahmed Alaa",
                                    "Ahmed Osama", "Ahmed Mamdouh",
                                    "Ahmed Samy", "Ahmed Hossain"};
}

```

```

int i;

sort(names, 5);

//printing the sorted array
for(i=0;i<5;i++)
    printf("%s\r\n", names[i]);
}

```

### 5.3. Calculating Polynomial Value

#### 5.3.1. Problem Statement

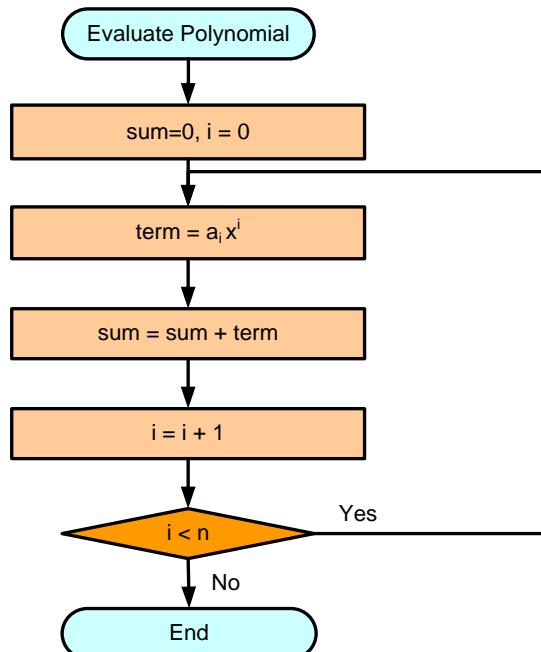
Consider the following polynomial form:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

Given an array of the coefficient values  $\{a_n, a_{n-1}, a_{n-2}, \dots, a_2, a_1, a_0\}$  and the value of  $x$ , evaluate the polynomial value.

#### 5.3.2. Design the Program

It is required to define a (sum) variable to hold the polynomial summation result, and then initialize it with zero, then loop over all coefficients and calculate the term  $a_i x^i$  and add the result to the (sum) variable.



### 5.3.3. Write the Program

```
#include "stdio.h"
#include "math.h"

double evaluatePolynomial(double coefficients[],
                           int n, double x)
{
    int i;
    double sum = 0, term;

    for(i=0;i<n;i++)
    {
        term = coefficients[i] * pow(x, i);
        sum += term;
    }

    return sum;
}

void main()
{
    double coefficients[10] = {1.2, -2.1, -6.4, 7.6,
                               -8.4, 6.3, -2.8, 1.7, 4.7, 8.0};
    double x = 1.2, f;
    f = evaluatePolynomial(coefficients, 10, x);
    printf("f(%lf) = %lf\r\n", x, f);
}
```

### 5.3.4. Optimize the Solution

Above program provides the solution, put not an optimized solution. Consider the polynomial of the 5<sup>th</sup> degree, above program will compute following terms:

$$x^5, x^4, x^3, x^2, x^1, x^0$$

By counting the number of multiplications:

$$4+3+2+1 \rightarrow \sum_{i=1}^{n-1} i \rightarrow 10 \text{ multiplications}$$

For a polynomial with degree = 20 the number of multiplications =  $\sum_{i=1}^{19} i = 190$

Considering that term  $x^5$  is related to term  $x^4$  by the formula  $x^5 = x \times x^4$ , this means we can reduce the number of multiplication by reusing previously multiplied terms.

$$\text{xterm} = 1 \rightarrow x^0$$

$$\text{xterm} = x * \text{xterm} \rightarrow x^1$$

$$\text{xterm} = x * \text{xterm} \rightarrow x^2$$

$$\text{xterm} = x * \text{xterm} \rightarrow x^3$$

$x_{term} = x * x_{term} \rightarrow x^4$   
 $x_{term} = x * x_{term} \rightarrow x^5$

By counting the number of multiplications  
 $1+1+1+1+1 \rightarrow 5$  multiplications (reduction ration  $10/5 \rightarrow 2$ )

For a polynomial with degree = 20 the number of multiplications is 20 (reduction ration  $190/20 \rightarrow 9.5$ )

It is clear that as the polynomial degree increase as the reduction ratio increase.

### 5.3.5. Apply the Optimization to the Program

```
#include "stdio.h"
#include "math.h"

double evaluatePolynomial(double coefficients[],
                          int n, double x)
{
    int i;
    double sum = 0, xterm = 1, term;

    for(i=0;i<n;i++)
    {
        term = coefficients[i] * xterm;
        sum += term;
        xterm *= x;
    }

    return sum;
}

void main()
{
    double coefficients[10] = {1.2, -2.1, -6.4, 7.6,
                             -8.4, 6.3, -2.8, 1.7, 4.7, 8.0};
    double x = 1.2, f;
    f = evaluatePolynomial(coefficients, 10, x);
    printf("f(%lf) = %lf\r\n", x, f);
}
```

## 5.4. Evaluating Series

### 5.4.1. Problem Statement

Consider the following series:

Finite Series

$$\sum_{m=0}^{10} \frac{2x^m}{m!} = \frac{2x^0}{0!} + \frac{2x^1}{1!} + \frac{2x^2}{2!} + \cdots + \frac{2x^{10}}{10!} = 2 + 2x + \frac{2x^2}{2!} + \cdots + \frac{2x^{10}}{10!}$$

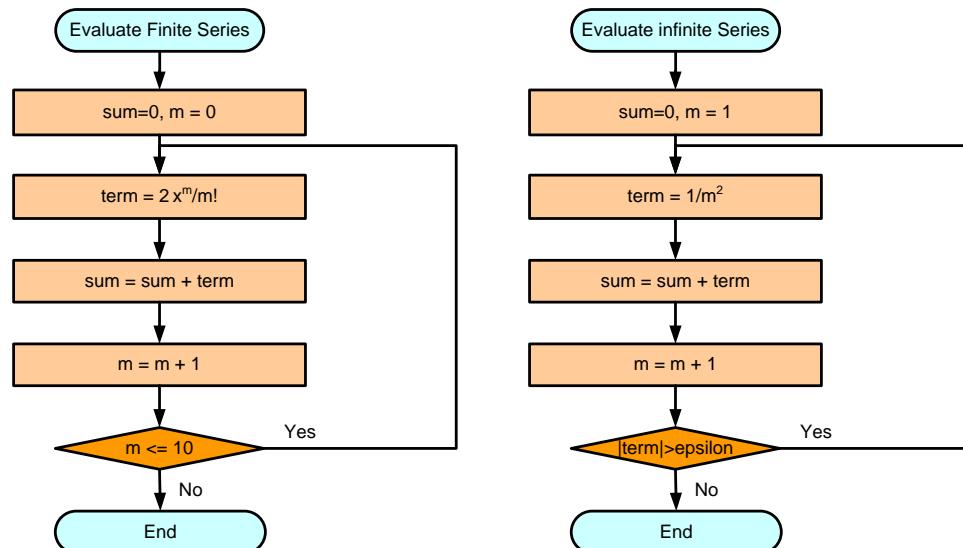
Infinite Series

$$\sum_{m=1}^{\infty} \frac{1}{m^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \cdots$$

Write a program that calculates the final result of each series.

### 5.4.2. Design the Program

For both series, it is required to define a (sum) variable to hold the polynomial summation result or each term. In finite series the number of iteration is determined. In infinite series the number of iteration is undetermined, term calculation and summation must continue until the absolute term value is nearly zero (epsilon).



**Very important**, while calculating the term you must optimize your calculation and reuse previously made calculations.

### 5.4.3. Write the Finite Series Program

```
#include "stdio.h"
#include "math.h"

int Factorial(int x)
{
    int f = 1;
    for(;x>0;x--)
        f *= x;
    return f;
}
void main()
{
    double sum = 0, term, x = 1.2;
    int m = 0;

    for(m=0;m<=10;m++)
    {
        term = 2 * pow(x, m)/Factorial(m);
        sum = sum + term;
    }

    printf("Value of series at x = %lf is %lf\r\n", x, sum);
}
```

### 5.4.4. Optimize the Finite Series Program

First Optimization, removing pow(x, m):

```
#include "stdio.h"

int Factorial(int x)
{
    int f = 1;
    for(;x>0;x--)
        f *= x;
    return f;
}
void main()
{
    double sum = 0, term, x = 1.2, xterm = 1;
    int m = 0;

    for(m=0;m<=10;m++)
    {
        term = 2 * xterm/Factorial(m);
```

```

        sum = sum + term;
        xterm *= x;
    }

    printf("Value of series at x = %lf is %lf\r\n", x, sum);
}

```

**Second Optimization:** removing the factorial calculation, considering that:

$$x! = x * (x-1)!$$

For  $x = 5$

$$5! = 5 * 4!$$

$$4! = 4 * 3!$$

$$3! = 3 * 2!$$

$$2! = 2 * 1!$$

$$1! = 1 * 0!$$

```

#include "stdio.h"

void main()
{
    double sum = 0, term, x = 1.2, xterm = 1, fterm = 1;
    int m = 0;

    for(m=0;m<=10;m++)
    {
        if(m>0) fterm *= m;
        term = 2 * xterm/fterm;
        sum = sum + term;
        xterm *= x;
    }

    printf("Value of series at x = %lf is %lf\r\n", x, sum);
}

```

#### 5.4.5. Write the Infinite Series Program

```

#include "stdio.h"
#include "math.h"

#define EPSILON 0.000001

void main()
{
    double sum = 0, term;
    int m = 1;

```

```
do
{
    term = 1.0/(m*m);
    sum = sum + term;
    m++;
}
while(fabs(term)>EPSILON);

printf("Value of series for %d terms is %lf\r\n",
      m, sum);
```

## 5.5. Performing Matrix Calculations

### 5.5.1. Problem Statement

It is required to perform matrix operations like addition, subtraction, multiplication over 4\*4 matrices.

### 5.5.2. Understand the Solution

Flowing figure summarize how matrix calculations are performed

$$\begin{array}{c}
 \text{A} \\
 \begin{array}{|c|c|c|c|} \hline 1 & 2 & 4 & 6 \\ \hline 5 & 3 & 2 & 4 \\ \hline 9 & 2 & 3 & 8 \\ \hline 2 & 1 & 6 & 9 \\ \hline \end{array}
 \end{array}
 +
 \begin{array}{c}
 \text{B} \\
 \begin{array}{|c|c|c|c|} \hline 5 & 3 & 3 & 1 \\ \hline 6 & 1 & 5 & 6 \\ \hline 7 & 3 & 3 & 5 \\ \hline 1 & 2 & 6 & 6 \\ \hline \end{array}
 \end{array}
 =
 \begin{array}{c}
 \text{C} \\
 \begin{array}{|c|c|c|c|} \hline 6 & 5 & 7 & 7 \\ \hline 11 & 4 & 7 & 10 \\ \hline 16 & 5 & 6 & 13 \\ \hline 3 & 3 & 12 & 15 \\ \hline \end{array}
 \end{array}$$

$\boxed{3} + \boxed{1} = \boxed{4} \rightarrow C[i][j] = A[i][j] + B[i][j]$

$$\begin{array}{c}
 \text{A} \\
 \begin{array}{|c|c|c|c|} \hline 1 & 2 & 4 & 6 \\ \hline 5 & 3 & 2 & 4 \\ \hline 9 & 2 & 3 & 8 \\ \hline 2 & 1 & 6 & 9 \\ \hline \end{array}
 \end{array}
 -
 \begin{array}{c}
 \text{B} \\
 \begin{array}{|c|c|c|c|} \hline 5 & 3 & 3 & 1 \\ \hline 6 & 1 & 5 & 6 \\ \hline 7 & 3 & 3 & 5 \\ \hline 1 & 2 & 6 & 6 \\ \hline \end{array}
 \end{array}
 =
 \begin{array}{c}
 \text{D} \\
 \begin{array}{|c|c|c|c|} \hline -4 & -1 & 1 & 5 \\ \hline -1 & 2 & -3 & -2 \\ \hline 2 & -1 & 0 & 3 \\ \hline 1 & -1 & 0 & 3 \\ \hline \end{array}
 \end{array}$$

$\boxed{5} - \boxed{6} = \boxed{-1} \rightarrow D[i][j] = A[i][j] - B[i][j]$

$$\begin{array}{c}
 \text{A} \\
 \begin{array}{|c|c|c|c|} \hline 1 & 2 & 4 & 6 \\ \hline 5 & 3 & 2 & 4 \\ \hline 9 & 2 & 3 & 8 \\ \hline 2 & 1 & 6 & 9 \\ \hline \end{array}
 \end{array}
 *
 \begin{array}{c}
 \text{B} \\
 \begin{array}{|c|c|c|c|} \hline 5 & 3 & 3 & 1 \\ \hline 6 & 1 & 5 & 6 \\ \hline 7 & 3 & 3 & 5 \\ \hline 1 & 2 & 6 & 6 \\ \hline \end{array}
 \end{array}
 =
 \begin{array}{c}
 \text{E} \\
 \begin{array}{|c|c|c|c|} \hline 51 & 29 & 61 & 69 \\ \hline 61 & 32 & 60 & 57 \\ \hline 86 & 54 & 94 & 84 \\ \hline 67 & 43 & 83 & 92 \\ \hline \end{array}
 \end{array}$$

$\boxed{9} * \boxed{3} + \boxed{2} * \boxed{5} + \boxed{3} * \boxed{3} + \boxed{8} * \boxed{6} = \boxed{94}$

$\rightarrow E[i][j] = A[i][0]*B[0][j] + A[i][1]*B[1][j] + A[i][2]*B[2][j]$

### 5.5.3. Write the Matrix Program

```

#include "stdio.h"
#include "math.h"

#define EPSILON 0.000001

void add4by4Matrices(double A[][4], double B[][4],
                      double R[][4])
{
    int i, j;
    for(i=0;i<4;i++)
        for(j=0;j<4;j++)
            R[i][j] = A[i][j] + B[i][j];
}

void subtract4by4Matrices(double A[][4], double B[][4],
                           double R[][4])
{
    int i, j;
    for(i=0;i<4;i++)
        for(j=0;j<4;j++)
            R[i][j] = A[i][j] - B[i][j];
}

void multibly4by4Matrices(double A[][4], double B[][4],
                           double R[][4])
{
    int i, j, k;
    for(i=0;i<4;i++)
        for(j=0;j<4;j++)
        {
            R[i][j] = 0;
            for(k=0;k<4;k++)
                R[i][j] += A[i][k] * B[k][j];
        }
}

void print4by4Matrix(double M[][4])
{
    int i, j;

    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
            printf("%2.2f\t", M[i][j]);
        printf("\r\n");
    }
}

```

```

}

void main()
{
    double A[4][4] = {
        {1, 2, 4, 6},
        {5, 3, 2, 4},
        {9, 2, 3, 8},
        {2, 1, 6, 9}};

    double B[4][4] = {
        {5, 3, 3, 1},
        {6, 1, 5, 6},
        {7, 3, 3, 5},
        {1, 2, 6, 6}};

    double R[4][4];

    add4by4Matrices(A, B, R);
    printf("\r\nC = A+B\r\n");
    print4by4Matrix(R);

    subtract4by4Matrices(A, B, R);
    printf("\r\nD = A-B\r\n");
    print4by4Matrix(R);

    multibly4by4Matrices(A, B, R);
    printf("\r\nC = A*B\r\n");
    print4by4Matrix(R);
}

```

The screenshot shows a command-line interface window titled 'cmd C:\Windows\system32\cmd.exe'. The window displays the results of a C program that performs three matrix operations: addition, subtraction, and multiplication of two 4x4 matrices, A and B.

The output is as follows:

```

C = A+B
6.00 5.00 7.00 7.00
11.00 4.00 7.00 10.00
16.00 5.00 6.00 13.00
3.00 3.00 12.00 15.00

D = A-B
-4.00 -1.00 1.00 5.00
-1.00 2.00 -3.00 -2.00
2.00 -1.00 0.00 3.00
1.00 -1.00 0.00 3.00

C = A*B
51.00 29.00 61.00 69.00
61.00 32.00 60.00 57.00
86.00 54.00 94.00 84.00
67.00 43.00 83.00 92.00

Press any key to continue . . .

```

## 5.6. Exercise

Q1.

Write a program to evaluate following mathematical series:

$$f(x, n) = \sum_{k=0}^n \frac{x^{2k+1}}{k!}$$

$$f(x) = \sum_{i=1}^{\infty} \frac{2i+x}{x^i} \quad \text{where } x > 1$$

Q2.

Write a function that arranges any given string array in ascending order. The function should have following prototype.

```
void arrange(char text[][][100], int n)

Usage Example:
char names[5][100] = {"Mostafa", "Ahmed", "Nady", "Hady",
"Monia"};
arrange(names, 5);
```

Q3.

Write a function that evaluates the determinant of a given 3x3 matrix. The function should have following prototype.

```
double evaluate3by3Matrix(double M[][3])

Usage Example:
double m[3][3] = {{3,2,9},{3,8,7},{9,3,2}};
printf("Determinant of m is %lf", evaluate3by3Matrix(m));
```

Q4.

Write a program to evaluate following mathematical series:

$$f(x, n) = \sum_{n=1}^{\infty} \frac{(3n)! + 4^{n+1}}{(3n+1)!}$$

$$f(x) = \sum_{i=1}^{\infty} \frac{2i+x}{x^i} \quad \text{where } x > 1$$

Q5.

Write a program to evaluate following infinite series:

$$f(x, n) = \sum_{n=1}^{\infty} \frac{(3n)! + 4^{n+1}}{(3n+1)!}$$

Q6.

Write a program to evaluate following infinite series:

$$\begin{aligned} f(x, p) = 1 + px + \frac{p(p-1)}{2!}x^2 + \frac{p(p-1)(p-2)}{3!}x^3 + \dots \\ + \frac{p(p-1)\dots(p-(m-1))}{n!}x^m + \dots \end{aligned}$$

## Chapter 6: Structures

## 6.1. Introduction

Consider the following problem; it is required to get the employees information (name, birth date, salary) then print it in a table. The printed information should be arranged by the employees' ages, older employee must be mentioned first.

### Example 1: Employee Sort without Structures

It appears that 5 arrays are required to store (names, date years, date months, date days and salaries). Each entry in all arrays represents the information of one employee. To sort the records based on the date a normal sort must be applied, however the three date values must be used. The sort algorithm must exchange all employees' data when sorting condition is satisfied.

```
#include "stdio.h"
#include "string.h"
#include "conio.h"

void main()
{
    char names[100][50], tempName[50];
    int birthDateYears[100], tempBirthDateYear;
    int birthDateMonths[100], tempBirthDateMonth;
    int birthDateDays[100], tempBirthDateDay;
    int salaries[100], tempSalary;
    int count = 0, i, j;
    char firstName[50], secondName[50];

    do
    {
        printf("Enter Employee First Name:");
        scanf("%s", firstName);

        printf("Enter Employee Second Name:");
        scanf("%s", secondName);
        strcpy(names[count], firstName);
        strcat(names[count], " ");
        strcat(names[count], secondName);

        printf("Enter Employee Birth Date (day/month/year)
                           example(23/3/1970):");
        scanf("%d/%d/%d", &birthDateYears[count],
              &birthDateMonths[count], &birthDateDays[count]);

        printf("Enter Employee Salary:");
        scanf("%d", &salaries[count]);

        count++; if(count==100)break;

        printf("Do you want to add more,
               press 'y' to continue?\r\n");
    }
    while(getch()=='y');
}
```

```

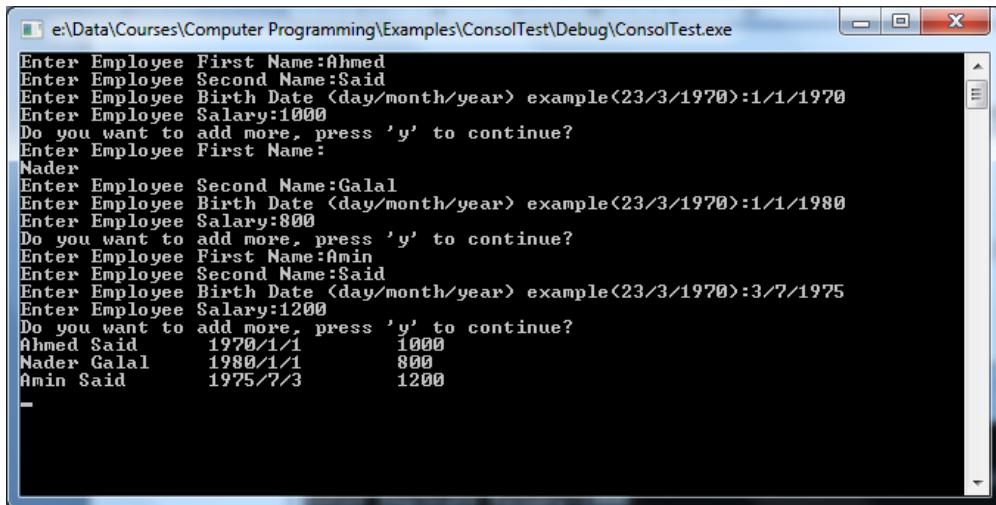
for(i=0;i<count-1;i++)
{
    for(j=i+1;j<count;j++)
    {
        if( birthDateYears[i]>birthDateYears[j] ||
            (birthDateYears[i]==birthDateYears[j] &&
            birthDateMonths[i]>birthDateMonths[j])
            ||
            (birthDateYears[i]==birthDateYears[j] &&
            birthDateMonths[i]==birthDateMonths[j]
            && birthDateDays[i]>birthDateDays[j]))
        {
            strcpy(tempName, names[i]);
            tempBirthDateYear = birthDateYears[i];
            tempBirthDateMonth = birthDateMonths[i];
            tempBirthDateDay = birthDateDays[i];
            tempSalary = salaries[i];

            strcpy(names[i], names[j]);
            birthDateYears[i] = birthDateYears[j];
            birthDateMonths[i] = birthDateMonths[j];
            birthDateDays[i] = birthDateDays[j];
            salaries[i] = salaries[j];

            strcpy(names[j], tempName);
            birthDateYears[j] = tempBirthDateYear;
            birthDateMonths[j] = tempBirthDateMonth;
            birthDateDays[j] = tempBirthDateDay;
            salaries[j] = tempSalary;
        }
    }
}

for(i=0;i<count;i++)
{
    printf("%s\t%d/%d/%d\t%d\r\n", names[i],
           birthDateDays[i], birthDateMonths[i],
           birthDateYears[i], salaries[i]);
}
}

```



```
e:\Data\Courses\Computer Programming\Examples\ConsolTest\Debug\ConsolTest.exe
Enter Employee First Name:Ahmed
Enter Employee Second Name:Said
Enter Employee Birth Date <day/month/year> example<23/3/1970>:1/1/1970
Enter Employee Salary:1000
Do you want to add more, press 'y' to continue?
Enter Employee First Name:
Nader
Enter Employee Second Name:Galal
Enter Employee Birth Date <day/month/year> example<23/3/1970>:1/1/1980
Enter Employee Salary:800
Do you want to add more, press 'y' to continue?
Enter Employee First Name:Amin
Enter Employee Second Name:Said
Enter Employee Birth Date <day/month/year> example<23/3/1970>:3/7/1975
Enter Employee Salary:1200
Do you want to add more, press 'y' to continue?
Ahmed Said      1970/1/1      1000
Nader Galal    1980/1/1      800
Amin Said      1975/7/3      1200
-
```

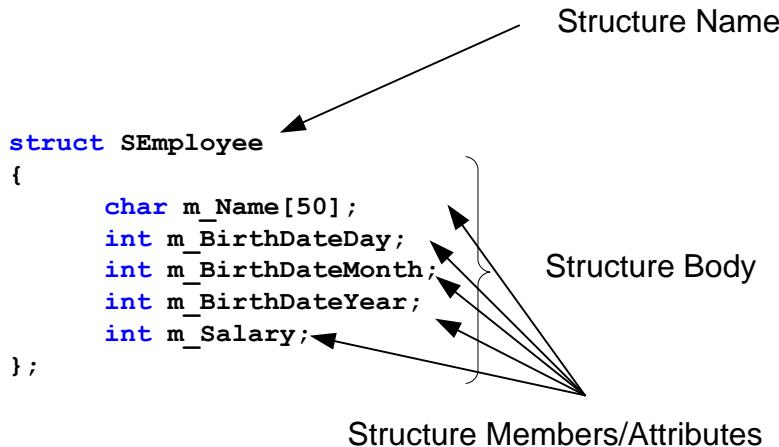
It is clear that code size is huge with respect to the problem complexity. Also the storage of the single employee information in different arrays complicates many operations like entering, sorting and printing. The situation becomes very complicated if it is required to add more attributes for the employee like experience fields, medical insurance number, driving license id...its.

**Structures** are used to collect related variables in one place. In this example all employee information can gathered in one place. This will simplify any further operation over the collected data, because all information items are treated as a single entity, called a **Structure**.

Following format is used to define an employee structure, which is used to collect all employee information in one place.

```
struct SEmployee
{
    char name[50];
    int birthDateDay;
    int birthDateMonth;
    int birthDateYear;
    int Salary;
};
```

**struct** keyword is used to inform that all data types between the next braces {...} are members of this new data type. You can define as many data types inside the structure.



**Structure** is a complex data type consisting of a set of members/attributes. Each attribute can have any data type. It can be **int**, **float**, **short**...its. It can be single value or array. It can be const or variable. Also, structures may hold another structure (nested structure).

Programmers are free to define as many structures as the program required. Each structure must be labeled with a unique label called **Structure Name**. **Structure Name** can be used as a new data type, it can be used to define variables, it can be used to define arrays and it can be passed to a functions.

## 6.2. Defining and Using Structure Variables

Each structure with different name is a new data type. Following example shows how to define a variable from the structure then write and read data from it.

### Example 2: Define and Use Structure Variable

```

#include "stdio.h"
#include "string.h"
#include "conio.h"

struct SEmployee
{
    char m_Name[50];
    int m_BirthDateDay;
    int m_BirthDateMonth;
    int m_BirthDateYear;
    int m_Salary;
};

void main()
{
}

```

```

struct SEmployee X;

strcpy(X.m_Name, "Ahmed Said");
X.m_BirthDateYear = 1980;
X.m_BirthDateMonth = 12;
X.m_BirthDateDay = 22;
X.m_Salary = 50;

printf("X contains(%s, %d/%d/%d, %d)\r\n", X.m_Name,
        X.m_BirthDateDay, X.m_BirthDateMonth,
        X.m_BirthDateYear, X.m_Salary);
}

```

In above example a variable **X** is defined from the data type (**SEmployee**). (**X**) variable contains the following members (**m\_Name**, **m\_BirthDateYear**, **m\_BirthDateMonth**, **m\_BirthDateDay**, **m\_Salary**). In this example a values is assigned to each member of (**X**) variable, then all members' values is printed.

Structure consists of a set of members; each member can be treated separately. It is applicable to read or write data to each member separately. However because the member is inside the structure, it is required to specify structure variable name before the member name.

(**X.m\_Salary**) means that you need to work with (**m\_Salary**) member inside the variable (**X**) of the type (**SEmployee**). Programmer is free to assign values to structure members or read it later.

It is applicable in Structure to deal with member separately or work with the whole structure at once. Following example show how to copy structures' variables to other structures' variables of the same type.

### Example 3: Copying Structure Variable Contents to another Variable

```

#include "stdio.h"
#include "string.h"
#include "conio.h"

struct SEmployee
{
    char m_Name[50];
    int m_BirthDateDay;
    int m_BirthDateMonth;
    int m_BirthDateYear;
    int m_Salary;
};

void main()
{

```

```

struct SEmployee X, Y, Z;

strcpy(X.m_Name, "Ahmed Said");
X.m_BirthDateDay = 22;
X.m_BirthDateMonth = 12;
X.m_BirthDateYear = 1980;
X.m_Salary = 50;

printf("X contains(%s, %d/%d/%d, %d)\r\n", X.m_Name,
       X.m_BirthDateDay, X.m_BirthDateMonth,
       X.m_BirthDateYear, X.m_Salary);

strcpy(Y.m_Name, X.m_Name);
Y.m_BirthDateDay = X.m_BirthDateDay;
Y.m_BirthDateMonth = X.m_BirthDateMonth;
Y.m_BirthDateYear = X.m_BirthDateYear;
Y.m_Salary = X.m_Salary;

printf("Y contains(%s, %d/%d/%d, %d)\r\n", Y.m_Name,
       Y.m_BirthDateDay, Y.m_BirthDateMonth,
       Y.m_BirthDateYear, Y.m_Salary);

Z = X;

printf("Z contains(%s, %d/%d/%d, %d)\r\n", Z.m_Name,
       Z.m_BirthDateDay, Z.m_BirthDateMonth,
       Z.m_BirthDateYear, Z.m_Salary);
}

```

In above example X variable is assigned to Y by copying each member separately. Alternatively, X variable is assigned to Z by direct copying using equal operator. It is clear that second method is easier and reduces the code size.

#### **Example 4: Employee Sort with Structures**

Following program uses structures to implement the employees sort problem.

```

#include "stdio.h"
#include "string.h"
#include "conio.h"

struct SEmployee
{
    char m_Name[50];
    int m_BirthDateDay;
    int m_BirthDateMonth;
    int m_BirthDateYear;
    int m_Salary;
};

```

```

void main()
{
    struct SEmployee employees[100], tempEmployee;
    int count = 0, i, j;
    char firstName[50], secondName[50];

    do
    {
        printf("Enter Employee First Name:");
        scanf("%s", firstName);

        printf("Enter Employee Second Name:");
        scanf("%s", secondName);
        strcpy(employees[count].m_Name, firstName);
        strcat(employees[count].m_Name, " ");
        strcat(employees[count].m_Name, secondName);

        printf("Enter Employee Birth Date (day/month/year)
                example(23/3/1970):");
        scanf("%d/%d/%d",
               &employees[count].m_BirthDateDay,
               &employees[count].m_BirthDateMonth,
               &employees[count].m_BirthDateYear);

        printf("Enter Employee Salary:");
        scanf("%d", &employees[count].m_Salary);

        count++; if(count==100)break;

        printf("Do you want to add more, press 'y' to
                continue?\r\n");
    }
    while(getch()=='y');

    for(i=0;i<count-1;i++)
    {
        for(j=i+1;j<count;j++)
        {
            if(
employees[i].m_BirthDateYear>employees[i].m_BirthDateYear ||
(employees[i].m_BirthDateYear==employees[j].m_BirthDateYear &&
employees[i].m_BirthDateMonth>employees[j].m_BirthDateMonth) ||
(employees[i].m_BirthDateYear==employees[j].m_BirthDateYear &&
employees[i].m_BirthDateMonth==employees[j].m_BirthDateMonth
&& employees[i].m_BirthDateDay>employees[j].m_BirthDateDay))
            {
                tempEmployee = employees[i];
                employees[i] = employees[j];
                employees[j] = tempEmployee;
            }
        }
    }
}

```

```

    }

    for(i=0;i<count;i++)
    {
        printf("%s\t%d/%d/%d\t%d\r\n",
            employees[i].m_Name, employees[i].m_BirthDateDay,
            employees[i].m_BirthDateMonth,
            employees[i].m_BirthDateYear,
            employees[i].m_Salary);
    }
}

```

The code (**SEmployee employees[100]**) means that defining an array of 100 variable of type (**struct SEmployee**).

It appears that the structure decrease the code size and simplify the exchange operation inside the sorting algorithm.

Employees Exchange without Structures	Employees Exchange with Structures
<pre> tempSerial = serials[i]; strcpy(tempName, names[i]); tempBirthDateYear = birthDateYears[i]; tempBirthDateMonth = birthDateMonths[i]; tempBirthDateDay = birthDateDays[i]; tempSalary = salaries[i];  serials[i] = serials[j]; strcpy(names[i], names[j]); birthDateYears[i] = birthDateYears[j]; birthDateMonths[i] = birthDateMonths[j]; birthDateDays[i] = birthDateDays[j]; salaries[i] = salaries[j];  serials[j] = tempSerial; strcpy(names[j], tempName); birthDateYears[j] = tempBirthDateYear; birthDateMonths[j] = tempBirthDateMonth; birthDateDays[j] = tempBirthDateDay; salaries[j] = tempSalary; </pre>	<pre> tempEmployee = employees[i];  employees[i] = employees[j];  employees[j] = tempEmployee; </pre>

The operation (**employees[i] = employees[j]**) means copy all contents of the employee entry j to employee entry i. There is no need to copy each employee attribute separately.

### 6.3. Initializing Structure Variables

Following code section shows how to initialize structure variable:

```
void main()
{
    struct SEmployee X = {"Ahmed Said", 22, 12, 1980, 500};
    printf("X contains(%s, %d/%d/%d, %d)\r\n", X.m_Name,
           X.m_BirthDateDay, X.m_BirthDateMonth,
           X.m_BirthDateYear, X.m_Salary);
}
```

The values { "Ahmed Said", 22, 12, 1980, 500 } are used to initialize members respectively.

### 6.4. Nested Structure Definition

As mentioned before, it is applicable to define structure members with any data type, even other structures type.

In Employee structure it appears that the members (**BirthDateDay**, **BirthDateMonth**, **BirthDateYear**) are related, what if, it is required to add graduation date data. The structure definition will be:

```
struct SEmployee
{
    char m_Name[50];
    int m_BirthDateDay;
    int m_BirthDateMonth;
    int m_BirthDateYear;
    int m_GraduationDateDay;
    int m_GraduationDateMonth;
    int m_GraduationDateYear;
    int m_Salary;
};
```

It appears that the readability of structure members decreases, because there are six members representing two actual data values (Birth Date, Graduation Date). Simply we can define a new structure called **SDate**. **SDate** collects all single date attributes as shown below.

```
struct SDate
{
    int m_Day;
    int m_Month;
    int m_Year;
};
```

```
struct SEmployee
{
    char m_Name[50];
    struct SDate m_BirthDate;
    struct SDate m_GraduationDate;
    int m_Salary;
};
```

Now the **SEmployee** structure becomes more readable. Following code shows how to initialize and print this structure value.

```
SEmployee X = {"Ahmed Said", {22, 12, 1990},
                {2, 7, 1970}, 5000};
printf("X contains (%s, %d/%d/%d, %d/%d/%d, %d) \r\n",
       X.m_Name, X.m_BirthDate.m_Day,
       X.m_BirthDate.m_Month, X.m_BirthDate.m_Year,
       X.m_GraduationDate.m_Day,
       X.m_GraduationDate.m_Month,
       X.m_GraduationDate.m_Year, X.m_Salary);
```

The code (**X.m\_BirthDate.m\_Day**) means reading the **m\_Day** member of **m\_BirthDate** of **X** variable.

The employee sort program can be re-written as shown in the following example:

#### Example 5: Employee Sort with Nested Structures

```
#include "stdio.h"
#include "string.h"
#include "conio.h"

struct SDate
{
    int m_Day;
    int m_Month;
    int m_Year;
};

struct SEmployee
{
    char m_Name[50];
    struct SDate m_GraduationDate;
    struct SDate m_BirthDate;
    int m_Salary;
};

void main()
```

```

{
    struct SEmployee employees[100], tempEmployee;
    int count = 0, i, j;
    char firstName[50], secondName[50];

    do
    {
        printf("Enter Employee First Name:");
        scanf("%s", firstName);

        printf("Enter Employee Second Name:");
        scanf("%s", secondName);
        strcpy(employees[count].m_Name, firstName);
        strcat(employees[count].m_Name, " ");
        strcat(employees[count].m_Name, secondName);

        printf("Enter Employee Birth Date (day/month/year)
                           example(23/3/1970):");
        scanf("%d/%d/%d",
              &employees[count].m_BirthDate.m_Day,
              &employees[count].m_BirthDate.m_Month,
              &employees[count].m_BirthDate.m_Year);

        printf("Enter Employee Graduation Date
                           (day/month/year) example(23/3/1970):");
        scanf("%d/%d/%d",
              &employees[count].m_GraduationDate.m_Day,
              &employees[count].m_GraduationDate.m_Month,
              &employees[count].m_GraduationDate.m_Year);

        printf("Enter Employee Salary:");
        scanf("%d", &employees[count].m_Salary);

        count++; if(count==100)break;

        printf("Do you want to add more, press 'y'
                           to continue?\r\n");
    }
    while(getch()=='y');

    for(i=0;i<count-1;i++)
    {
        for(j=i+1;j<count;j++)
        {
            if(
employees[i].m_BirthDate.m_Year>employees[i].m_BirthDate.m_Year ||
(employees[i].m_BirthDate.m_Year==employees[j].m_BirthDate.m_Year &
employees[i].m_BirthDate.m_Month>employees[j].m_BirthDate.m_Month) ||
(employees[i].m_BirthDate.m_Year==employees[j].m_BirthDate.m_Year &
employees[i].m_BirthDate.m_Month==employees[j].m_BirthDate.m_Month &&
employees[i].m_BirthDate.m_Day>employees[j].m_BirthDate.m_Day))
            {
                tempEmployee = employees[i];
                employees[i] = employees[j];
                employees[j] = tempEmployee;
            }
        }
    }
}

```

```

    }

    for(i=0;i<count;i++)
    {
        printf("%s\t%d/%d/%d\t%d/%d/%d\t%d\r\n",
               employees[i].m_Name,
               employees[i].m_BirthDate.m_Day,
               employees[i].m_BirthDate.m_Month,
               employees[i].m_BirthDate.m_Year,
               employees[i].m_GraduationDate.m_Day,
               employees[i].m_GraduationDate.m_Month,
               employees[i].m_GraduationDate.m_Year,
               employees[i].m_Salary);
    }
}

```

## 6.5. Using Structures with Functions

Structures are used with functions like any normal variable, it can be passed as a single value or array, and also it can be returned as a result of the function. Following sample code shows how to define a function that reads and prints Date and Employee structures.

### Example 6: Read and Print Employee and Date Values using Functions

```

#include "stdio.h"
#include "string.h"
#include "conio.h"

struct SDate
{
    int m_Day;
    int m_Month;
    int m_Year;
};

struct SEmployee
{
    char m_Name[50];
    struct SDate m_GraduationDate;
    struct SDate m_BirthDate;
    int m_Salary;
};

struct SDate ReadDate(char dateName[])
{
    struct SDate date;
    printf("Enter %s (day/month/year) example(23/3/1970) :",
           dateName);

```

```

        scanf("%d/%d/%d", &date.m_Day, &date.m_Month,
                           &date.m_Year);
    return date;
}

struct SEmployee ReadEmployee()
{
    struct SEmployee employee;
    char firstName[50], secondName[50];

    printf("Enter Employee First Name:");
    scanf("%s", firstName);

    printf("Enter Employee Second Name:");
    scanf("%s", secondName);

    strcpy(employee.m_Name, firstName);
    strcat(employee.m_Name, " ");
    strcat(employee.m_Name, secondName);

    employee.m_BirthDate = ReadDate("Employee Birth Date");
    employee.m_GraduationDate =
        ReadDate("Employee Graduation Date");

    printf("Enter Employee Salary:");
    scanf("%d", &employee.m_Salary);

    return employee;
}

void PrintEmployee(struct SEmployee employee)
{
    printf("%s\t%d/%d/%d\t%d/%d\t%d\r\n",
           employee.m_Name,
           employee.m_BirthDate.m_Year,
           employee.m_BirthDate.m_Month,
           employee.m_BirthDate.m_Day,
           employee.m_GraduationDate.m_Year,
           employee.m_GraduationDate.m_Month,
           employee.m_GraduationDate.m_Day,
           employee.m_Salary);
}

void main()
{
    struct SEmployee X = ReadEmployee();
    PrintEmployee(X);
}

```

**ReadDate** function read date value from user, then store it in **SDate** structure then return its value. **ReadEmployee** function read employee value from user, then store it in **SEmployee** structure then return its value. **PrintEmployee** function takes employee value and prints it.

### Example 7: Comparing Two Date Values using Functions

Following example show how to write and use a function that compare two date values.

```
int CompareDates(struct SDate A, struct SDate B)
{
    if( A.m_Year == B.m_Year &&
        A.m_Month == B.m_Month &&
        A.m_Day == B.m_Day)
        return 0;

    if( A.m_Year > B.m_Year ||
        (A.m_Year == B.m_Year && A.m_Month > B.m_Month) ||
        (A.m_Year == B.m_Year && A.m_Month == B.m_Month &&
         A.m_Day > B.m_Day))
        return 1;

    return -1;
}

void main()
{
    struct SDate X = ReadDate("X Date");
    struct SDate Y = ReadDate("Y Date");
    switch(CompareDates(X, Y))
    {
        case -1:printf("X date < Y date\r\n");break;
        case 0:printf("X date = Y date\r\n");break;
        case 1:printf("X date > Y date\r\n");break;
    }
}
```

Using above functions (**ReadEmployee**, **PrintEmployee**, **ReadDate**, **CompareDates**) we can simplify the employee sort program as shown below:

### Example 8: Simplified Employee Sort Program using Functions

```
void main()
{
    struct SEmployee employees[100], tempEmployee;
    int count = 0, i, j;
```

```

do
{
    employees[count] = ReadEmployee();
    count++; if(count==100)break;
    printf("Do you want to add more,
           press 'y' to continue?\r\n");
}
while(getch()=='y');

for(i=0;i<count-1;i++)
{
    for(j=i+1;j<count;j++)
    {

        if(CompareDates(employees[i].m_BirthDate,
                        employees[j].m_BirthDate)>0)
        {
            tempEmployee = employees[i];
            employees[i] = employees[j];
            employees[j] = tempEmployee;
        }
    }
}

for(i=0;i<count;i++)
    PrintEmployee(employees[i]);
}

```

## 6.6. Supporting Complex Numbers

Structure can be used to define complex number, because it consists of two parts, real and imaginary. The two parts of complex number must be together and the whole complex number is treated as one number. The complex structure should be defined as follows:

```

struct SComplex
{
    double m_R; //Real Part
    double m_I; //Imaginary Part
};

```

### Example 9: Adding Two Complex Numbers

Following example defines three complex values X, Y and Z, then takes X and Y values from users, add them and place the result in Z, and finally prints Z value.

```
#include "stdio.h"

struct SComplex
{
    double m_R; //Real Part
    double m_I; //Imaginary Part
};

void main()
{
    struct SComplex X, Y, Z;

    printf("Enter X Complex Value (Ex: 5, -3):");
    scanf("%lf, %lf", &X.m_R, &X.m_I);
    printf("Enter Y Complex Value (Ex: 5, -3):");
    scanf("%lf, %lf", &Y.m_R, &Y.m_I);

    Z.m_R = X.m_R + Y.m_R;
    Z.m_I = X.m_I + Y.m_I;

    printf("Z = (%lf) + j (%lf)", Z.m_R, Z.m_I);
}
```

### Example 10: Adding Two Complex Numbers with Functions

Above example is rewritten using the functions.

```
#include "stdio.h"

struct SComplex
{
    double m_R; //Real Part
    double m_I; //Imaginary Part
};

struct SComplex ReadComplex(char name[])
{
    struct SComplex C;
    printf("Enter %s Complex Value (Ex: 5, -3):", name);
    scanf("%lf, %lf", &C.m_R, &C.m_I);

    return C;
}

struct SComplex AddComplex(struct SComplex A, struct SComplex B)
```

```

{
    struct SComplex C;
    C.m_R = A.m_R + B.m_R;
    C.m_I = A.m_I + B.m_I;
    return C;
}

void PrintComplex(char name[], struct SComplex C)
{
    printf("%s = (%lf) + j (%lf)\r\n", name, C.m_R, C.m_I);
}

void main()
{
    struct SComplex X, Y, Z;

    X = ReadComplex("X");
    Y = ReadComplex("Y");
    Z = AddComplex(X, Y);
    PrintComplex("Z", Z);
}

```

## 6.7. enum

**enum** is a way to define a linguistic name instead of numeric values. For example week day can be defined as integer value from 0 to 6; in the other hand enum allows the definition of week days with their real names.

```
enum Weekday {SATURDAY, SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
THURSDAY, FRIDAY};
```

Above definition, assign default numerical value starting from 0 to each value respectively. It is applicable to assign another numeric value as shown below:

```
enum Weekday {SATURDAY=1, SUNDAY=2, MONDAY=3, TUESDAY=4,
WEDNESDAY=5, THURSDAY=6, FRIDAY=7};
```

### Example 11: Personal Data

```
#include "stdio.h"

enum Gender{MALE, FEMALE};

struct SPerson
{
    char m_Name[100];
    enum Gender m_Gender;
```

```

};

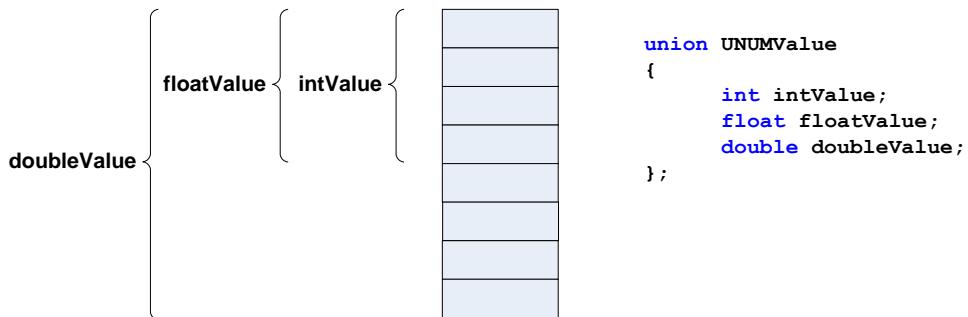
void main()
{
    int i;
    struct SPerson persons[] = { {"Ahmed Ali", MALE},
                                 {"Mona Mohamed", FEMALE}};

    for(i=0;i<sizeof(persons)/sizeof(struct SPerson);i++)
    {
        printf("%s - %s\n", persons[i].m_Name,
               (persons[i].m_Gender==MALE)?"Male":"Female");
    }
}

```

## 6.8. Unions

Union is a special data type that supports different overlapped data types. For example following union can hold integer or float or double values, the developer is responsible on using one of those types.



Union size equals to the size of the biggest data type. For **UNUMValue** union the size equals to the double value size (8 bytes).

Normally unions are used to give the programmer a chance to store different data types for certain value.

### Example 12: Using Union

Following example illustrates how to combine the three data types (**int**, **float**, **double**) in one overlapped data type. Programmer is responsible on supplying and retrieving the data correctly. If integer data is supplied, integer data must be retrieved.

```

#include "stdio.h"

enum NUMType{INT, FLOAT, DOUBLE};

```

```

union UNUMValue
{
    int u_intValue;
    float u_floatValue;
    double u_doubleValue;
};

union UNUMValue Add(union UNUMValue value1, union UNUMValue
value2, enum NUMType type)
{
    union UNUMValue result;

    switch(type)
    {
        case INT: result.u_intValue = value1.u_intValue +
                    value1.u_intValue; break;

        case FLOAT: result.u_floatValue = value1.u_floatValue +
                     value1.u_floatValue; break;

        case DOUBLE: result.u_doubleValue = value1.u_doubleValue +
                     + value1.u_doubleValue; break;
    }

    return result;
}

void main()
{
    union UNUMValue V1, V2, R;

    printf("the size of UNUMValue is %d bytes\n\n",
           sizeof(union UNUMValue));

    V1.u_intValue = 9898;
    V2.u_intValue = 8776;
    R = Add(V1, V2, INT);
    printf("int: %d + %d = %d\n",
           V1.u_intValue, V2.u_intValue, R.u_intValue);

    V1.u_floatValue = 86.82;
    V2.u_floatValue = 83.11;
    R = Add(V1, V2, FLOAT);
    printf("float: %f + %f = %f\n",
           V1.u_floatValue, V2.u_floatValue, R.u_floatValue);

    V1.u_doubleValue = 821.8;
    V2.u_doubleValue = 988.2;
    R = Add(V1, V2, DOUBLE);
    printf("double: %lf + %lf = %lf\n",
           V1.u_doubleValue, V2.u_doubleValue, R.u_doubleValue);
}

```

```
V1.u_doubleValue, V2.u_doubleValue, R.u_doubleValue);  
}
```

## 6.8. Coding Convention

### 6.8.1. Structure, Union Naming

Structure name must begin with ‘S’ letter.

```
struct SStudent
```

Union name must begin with ‘U’ letter.

```
union UNUMValue
```

### 6.8.2. Structure, Union Members Naming

Structure members name must begin with ‘m\_’ letter.

```
struct SPerson  
{  
    char m_Name[100];  
    enum Gender m_Gender;  
};
```

Union members name must begin with ‘u\_’ letter.

```
union UNUMValue  
{  
    int u_intValue;  
    float u_floatValue;  
    double u_doubleValue;  
};
```

### 6.8.2. enum Members Naming

First: enum member’s names must use Capital Letters.

```
enum NUMType{INT, FLOAT, DOUBLE};
```

Second: enum member’s names must use ‘\_’ to separate several words.

```
enum NUMType{TYPE_INT, TYPE_FLOAT, TYPE_DOUBLE};
```

## 6.10. Exercise

### Q1.

Define student Structure with name, id, birth date, and all subjects' grades; you must specify for each subject a name, id, and a maximum degree. Write a program that take above all values then prints a certificate for the students containing:

1. ID
2. Name
3. Age
4. Subjects Degree over the Maximum Degree EX(52/70)
5. Subject Success Status (Failed, Passed, Good, V. Good, Excellent)
6. Total Degrees
7. Total Percentage
8. Total Success Status (Failed, Passed, Good, V. Good, Excellent)

### Q2.

Write a program that takes persons information (name, id, father id). The program should take persons information in any order. When the user finishes, the program should prints all persons and show fathers and sons.

You should provide two solutions, one with recursion and the other without recursion.

Example if the supplied date is:

Ahmed Mohamed, 99, 11  
 Ali Said, 18, 9  
 Mohamed Mostafa, 11, 5  
 Said Fayed, 9, 12  
 Fareed Ali, 26, 18  
 Nader Mohamed, 23, 11  
 Waleed Ali, 92, 18

The program should print the data as following

- Mohamed Mostafa (11)
  - o Ahmed Mohamed (99)
  - o Nader Mohamed (23)
- Said Fayed (9)
  - o Ali Said (18)
    - Fareed Ali (26)
    - Waleed Ali (92)

Hint: The non-recursive program should arrange all persons starting from Parents to the Childs.

**Q3.**

For the program in Q2, it is required to print the full name of all members. The program output should be:

- Mohamed Mostafa (11)
  - o Ahmed Mohamed Mostafa (99)
  - o Nader Mohamed Mostafa (23)
- Said Fayed (9)
  - o Ali Said Fayed (18)
    - Fareed Ali Said Fayed (26)
    - Waleed Ali Said Fayed (92)

**Q4.**

Modify the `PrintComplex` function to provide more readable output:

For Example the value { (4) + j (-5) } should be printed as {4-5j}.

For Example the value { (4) + j (+5) } should be printed as {4+5j}.

For Example the value { (4) + j (0) } should be printed as {4}.

For Example the value { (0) + j (-5) } should be printed as {-5j}.

**Q5.**

Using structures and functions, write a complete Complex mathematics library that provide Complex addition, subtraction, multiplication, division, inversion, amplitude evaluation and phase evaluation.

**Q6. (Report)**

Write a program that takes a number of courses from user (id, name). After entering all courses it re-ask user to supply for each course the direct prerequisite courses. After all, when the user provides a course id, the program should list all direct and indirect direct prerequisite courses.

Example:

Courses:

1. Mathematics (1)
2. Physics (2)
3. Mechanics (3)
4. Introduction to Computer (4)
5. Advanced Mathematics (5)
6. C Programming (6)
7. Circuits (7)
8. Electronics (8)
9. Computer Organization (9)

The direct course prerequisites:

1. Mathematics (1):
2. Physics (2): 1
3. Mechanics (3): 1, 2
4. Introduction to Computer (4): 1
5. Advanced Mathematics (5): 1
6. C Programming (6): 4

7. Circuits (7): 2
8. Electronics (8): 7
9. Computer Organization (9): 6, 8

The direct and indirect prerequisites for Computer Organization are:

1. C Programming (6)
2. Introduction to Computer (4)
3. Mathematics (1)
4. Electronics (8)
5. Circuits (7)
6. Physics (2)

## Chapter 7: Pointers

## 7.1. Introduction

Programming with Pointers is the heart of C language, powerful applications built by programmers who knows how to use the pointers. Pointers may lead to very fast and efficient programs if they are used correctly, or may lead a very buggy and faulty programs if they are not.

Pointer is a new type of variables used to store the address of another variable, for that reason it can be used to read or change the pointed value indirectly. Following two examples illustrates this idea:

### Example 1: Printing Employee Data without Pointers

```

struct SDate
{
    int m_Day;
    int m_Month;
    int m_Year;
};

struct SEmployee
{
    char m_Name[50];
    struct SDate m_GraduationDate;
    struct SDate m_BirthDate;
    int m_Salary;
};

void PrintEmployee(struct SEmployee employee)
{
    printf("%s, %d/%d/%d, %d/%d/%d, %d\r\n",
    employee.m_Name,
    employee.m_GraduationDate.m_Day,
    employee.m_GraduationDate.m_Month,
    employee.m_GraduationDate.m_Year,
    employee.m_BirthDate.m_Day,
    employee.m_BirthDate.m_Month,
    employee.m_BirthDate.m_Year,
    employee.m_Salary);
}

void main()
{
    struct SEmployee X = {"Ahmed Said", {22, 12, 1990},
                           {2, 7, 1970}, 5000};
    PrintEmployee(X);
}

```

### Example 2: Printing Employee Data with Pointers

```

struct SDate
{
    int m_Day;
    int m_Month;
    int m_Year;
};

struct SEmployee
{
    char m_Name[50];
    struct SDate m_GraduationDate;
    struct SDate m_BirthDate;
    int m_Salary;
};

void PrintEmployee(struct SEmployee* pEmployee)
{
    printf("%s, %d/%d/%d, %d/%d/%d, %d\r\n",
            pEmployee->m_Name,
            pEmployee->m_GraduationDate.m_Day,
            pEmployee->m_GraduationDate.m_Month,
            pEmployee->m_GraduationDate.m_Year,
            pEmployee->m_BirthDate.m_Day,
            pEmployee->m_BirthDate.m_Month,
            pEmployee->m_BirthDate.m_Year,
            pEmployee->m_Salary);
}

void main()
{
    struct SEmployee X = {"Ahmed Said", {22, 12, 1990},
                           {2, 7, 1970}, 5000};
    PrintEmployee(&X);
}

```

In example 1&2 you will notice following changes.

Without Pointer	With Pointer
<b>void</b> PrintEmployee(SEmployee employee)	<b>void</b> PrintEmployee(SEmployee* pEmployee)
PrintEmployee(X)	PrintEmployee(&X)
employee.m_Name	pEmployee->m_Name

In the second example (**struct SEmployee\* pEmployee**) means a pointer variable that will hold the submitted address. (**&x**) means the address of **x**. (**pEmployee->**) this notation is used to access structure members in case of pointers instead of (**employee.**) in case of non-pointers.

To understand the importance of pointers:

In Example 1, the (**PrintEmployee (x)**) function takes employee value as an input, which means transmitting all **SEmployee** contents to the function. The transmitted size equals to  $(50 + 3*4 + 3*4 + 4 = 78$  byte).

In example 2, the (**PrintEmployee (&x)**) function takes only the address of the employee value as an input, which means only the address is transmitted (4 bytes).

Pointers in example 2 reduce the transmission size from 78 byte to 4 bytes. Both programs provide the same operation, however Example 2 is faster.

## 7.2. Address of Variable

Address is the location of the variable or the function in the memory. When the program is loaded all variables and functions are placed in the main memory. When the program tries to read a value from a variable, it actually goes to the variable address and brings the required value.

```
void main()
{
    int x = 5, y = 9, z;

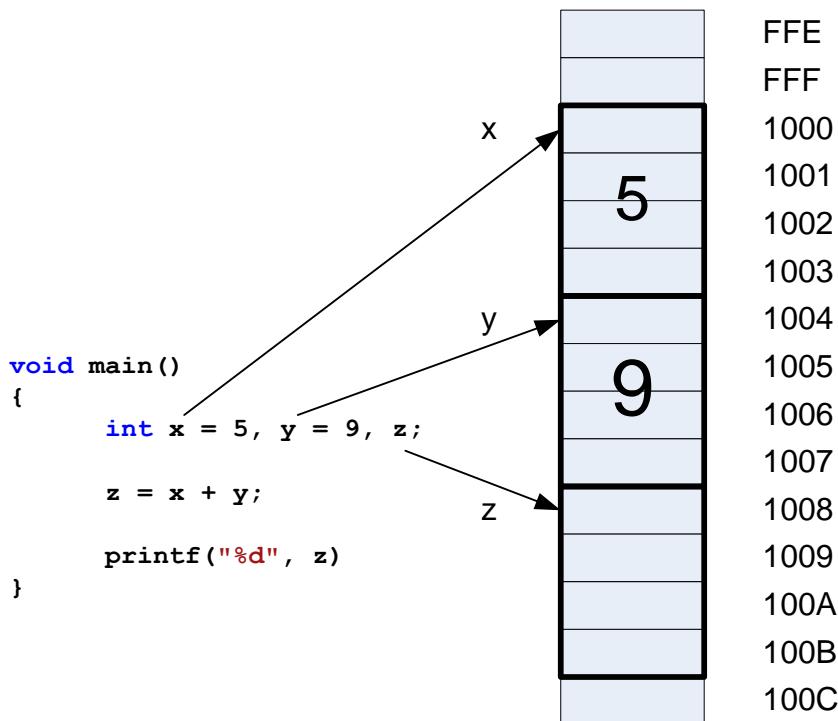
    z = x + y;

    printf("%d", z)
}
```

Above code segment means that, x, y and z are three integer values stored in the computer memory. x has a value of 5. y has a value of 9. z has a garbage value because it is not initialized.

When the processor executes the command (**z = x + y**) it performs the following steps respectively:

1. Brings the value stored at **x address**
2. Brings the value stored at **y address**
3. Add the values inside the processor
4. Place the result at **z address**



As shown in the figure the computer memory looks like a very long array of bytes starting from 0 up to the memory size. Each cell stores one byte only. In this example x, y and z are integer, each of them occupy 4 bytes from the main memory. As shown in the figure the address of x is 0x1000, the address of y is 0x1004 and the address of z is 0x1008. Know it is not mandatory to place variables in one place. The compiler can place variables in any place as shown in the following example.

### Example 3: Printing Variables Addresses and Values

Following example shows how to print the address and the value of any variable.

```

#include "stdio.h"

void main()
{
    int x = 5, y = 9, z;

    printf("x at location 0x%x contains %d\r\n", &x, x);
    printf("y at location 0x%x contains %d\r\n", &y, y);

    z = x + y;

    printf("z at location 0x%x contains %d\r\n", &z, z);
}
  
```

```
x at location 0x12ff28 contains 5
y at location 0x12ff1c contains 9
z at location 0x12ff10 contains 14
Press any key to continue . . .
```

The program output indicates that:

Variable	Address
x	0x12ff28
y	0x12ff1c
z	0x12ff10

It is clear that x, y and z are not placed beside each others. C compiler is fully responsible on selecting the suitable address of each variable.

Also, it appears that (**&x**, **&y**, **&z**) means that the address of the variable.  
 (&operator) means the “the address of”.  
 (&X) means the address of X.

Address is a normal integer value, however normally we deal with addresses in hexadecimal format.

### 7.3. Address of Array

Array is a set of arranged variables placed at one location. In C language array name is the beginning address of the whole array.

```
#include "stdio.h"

void main()
{
    int x[5] = {1, 2, 3, 4, 5};
    printf("The address of arary x is 0x%x\r\n", x);
}
```

```
The address of arary x is 0x12ff18
Press any key to continue . . .
```

In the code above, (x) is the name and the address of the array.

Know that, the notation x[3] means that the forth item of the array, alternatively another notation can be used as shown in the following example:

```
#include "stdio.h"

void main()
{
    int x[5] = {1, 2, 3, 4, 5};
    int i;
    for(i=0;i<5;i++)
    {
        printf("%d %d\r\n", x[i], *(x+i));
    }
}
```

```
1 1
2 2
3 3
4 4
5 5
Press any key to continue . . . .
```

`* (x+i)` is equivalent to `x[i]`. Actually `(x[i])` is a shortcut.

`* (x+i)` means the value pointed by the address `(x+i)`.

`(x+i)` means the address at  $i^{\text{th}}$  location after `x` address. Know that `x` is an array of integer each location in the array occupies 4 bytes in the program memory.

#### Example 4: Printing Array of Elements Addresses and Values

Following example shows how to print the address and the value of all array elements.

Method 1:

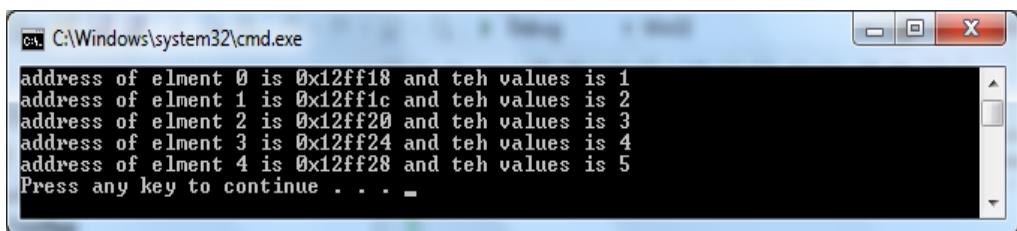
```
#include "stdio.h"

void main()
{
    int x[5] = {1, 2, 3, 4, 5};
    int i;
    for(i=0;i<5;i++)
    {
        printf("address of element %d is 0x%x and the
               values is %d\r\n", i, x+i, *(x+i));
    }
}
```

Method 2:

```
#include "stdio.h"

void main()
{
    int x[5] = {1, 2, 3, 4, 5};
    int i;
    for(i=0;i<5;i++)
    {
        printf("address of element %d is 0x%x and the
               values is %d\r\n", i, &x[i], x[i]);
    }
}
```



It is clear that each array element takes exactly 4 byte.

Element	Address	Value
0	(x+0) or &x[0] → 0x12ff18	*(x+0) or x[0] → 1
1	(x+1) or &x[1] → 0x12ff1c	*(x+1) or x[1] → 2
2	(x+2) or &x[2] → 0x12ff20	*(x+2) or x[2] → 3
3	(x+3) or &x[3] → 0x12ff24	*(x+3) or x[3] → 4
4	(x+4) or &x[4] → 0x12ff28	*(x+4) or x[4] → 5

Know that the address notation (**x+i**) is equivalent to **&x[i]**.

Also, know that the value retrieval notations **\*(x+i)** is equivalent to **x[i]**.

#### 7.4. Total Memory Used by Program Variables

This section illustrates how to calculate the total memory required by the program variables. Following program define 6 variables as shown bellow, and it is required to know the total allocated memory by the program:

```
void main()
{
    Long x, y;
    char a, b;
```

```

short m, n;
double z;
short s[25];
float x[20];

.....
}

```

Simply the expected size can be calculated by the direct summation of variables size in bytes which gives:

4 bytes (x) + 4 bytes (y) + 1 byte (a) + 1 byte (b) + 2 bytes (m) + 2 bytes (n) + 8 bytes (z) + 50 bytes (s) + 80 bytes (x) = 152 bytes

Actually the reserved size for program variables may exceed (152 bytes) value. This variation comes from a Compiler Configuration value called (Bytes Alignment) which means variables should be placed in a memory aligned to certain number.

For example in 32 Bit computers the processor reads 4 bytes from memory at once even if it is required to read 1 byte. The best Byte Alignment value is 4, which makes all variables takes at least 4 bytes or a size divisible by 4.

Another example, in 64 Bit computers the (Byte Alignment) value is pre-configured with 8.

Again, to calculate the actual memory size required by this program in 32 Bit Computer with default Byte Alignment value (4):

4 bytes (x) + 4 bytes (y) + 4 byte (a) + 4 byte (b) + 4 bytes (m) + 4 bytes (n) + 8 bytes (z) + 52 bytes (s) + 80 bytes (x) = 164 bytes

Another example, it is required to calculate the required memory to store following SPerson structure:

```

struct SPerson
{
    char m_Name[18];
    long m_ID;
    char m_Age;
    short m_Salary;
    double m_Weight;
};

```

In 32 Bit Computer with default Byte Alignment value (4):

Variable Name	Variable Size	Actual Allocated Size
<b>char</b> m_Name[18]	18	20
<b>long</b> m_ID	4	4
<b>char</b> m_Age	1	4
<b>short</b> m_Salary	2	4

<b>double m_Weight</b>	<b>8</b>	<b>8</b>
Total		40

If Byte Alignment value is configured to (8):

Variable Name	Variable Size	Actual Allocated Size
<b>char m_Name[18]</b>	<b>18</b>	<b>24</b>
<b>long m_ID</b>	<b>4</b>	<b>8</b>
<b>char m_Age</b>	<b>1</b>	<b>8</b>
<b>short m_Salary</b>	<b>2</b>	<b>8</b>
<b>double m_Weight</b>	<b>8</b>	<b>8</b>
Total		56

Even if Byte Alignment is known, some compilers have special implements for memory allocation which may lead to unexpected memory size.

You can programmatically evaluate the actual size of any variable, array, data type and structure using **sizeof()** directive, following examples show that:

#### Example 5: Calculating Structure Size using **sizeof()** directive

```
#include "stdio.h"

struct SPerson
{
    char m_Name[18];
    long m_ID;
    char m_Age;
    short m_Salary;
    double m_Weight;
};

void main()
{
    printf("the size of person structure is %d\r\n",
           sizeof(struct SPerson));
}
```

#### Example 6: Calculating Number of Array Elements using **sizeof()** directive

```
#include "stdio.h"

void main()
{
    long x[5] = {1, 2, 3, 4, 5};
    int i;
```

```

for(i=0;i<sizeof(x)/sizeof(long);i++)
{
    printf("%d\r\n", x[i]);
}

```

(**sizeof(x)/sizeof(int)**) calculate the number of array elements which is (20/4 → 5). This way is preferable instead of placing (5).

## 7.5. Pointer to Variable

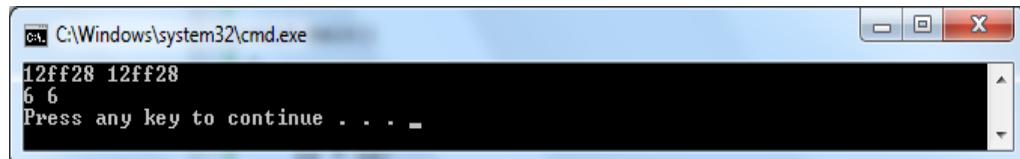
Simply pointer is a special type of variables used to store the address value. For example:

```

#include "stdio.h"

void main()
{
    int x = 6;
    int* px;
    px = &x;
    printf("%x %x\r\n", &x, px);
    printf("%d %d\r\n", x, *px);
}

```



(**int\* px**) means that px is a pointer.

(**px = &x**) means that the address of x (**&x**) is assigned in px. Now px contains the address of x, in another way (**px now points to x**).

**printf("%x %x\r\n", &x, px);** prints the address of x (**&x**) and the value of px which are the same.

**printf("%d %d\r\n", x, \*px);** prints the values inside x and the value pointed by px. (**\*px**) can be read as (the value pointed by px).

### Example 7: Using Pointer to Variable

Following program gives more examples on how to use pointers:

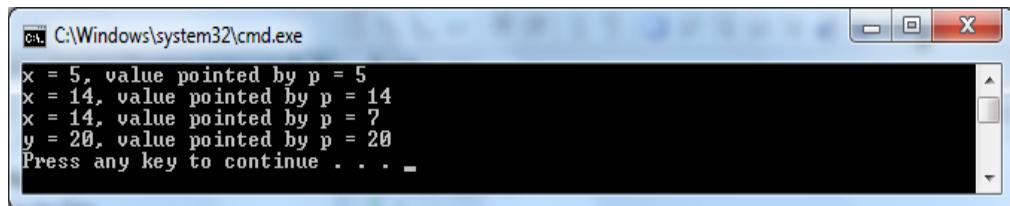
```
#include "stdio.h"

void main()
{
    int x = 5;
    int y = 7;
    int* p;

    p = &x;
    printf("x = %d, value pointed by p = %d\r\n", x, *p);
    *p = 14;
    printf("x = %d, value pointed by p = %d\r\n", x, *p);

    p = &y;
    printf("x = %d, value pointed by p = %d\r\n", x, *p);
    *p = 20;
    printf("y = %d, value pointed by p = %d\r\n", y, *p);

    p = 0;
    /*p = 15; //Wrong and the program will crash here
}
```



In above program:

(**p = &x**) means storing the address of (**x**) inside the pointer variable (**p**). Later on the print statement prints the value of (**x**) and the value pointed by (**p**) which is the same. **printf** prints (5) value for both of them.

(**\*p = 14**) means assigning (14) to the value pointed by (**p**), which of course affect the (**x**) value. Later on the print statement prints the value of (**x**) and the value pointed by (**p**) which is the same. **printf** prints (14) value for both of them.

Again, (**p = &y**) means storing the address of (**y**) inside the pointer variable (**p**). Later on the print statement prints the value of (**y**) and the value pointed by (**p**) which is the same. **printf** prints (7) value for both of them.

(**\*p = 20**) means assigning (20) to the variable pointed by (**p**), which of course affect the (**y**) variable. Later on the print statement prints the value of (**y**) and the value pointed by (**p**) which is the same. **printf** prints 20 value for both of them.

Finally, (**p = 0**) means storing (0) address inside the pointer variable (**p**). Now (**p**) points to the location (0) which is not belong to any of program variables.

If the commented statement (`*p = 15`) is executed the program should give an error then crash, this because the program tries to store the value (15) at location (0) which is not belong to any of program variables.

## 7.6. Pointer to Array

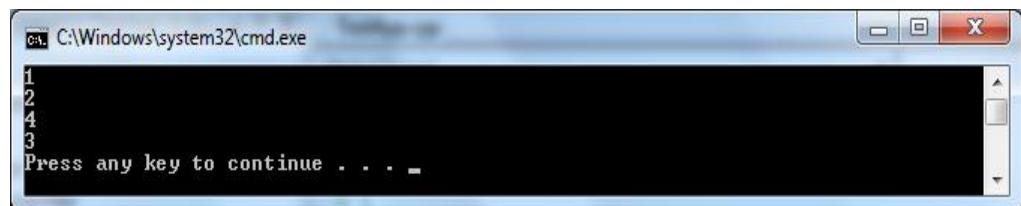
As mentioned before array name is the address of the array. Following example illustrate how to work with pointers to arrays.

### Example 8: Using Pointer to Array

```
#include "stdio.h"

void main()
{
    int x[5] = {1, 2, 3, 4, 5};
    int* p = x;
    printf("%d\r\n", *p);
    p++;
    printf("%d\r\n", *p);
    p = x + 3;
    printf("%d\r\n", *p);
    p--;
    printf("%d\r\n", *p);

    //x++; Wrong, Array Address is Fixed
}
```



`(int* p = x)` defines a pointer with name (`p`) and store the address of the array `x` inside it.

`printf("%d\r\n", *p)` prints the value pointed by the pointer (`p`) which is the first value of the (`x`) array. This line prints (1).

`(p++)` this statement moves the pointer one location forward, which means adding 4 to the pointer value so that it points to the second element in the array (`x`).

`printf("%d\r\n", *p)` prints the value pointed by the pointer (`p`) which is the second value of the (`x`) array. This line prints (2).

**(p = x + 3)** this statement stores (the address of x + 3x4) to the pointer (p), which means that p now points to the fourth element in (x) array.

**printf("%d\r\n", \*p)** prints the value pointed by the pointer (p) which is the fourth value of the (x) array. This line prints (4).

**(p--)** this statement moves the pointer one location backward, which means subtracting 4 from the pointer value so that it points to the third element in the array (x).

**printf("%d\r\n", \*p)** prints the value pointed by the pointer (p) which is the third value of the (x) array. This line prints (3).

Know that **(p+i)** or **(x+i)** do not add i number to the address, it actually add (i \* 4) value. this because (x) and (p) are of type **(int)**.

### Example 9: Average of Weights

It is required to calculate the average weight of boxes. The user should enter the boxes weight to the program. When the user finishes he must supply a zero value. This problem will be solved using three methods:

#### Method 1: Using Arrays

```
#include "stdio.h"

void main()
{
    int X[] = {43, 98, 79, 57, 34, 96};
    int sum = 0;
    int i;

    printf("Enter the values\n");
    for(i=0;i<sizeof(X)/sizeof(int);i++)
        scanf("%d\n", &X[i]);

    printf("You have entered\n");
    for(i=0;i<sizeof(X)/sizeof(int);i++)
        printf("%d\n", X[i]);

    for(i=0;i<sizeof(X)/sizeof(int);i++)
        sum += *(X+i);

    printf("sum = %d\n", sum);
}
```

**sum += \* (X+i) ;** adds the value pointed by the address (X + i) to sum.

### Method 2: Using Pointers

```
#include "stdio.h"

void main()
{
    int X[5];
    int sum = 0;
    int* pX = X;

    for(i=0;i<sizeof(X)/sizeof(int);i++)
        scanf("%d\n", pX + i);

    for(i=0;i<sizeof(X)/sizeof(int);i++)
        printf("%d\n", *pX++);

    pX = X;
    for(i=0;i<sizeof(X)/sizeof(int);i++, pX++)
        sum += *pX;

    printf("sum = %d\n", sum);
}
```

**int\* pX;** defines arary of type int and a name (**pX**).

**pX = X;** stores the address of the array (**X**) inside the pointer (**pX**).

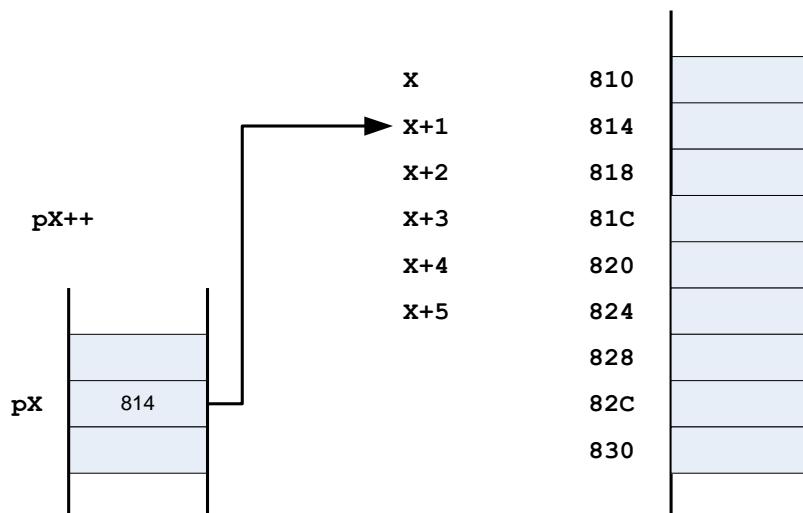
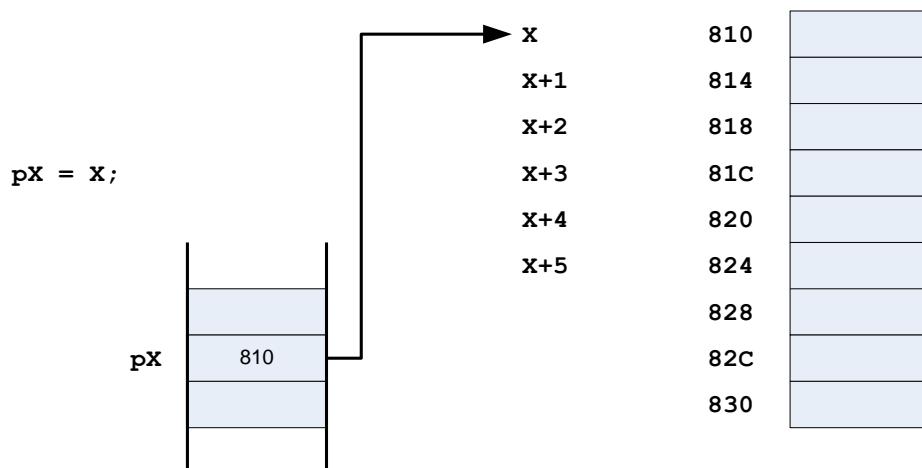
**scanf("%f", pX);** supplies the value entered by user to the location pointed by the pointer (**pX**).

**pX++** increments the pointer (**pX**) value by one element, so that it points to the next location in the original array (**X**).

**printf("%d\n", \*pX++);** prints the value pointed by (**pX**) then increments the pointer (**pX**) value by one element.

**pX = X;** again stores the address of the array (**X**) inside the pointer (**pX**).

**sum += \*pX;** adds the value pointed by (**pX**).



Above figure shows how (`pX`) points to the array (`X`). `pX++` moves the pointer to the next item in the original array.

## 7.7. Pointer to Structure

Structures are big variables. The only difference that Structure contains inner members. Following example illustrate how to use pointer to structures variable and array.

### Example 10: Using Pointer to Structure

```
#include "stdio.h"

struct SPerson
```

```

{
    char m_Name[18];
    int m_ID;
    char m_Age;
    short m_Salary;
    double m_Weight;
};

void main()
{
    struct SPerson manager =
        {"Mohamed Hady", 162, 39, 3000, 79.5};
    struct SPerson employees[] = {
        {"Mostafa Said", 163, 30, 1500, 81.0},
        {"Ahmed Salah", 164, 25, 1200, 91.0},
        {"Safa Fayed", 165, 28, 1400, 65.0}};
    int i;

    struct SPerson* p;
    p = &manager;
    printf("manager: %s, %d, %d, %d, %lf\r\n",
           p->m_Name, p->m_ID, (int)p->m_Age,
           (int)p->m_Salary, p->m_Weight);

    p->m_Salary = 4000;
    printf("manager: %s, %d, %d, %d, %lf\r\n",
           manager.m_Name, manager.m_ID,
           (int) manager.m_Age, (int) manager.m_Salary,
           manager.m_Weight);

    p = employees;
    for(i=0;i<sizeof(employees)/sizeof(struct SPerson);
        i++, p++)
    {
        printf("employee %d: %s, %d, %d, %d, %lf\r\n",
               i+1, p->m_Name, p->m_ID,
               (int)p->m_Age, (int)p->m_Salary,
               p->m_Weight);
    }
}

```

**SPerson\*** **p**; is a pointer of type (**SPerson**).

**p = &manager;** stores the address of structure (**manager**) into the pointer p.

**p->m\_Salary = 4000;** means assigning 4000 value to the inner value (**m\_Salary**) of the pointer (p). Know that (p) is pointing to (**manager**) variable.

**p = employees;** stores the address of structure array (**employees**) into the pointer p.

`p++;` increment the pointer value by one element, so that it points to the next value of the array.

## 7.8. Pointers and Functions

Pointers are used efficiently with functions. Using pointers provides two main features:

1. Fast data transfer, because only pointers are transferred.
2. Pointers allow the definition of several outputs for the same function.

### Example 11: Fast Data Transfer Using Pointers

Following program uses (**SStudent**) structure to store student information. It is required to calculate the total student degree after the adjustment and the addition of external activities degrees.

```
#include "stdio.h"

struct SDate
{
    int m_Day;
    int m_Month;
    int m_Year;
};

struct SStudent
{
    char m_Name[256];
    char m_Description[8192];
    struct SDate m_BirthDate;
    double m_Degrees[10];
    double m_TotalDegrees;
};

struct SStudent SlowUpdateTotalDegree(struct SStudent student)

{
    student.m_TotalDegrees = 0;
    int i = 0;
    for(i=0;i<10;i++)
        student.m_TotalDegrees += student.m_Degrees[i];

    return student;
}

void FastUpdateTotalDegree(struct SStudent* pStudent)
{
    pStudent->m_TotalDegrees = 0;
    int i = 0;
```

```

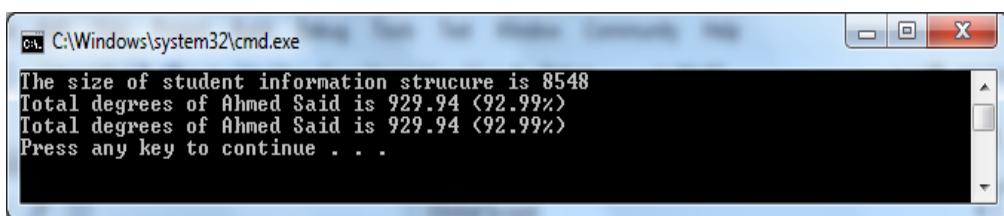
for(i=0;i<10;i++)
    pStudent->m_TotalDegrees += pStudent->m_Degrees[i];
}

void main()
{
    struct SStudent S = {"Ahmed Said", "Ahmed's description",
                         {22, 12, 1990},
                         {88, 98, 88, 92, 98, 87, 66, 94, 87, 99}};

    //Method 1: Without Pointers
    S = SlowUpdateTotalDegree(S);
    printf("Total degrees of %s is %2.2lf (%2.2lf%%)\r\n",
           S.m_Name, S.m_TotalDegrees,
           (double) (100.0 * S.m_TotalDegrees/1000.0));

    //Method 2: With Pointers
    FastUpdateTotalDegree(&S);
    printf("Total degrees of %s is %2.2lf (%2.2lf%%)\r\n",
           S.m_Name, S.m_TotalDegrees,
           (double) (100.0 * S.m_TotalDegrees/1000.0));
}

```



Above program uses two methods:

**Method 1:** Implements (**SlowUpdateTotalDegree**) without using pointers. The function takes the structure variable value, updates the total degree member (**m\_TotalDegrees**), and finally returns the updated structure.

**Method 2:** Implements (**FastUpdateTotalDegree**) with pointers. The function sends only the structure variable pointer, directly update the total degree member (**m\_TotalDegrees**) of the original structure variable.

Method 1 sends the structure value and return it back which means transmitting the whole structure size twice, which is  $2 * 8548$  byte = 17096.

Method 2 sends only structure value pointer and does not return any thing because the pointer allow the function to change the original data. Method 2 transmit only 4 bytes (the pointer size).

To actually prove the efficiency and feel the difference, every operation will be repeated 1 million times, the total spent time in each method is calculated and printed.

```

#include "stdio.h"
#include "time.h"

...
...

void main()
{
    int i;
    struct SStudent S = {"Ahmed Said", "Ahmed's description",
                         {22, 12, 1990},
                         {88, 98, 88, 92, 98, 87, 66, 94, 87, 99}};
    clock_t startTime, endTime;

    printf("The size of student information strucure
          is %d\r\n\r\n", sizeof(struct SStudent));

    //Method 1: Without Pointers
    startTime = clock();
    for(i=0;i<1000000;i++)
        S = SlowUpdateTotalDegree(S);

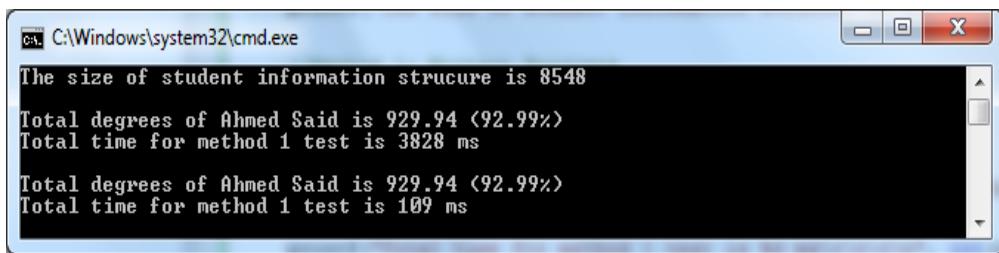
    printf("Total degrees of %s is %2.2lf (%2.2lf%%)\r\n",
           S.m_Name, S.m_TotalDegrees,
           (double) (100.0 * S.m_TotalDegrees/1000.0));

    endTime = clock();
    printf("Total time for method 1 test is %d ms\r\n\r\n",
           (int) (endTime-startTime));

    //Method 2: With Pointers
    startTime = clock();
    for(i=0;i<1000000;i++)
        FastUpdateTotalDegree(&S);

    printf("Total degrees of %s is %2.2lf (%2.2lf%%)\r\n",
           S.m_Name, S.m_TotalDegrees,
           (double) (100.0 * S.m_TotalDegrees/1000.0));
    endTime = clock();
    printf("Total time for method 1 test is %d ms\r\n\r\n",
           (int) (endTime-startTime));
}

```



The size of student information strucure is 8548  
 Total degrees of Ahmed Said is 929.94 (92.99%)  
 Total time for method 1 test is 3828 ms  
 Total degrees of Ahmed Said is 929.94 (92.99%)  
 Total time for method 1 test is 109 ms

Above bechmark test gives that: method 1 takes 3828 millisecons however method 2 takes 109 ms.

Hint: clock is a system function defined in “time.h”. It calcutes the time spent in millisecons science the program start.

### Example 12: Definition of Several Outputs for Function using Pointers

It is required to take an array and calculates the average, the variance, and the standard deviation.

Know that:

$$\text{Mean: } \mu = \frac{\sum_{i=1}^n X_i}{N}$$

$$\text{Variance: } \sigma^2 = \frac{\sum_{i=1}^n (X_i - \mu)^2}{N}$$

Standard Deviation:  $\sigma$

#### Method 1: Without Pointers

```
#include "stdio.h"
#include "math.h"

double CalcMean(double values[], int nValues)
{
    double sum = 0;
    int i;
    for(i=0;i<nValues;i++)
        sum += values[i];
    return sum/nValues;
}

double CalcVariance(double values[], int nValues)
{
    double sum = 0, mean;
    int i;
    mean = CalcMean(values, nValues);
    for(i=0;i<nValues;i++)
        sum += (values[i] - mean) * (values[i] - mean);
    return sum / (nValues * (nValues - 1));
}
```

```

        for(i=0;i<nValues;i++)
            sum += (values[i]-mean)*(values[i]-mean);
        return sum/nValues;
    }

double CalcSTD(double values[], int nValues)
{
    double variance;
    variance = CalcVariance(values, nValues);
    return sqrt(variance);
}

void main()
{
    double values[] = {81.7, 23.8, 74.6, 74.6, 18.7, 61.5,
    36.2, 16.5, 34.6, 51.2, 53.1, 26.5, 34.6, 72.1, 53.6};
    int nValues = sizeof(values)/sizeof(double);
    double mean, variance, STD;

    mean = CalcMean(values, nValues);
    variance = CalcVariance(values, nValues);
    STD = CalcSTD(values, nValues);

    printf("mean: %lf, variance: %lf, std: %lf\r\n",
           mean, variance, STD);
}

```

### Method 2: With Pointers

```

#include "stdio.h"
#include "math.h"

void CalcStatistics(double values[], int nValues, double*
pMean, double* pVariance, double* pSTD)
{
    double sum = 0;
    int i;
    for(i=0;i<nValues;i++)
        sum += values[i];
    *pMean = sum/nValues;

    sum = 0;
    for(i=0;i<nValues;i++)
        sum += (values[i]-*pMean)*(values[i]-*pMean);
    *pVariance = sum/nValues;

    *pSTD = sqrt(*pVariance);
}

```

```

void main()
{
    double values[] = {81.7, 23.8, 74.6, 74.6, 18.7, 61.5,
    36.2, 16.5, 34.6, 51.2, 53.1, 26.5, 34.6, 72.1, 53.6};
    int nValues = sizeof(values)/sizeof(double);
    double mean, variance, STD;

    CalcStatistics(values, nValues, &mean, &variance, &STD);

    printf("mean: %lf, variance: %lf, std: %lf\r\n",
           mean, variance, STD);
}

```

Method 1 defines three separate function (**CalcMean**, **CalcVariance**, **CalcSTD**). The functions contain a lot of code and repeated operation, because variance depends on the mean, and STD depends on the variance.

Method 2 defines one function that calculate the three results once and internally reuse the generated outputs to minimize the overall operations.

## 7.9. Passing Arrays and Pointers to Functions

**Method 1:** Normal Array Passing

```

#include "stdio.h"

void Sort(int values[], int nValues)
{
    int i, j, temp;
    for(i=0;i<nValues-1;i++)
        for(j=i;j<nValues;j++)
            if(values[i]>values[j])
            {
                temp = values[i];
                values[i] = values[j];
                values[j] = temp;
            }
}
void main()
{
    int i, values[5] = {89,73,28,94,32};
    Sort(values, 5);
    for(i=0;i<5;i++)
        printf("%d\r\n", values[i]);
}

```

When the function (**Sort**) is called the values array reference is sent. Sending array reference gives the function the ability to read and to change the contents of the sent item.

If above program is re-written using pointers:

### Method 2: Array Passing with Pointers

```
#include "stdio.h"

void Sort(int* values, int nValues)
{
    int i, j, temp;
    for(i=0;i<nValues-1;i++)
        for(j=i;j<nValues;j++)
            if(values[i]>values[j])
            {
                temp = values[i];
                values[i] = values[j];
                values[j] = temp;
            }
}

void main()
{
    int i, values[5] = {89,73,28,94,32};
    Sort(values, 5);
    for(i=0;i<5;i++)
        printf("%d\r\n", values[i]);
}
```

Method 2 is completely equivalent to Method 1 which means that:

<code>void Sort(int values[], int nValues)</code>	is equivalent to	<code>void Sort(int* values, int nValues)</code>
---	------------------	--

Programmer is free to choose which notation is suitable, because both methods gives the same behaviour.

## 7.10. Function Parameters Types

Finally we can summarize function parameters types.

1. Input Parameters (Calling by Value)  
The parameter values is completely transmitted to the function. This gives the function the ability to **read** the transmitted data **only**.
2. Input/Output Parameters (Reference or Pointer)  
The parameter pointer (reference) is transmitted only. This gives the function the ability to **read from** and **write to** the original parameters.
3. Output Parameters (Return Value)

The return data of the function is assumed as an output parameter. Normally C does not provide other Output parameters except the return value.

The sort function contains the two types of parameters.

```
void Sort(int* values /*input/output*/, int nValues /*input*/)
{
    int i, j, temp;
    for(i=0;i<nValues-1;i++)
        for(j=i;j<nValues;j++)
            if(values[i]>values[j])
            {
                temp = values[i];
                values[i] = values[j];
                values[j] = temp;
            }
}
```

**values** parameter is an input/output parameter.

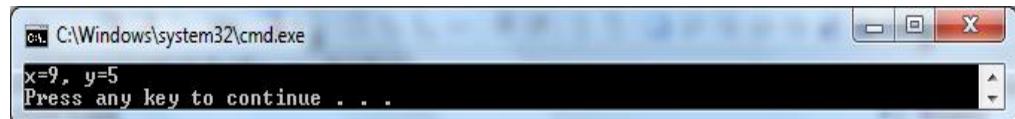
**nValues** parameter is an input parameter.

## 7.11. Pointer Data Type

Normally pointers are linked to certain data type which affects the data access operation using that pointer. For example (**int\* pX**) means (**pX**) is a pointer that point to an integer value so that it can be used indirectly to read or write integer values. This idea is illustrated by the following code.

```
#include "stdio.h"

void main()
{
    int x = 5, y;
    int* pX = &x;
    y = *pX;
    *pX = 9;
    printf("x=%d, y=%d", x, y);
}
```



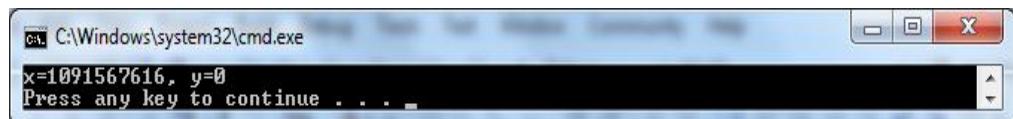
(**y = \*pX**) means using pointer to read the contents of x indirectly.

(**\*pX = 9**) means using pointer to write a value to x indirectly.

Programmer must specify the pointer type correctly to match the type of the pointed data. Following code enforces wrong pointer type, which will lead to wrong data assignment.

```
#include "stdio.h"

void main()
{
    int x = 5, y;
    float* pX;
    pX = (float*) &x;
    y = *pX;
    *pX = 9;
    printf("x=%d, y=%d\r\n", x, y);
}
```



(**float\* pX**) means defining a pointer to float.

(**pX = (float\*) &x**) means using type casting to enforce the integer address assignment to the float pointer variable. This means (**pX**) of type float pointer is pointing to (**x**) of type integer.

(**y = \*pX**) means using the float pointer (**pX**) to read the contents of x indirectly. (**\*pX**) will bring float value from (**x**), however (**x**) contains an integer value, which will lead to invalid data retrieval.

(**\*pX = 9**) means using float pointer (**pX**) to write a value to (**x**) indirectly. (**\*pX**) will treat the value (**9**) as a float value and store it in (**x**) in float format, which will lead to an invalid data retrieval.

Finally, it is clear from the program output that both x and y contains an invalid data because of the misuse of pointers.

## 7.12. Dealing with Pointer Address Value

Pointer is a variable that holds an address. Memory address is an integer value. The pointer variable size must support the maximum address the computer can support. In 32 Bit computers maximum address is ( $2^{32}-1$ ) which means that the required variable size is 4 bytes (32 Bit). In 64 Bit computers maximum address is ( $2^{64}-1$ ) which means that the required variable size is 8 bytes (64 Bit).

It is applicable to add or subtract a value from pointer, however this will not be treated like normal addition or subtraction. Following code shows this behaviour.

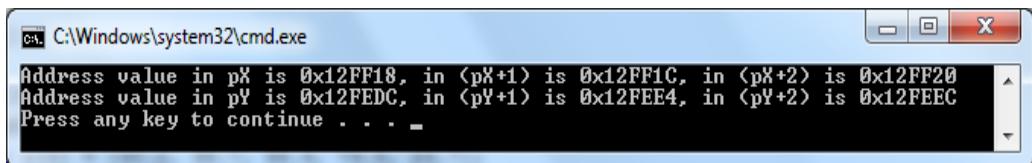
```
#include "stdio.h"

void main()
{
    int x[5] = {72, 30, 91, 29, 74};
    int* pX = x;
    double y[5] = {90.2, 38.7, 98.4, 79.8, 23.7};
    double* pY = y;

    printf("Address value in pX is 0x%X, in (pX+1) is 0x%X,
           in (pX+2) is 0x%X\r\n", pX, pX+1, pX+2);

    printf("Address value in pY is 0x%X, in (pY+1) is 0x%X,
           in (pY+2) is 0x%X\r\n", pY, pY+1, pY+2);

}
```



(**int\* pX = x**) means assigning integer array address (**x**) to the integer pointer (**pX**).  
 (**double\* pY = y**) means assigning double array address (**y**) to the integer pointer (**pY**).

Normally (**pX+1**) means adding a value of (1), however in integer pointer case it means adding (4). (4) represents the size of integer variable. This idea is clear in the printed values for (**pX**, **pX+1**, **pX+2**), it appears that the difference is (4).

Also, normally (**pY+1**) means adding a value of (1), however in double pointer case it means adding (8). (8) represents the size of double variable. This idea is clear in the printed values for (**pY**, **pY+1**, **pY+2**), it appears that the difference is (8).

Generally:

$\text{pointer} + i = \text{pointer} + i * \text{sizeof(pointer data type)}$   
 $\text{pointer} - i = \text{pointer} - i * \text{sizeof(pointer data type)}$

### 7.13. Pointer with Unknown Type (**void\***)

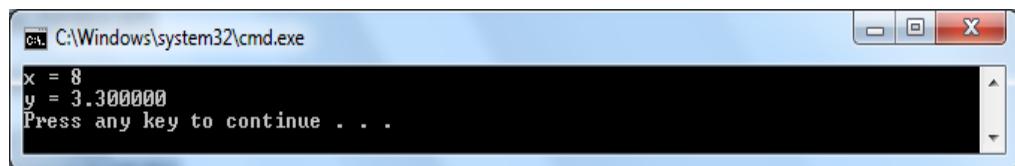
Programmer can define a general pointer without specifying a linked data type. This type of pointers called (**void pointer**). **void pointer** can not be used normally to manipulate data, it is required to **type cast** each data access operation.

```
#include "stdio.h"

void main()
{
    int x = 5;
    double y = 10.3;
    void* p;

    p = &x;
    *(int*)p = 8;
    printf("x = %d\r\n", x);

    p = &y;
    *(double*)p = 3.3;
    printf("y = %lf\r\n", y);
}
```



(**p = &x**) means the address of (**x**) is stored in the (**void pointer p**).

(\***(int\*)p = 8**) means the (**void pointer p**) is first casted to (**int\***) and this allows the computer to assign an integer value (8) correctly.

(**p = &y**) means the address of (**y**) is stored in the (**void pointer p**).

(\***(double\*)p = 3.3**) means the (**void pointer p**) is first casted to (**double\***) and this allows the computer to assign the double value (3.3) correctly.

### Example 13: Universal Compare with void Pointers

Following program implements a compare function that compares any two values, integer or double.

```
#include "stdio.h"

int Compare(void* value1, void* value2, int type)
{
    int r;
    switch(type)
    {
        case 1://Integer
```

```

        if( *(int*)value1 == *(int*)value2 )r = 0;
        else if( *(int*)value1 > *(int*)value2 )r = 1;
        else r = -1;
        break;

    case 2://double
        if( *(double*)value1 == *(double*)value2 )r = 0;
        else if( *(double*)value1 > *(double*)value2 )r = 1;
        else r = -1;
        break;
    }

    return r;
}

void main()
{
    int x1 = 5, x2 = 6, x3 = 5;
    double y1 = 10.3, y2 = 8.3, y3 = 11.9;

    printf("Compare x1 and x2 gives %d\r\n",
           Compare(&x1, &x2, 1));
    printf("Compare x1 and x3 gives %d\r\n",
           Compare(&x1, &x3, 1));
    printf("Compare y1 and y2 gives %d\r\n",
           Compare(&y1, &y2, 2));
    printf("Compare y1 and y3 gives %d\r\n",
           Compare(&y1, &y3, 2));
}

```

```

C:\Windows\system32\cmd.exe
Compare x1 and x2 gives -1
Compare x1 and x3 gives 0
Compare y1 and y2 gives 1
Compare y1 and y3 gives -1
Press any key to continue . .

```

The function prototype is:

```
int Compare(void* value1, void* value2, int type)
```

which means it takes any two pointers with unknown type, the third parameter informs the type of the submitted values (1 means integer, 2 means double).

## 7.14. Pointer to Pointer

Pointers hold an address value of other variables, Pointer to Pointer holds an address value of other pointers. Following example shows how to define and to use the Pointer to Pointer.

### Example 14: Using Pointer to Pointer

```
#include "stdio.h"

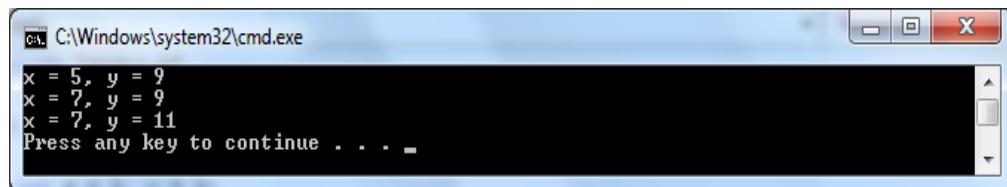
void main()
{
    int x = 5, y = 9;
    int* px = &x; //Pointer
    int** ppX = &px; //Pointer to Pointer

    printf("x = %d, y = %d\n", x, y);

    **ppX = 7;
    printf("x = %d, y = %d\n", x, y);

    *px = &y;

    *px = 11;
    printf("x = %d, y = %d\n", x, y);
}
```



(**int\* px = &x**) menas defining a pointer (**px**) and assign to it the address of (**x**).

(**int\*\* ppX = &px**) menas defining a pointer to pointer (**ppX**) and assign to it the address of (**px**).

(**\*\*ppX = 7**) menas writing a value of (7) to (**x**). This happens because (**ppX**) points to (**px**) and (**px**) points to (**x**).

(**\*px = &y**) menas altering the address value in (**px**) with the address of (**y**). This happens because (**ppX**) points to (**px**).

(**\*px = 11**) menas writing a value of (11) to (**y**). This happens because in previous step (**px**) is pointed to (**y**).

### Example 15: Finding Employees with Maximum and Minimum Salaries

It is required to write a one function that retrieve a pointer to the employee with maximum salary and the employee with minimum salary.

```

#include "stdio.h"

struct SEmployee
{
    char m_Name[50];
    int m_Salary;
};

void FindMaxMin(struct SEmployee* employees, int nEmployees,
                struct SEmployee** ppMaxEmployee,
                struct SEmployee** ppMinEmployee)
{
    int i=0;
    *ppMaxEmployee = &employees[0];
    *ppMinEmployee = &employees[0];
    for(i=0;i<nEmployees;i++)
    {
        if(employees[i].m_Salary >
           (*ppMaxEmployee)->m_Salary)
            *ppMaxEmployee = &employees[i];

        if(employees[i].m_Salary<
           (*ppMinEmployee)->m_Salary)
            *ppMinEmployee = &employees[i];
    }
}

void main()
{
    struct SEmployee employees[] = {{"Ahmed", 800},
                                    {"Ali", 1800}, {"Mostafa", 1000},
                                    {"Mohamed", 300}, {"Said", 500}};
    struct SEmployee* pMaxEmployee = NULL;
    struct SEmployee* pMinEmployee = NULL;

    FindMaxMin(employees,
               sizeof(employees)/sizeof(struct SEmployee),
               &pMaxEmployee, &pMinEmployee);
    printf("Employee with maximum salary is %s, %d\r\n",
          pMaxEmployee->m_Name, pMaxEmployee->m_Salary);
    printf("Employee with minimum salary is %s, %d\r\n",
          pMinEmployee->m_Name, pMinEmployee->m_Salary);
}

```



**FindMaxMin** function takes two pointers to pointers, the first one is filled with the address of the employee with maximum salary, the second one is filled with the address of the employee with minimum salary.

### 7.15. NULL and Unassigned Pointers

If the pointer is unassigned it will contain an invalid address, it is unsafe to use an unassigned pointer, normally the program will crash. Following program will crash because the (**pX**) pointer is not pointed to a valid address, it contains a memory garbage.

```
#include "stdio.h"

void main()
{
    int* pX;
    printf("pX point to %d", *pX);
}
```

To avoid using unassigned pointers, all pointers must hold a valid address, if not it must hold a zero value. Sometimes zero value called (NULL). Above program may be fixed as shown below:

```
#include "stdio.h"

void main()
{
    int* pX = NULL;
    if(pX!=NULL)
        printf("pX point to %d", *pX);
    else
        printf("pX is not initialized");
}
```

### 7.16. Pointer to Function

Pointer can hold also the address of program functions, to illustrate this idea, see the following example:

#### Example 16: Using Pointer to Function

```
#include "stdio.h"

int IntCompare(void* value1, void* value2)
{
```

```

    if( *(int*)value1 == *(int*)value2 )return 0;
    else if( *(int*)value1 > *(int*)value2 )return 1;

    return -1;
}

int DoubleCompare(void* value1, void* value2)
{
    if( *(double*)value1 == *(double*)value2 )return 0;
    else if( *(double*)value1 > *(double*)value2 )return 1;

    return -1;
}

void main()
{
    int x1 = 5, x2 = 6, x3 = 5;
    double y1 = 10.3, y2 = 8.3, y3 = 11.9;

    int (*fpCompare) (void*,void*) = NULL; //Function Pointer

    fpCompare = IntCompare;
    printf("Compare x1 and x2 gives %d\r\n",
           fpCompare(&x1, &x2));
    printf("Compare x1 and x3 gives %d\r\n",
           fpCompare(&x1, &x3));

    fpCompare = DoubleCompare;
    printf("Compare y1 and y2 gives %d\r\n",
           fpCompare(&y1, &y2));
    printf("Compare y1 and y3 gives %d\r\n",
           fpCompare(&y1, &y3));
}

```

`(int (*fpCompare) (void*,void*) = NULL;)` means defining a function pointer of name `(fpCompare)`, this function returns an `(int)` and takes two parameters `(void*,void*)`.

`(fpCompare = IntCompare)` means assigning the `fpCompare` pointer the address of `IntCompare` function. Later calls will be directed to `IntCompare` function.

`(fpCompare = DoubleCompare)` means assigning the `fpCompare` pointer the address of `DoubleCompare` function. Later calls will be directed to `DoubleCompare` function.

## 7.17. Advanced String Manipulation with Pointers

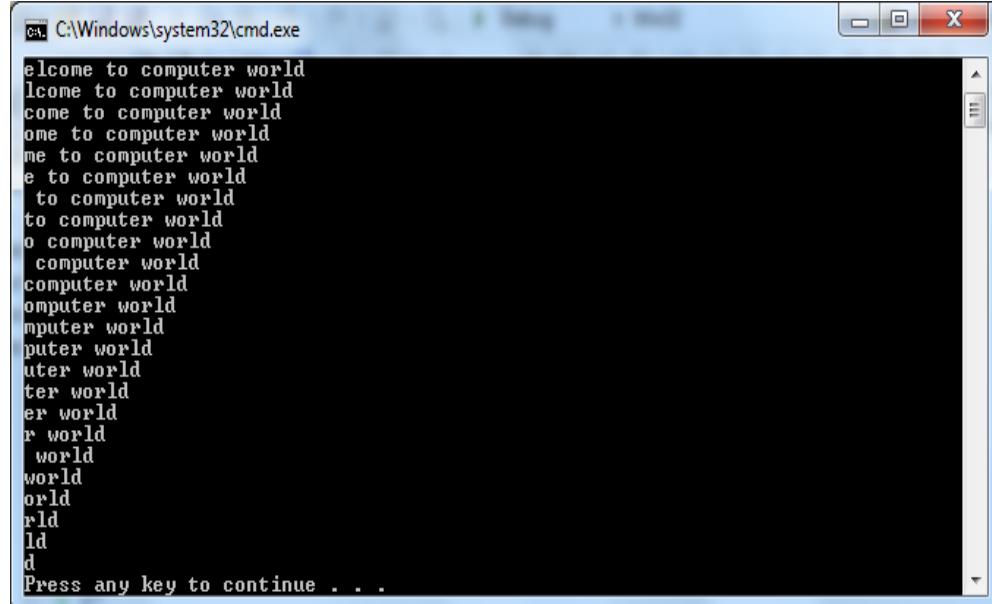
String is an array of characters. Array name represents the beginning address of the array. Each element is one character with a size of one byte. Following examples illustrate how to use pointers with strings.

### Example 17: Playing with Text

```
#include "stdio.h"

void main()
{
    char Text[] = "Welcome to computer world";
    char* pText = Text;

    do
    {
        printf("%s\r\n", pText);
        pText++;
    }
    while ((*pText));
}
```



`(char* pText = Text)` means the Text array address is stored in the pointer pText.

`(printf("%s\r\n", pText))` means printing the characters starting from the address pText.

(**pText++**) means incrementing the stored address in the pText pointer, so that it points to the next character.

(**while ((\*pText))**) means looping until the pText reaches the null termination of the text.

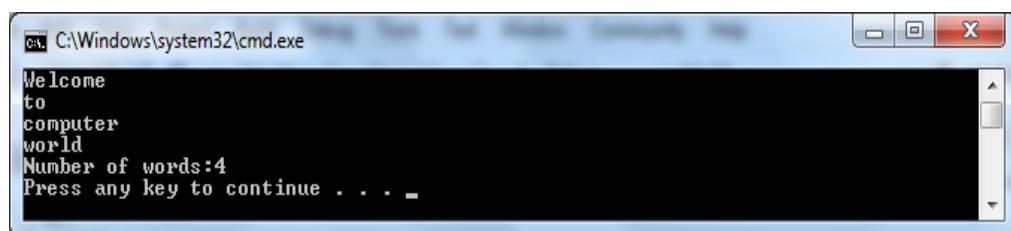
### Example 18: Tokenizing Strings

For any given text paragraph, count and print all words inside it.

```
#include "stdio.h"
#include "string.h"

void main()
{
    char text[] = "Welcome to computer world.";
    char separators[] = ".;\\/-,\\'\"";
    char* pToken = NULL;
    int count = 0;

    pToken = strtok( text, separators );
    while( pToken != NULL )
    {
        printf("%s\r\n", pToken);
        pToken = strtok(NULL, separators);
        count++;
    }
    printf("Number of words:%d\r\n", count);
}
```



String library provide a very useful function (strtok) this function tokenize any string with the given separators.

In above program strtok function initially takes the text to be tokenized and the list of all expected separators. At beginning, it returns a pointer to the first word. Later calls return a pointer to the other words. When there is no word found it returns NULL.

## 7.18. Coding Convention

### 7.18.1. Pointer Naming

Pointer must begin with ‘p’ letter.

```
int *pValue;
```

Pointer to Pointer must begin with “pp” letters.

```
int **ppValue;
```

Function Pointer must begin with “fp” letters.

```
int (*fpCompare) (void*,void*) = NULL;
```

## 7.18 Exercise

**Q1.**

Write down and discuss the expected output of the following program, assuming X and P are located at the addresses (0x1024, 0x1128).

```
int    X = 3;
int*  P = &X;
printf("%d %d %d %d", *P, P, &X, &P);
```

**Q2.**

Write down and discuss the expected output of the following program, assuming M and P is located at the addresses (0x1024, 0x1128).

```
int    M[] = {3,4,5,2};
int*  P;

P = M;
printf("\n%u %u %u %u", *P, *(P+1), P[2], *(M+3));

P = M;
printf("\n%u %u %u %u", P, (P+1), &P[2], (M+3));

P = &M[1];
printf("\n%u %u %u", *P, P-M, (int)P-(int)M);
```

**Q3.**

Write down and discuss the expected output of the following program.

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

void F1(int* A, int* B, int* C)
{
    int D;
    D = *A;
    *A = *B;
    *B = *C;
    *C = D;
}

void main()
{
    int    X = 7, Y = 8, Z = 2;

    printf("\n%d %d %d", X, Y, Z);
    F1(&X, &Y, &Z);
    printf("\n%d %d %d", X, Y, Z);
}
```

**Q4.**

Write down and discuss the expected output of the following program.

```
float F[] = {1.2, 3.4, 7.1, 3.1};
float* P = F;

printf("\n%f", *P);
P++;
printf("\n%f", *P);
P+=2;
printf("\n%f", *P);
```

**Q5.**

Write down and discuss the expected output of the following program.

```
int X = 4;
float F = 7.1;
void* P;

P = &X;
printf("\n%d", *(int*)P);

P = &F;
printf("\n%f", *(float*)P);
```

**Q6.**

Write a function that takes a string consists of several numbers separated by a space character and convert it into an array of double values. Consider the following program:

```
int GetValues(char* Text, double* Values)
{
    int nValues = 0;
    ...
    return nValues;
}

void main()
{
    char Text[] = "7672.28 276763.22 0.767 1.2878 772.2 1878
152 0.0123";
    double Values[100];
    int nValues;

    nValues = GetValues(Text, Values);

    for(int i=0;i<nValues;i++)
        printf("\n%lf", Values[i]);
}
```

Hint: use strtok function to divide the given string into pieces.

**Q7.**

Answer Q6 without using strtok function.

**Q8.**

Write one function that can sort arrays of values with following types (short, int, float and double). Consider the following program:

```
#include "stdio.h"
#include "string.h"

//type 1:short, 2:int, 3:float, 4:double
void Sort(void* values, int nValues, int type)
{
}

void main()
{
    short x[5] = {21, 90, 83, 92, 49};
    float y[5] = {12.7f, 98.3f, 79.0f, 82.1f, 74.9f};
    int i;

    Sort(x, 5, 1);
    printf("\r\nsorted x:\r\n");
    for(i=0;i<5;i++)
        printf("%d\r\n", x[i]);

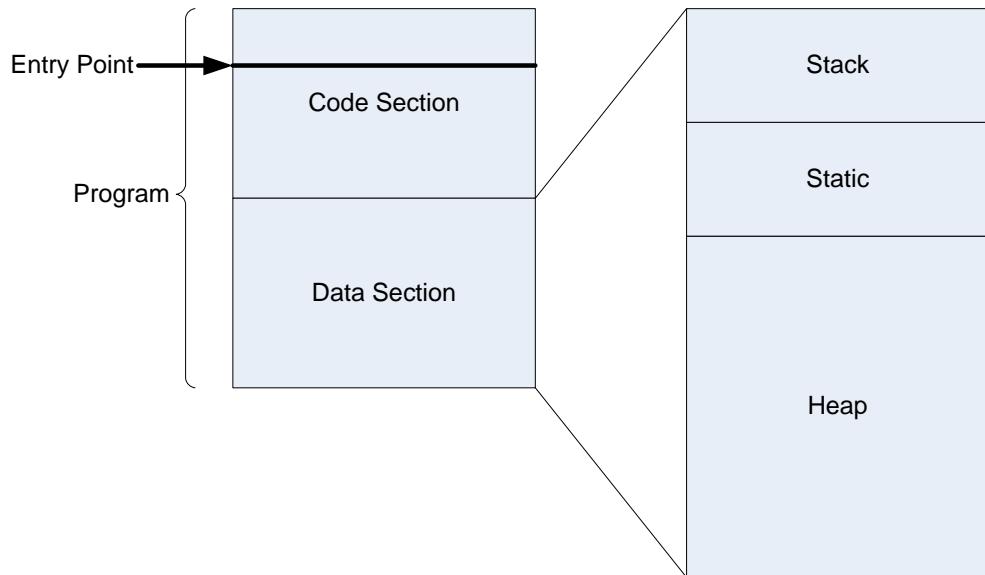
    Sort(y, 5, 3);
    printf("\r\nsorted y:\r\n");
    for(i=0;i<5;i++)
        printf("%d\r\n", y[i]);
}
```

## Chapter 8: Memory and Files

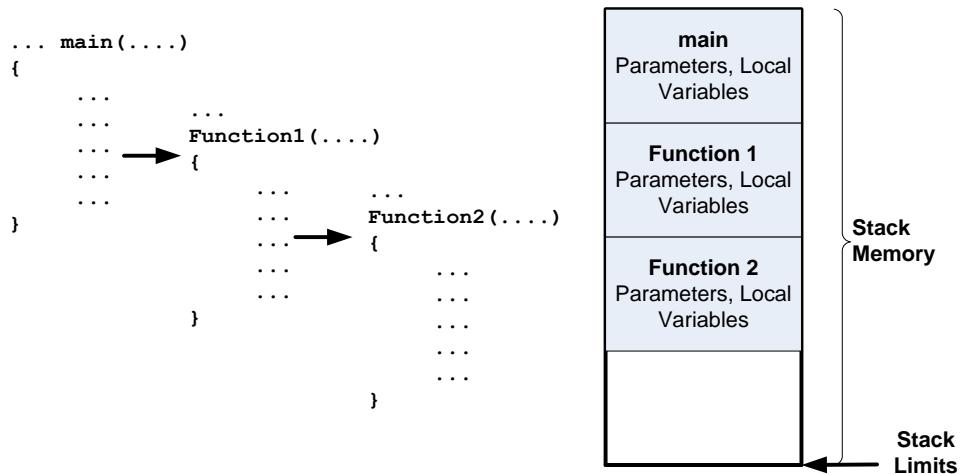
## 8.1. Memory

Computer memory is used to store programs' code and data. Whenever the program starts, all code and data are loaded into memory. After loading the processor starts to execute the program from a location in the code called the entry point. The entry code does some preparation then starts the execution of the main function of the program (main, WinMain, tmain...).

Program memory is divided into two principles sections, the code, and the data. Code section holds all program code. Data section holds all program data and is divided into three sub-sections, the Stack, the Heap, and the Static Data Area.



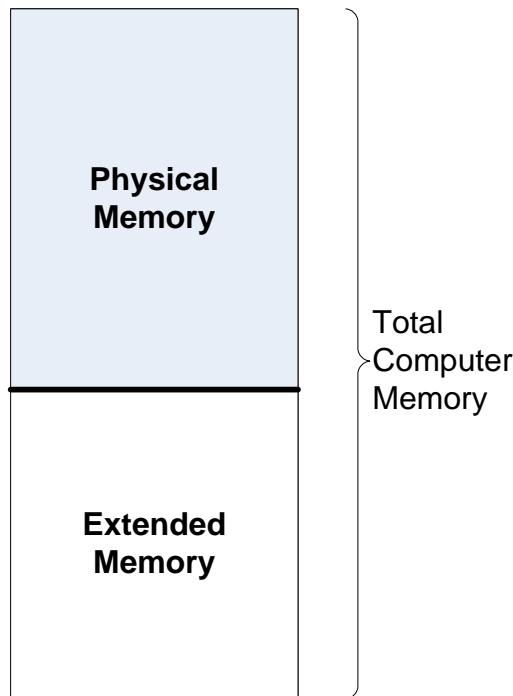
*Stack Memory* Section, is a temporary memory where functions stores all local variables including the parameters and the return value. Whenever a function is called, a new section in the stack is reserved to hold the parameters, the local variables, the return address, and other information. If another function is called from inside this function, another area is reserved from the stack, and so on. When the function finishes the execution all allocated area in the stack is freed. Nested functions calling are applicable until there is no extra space in the stack. If the stack is full further nested function calls gives an error “**Stack Overflow**”. The programmer should design the program to avoid “Stack Overflow” error. If more space is required, the stack memory size must be configured to have larger size. Usually stack size is configured from the project settings options.



*Static memory* holds all global and static variables in the program. This memory is normally small or taken from the top of the *Heap memory* at some systems.

*Heap memory* contains all dynamic variables. Dynamic variable is allocated and freed while program execution. Dynamic memory allocation is discussed in details in the following sections. Programmer can reserve as much memory as the heap memory can hold. In some operating systems *Heap memory* size is limited, in other systems the *Heap size* is limited by the total available memory in the computer.

Total computer memory equals to the virtual memory size. Virtual memory consists of the actual physical memory and the other extended memory placed on the hard disk. Extended memory access is slow, when the program starts to use the extended memory, the overall computer speed decreases.



Following program illustrate different variable types.

```

int g;          //Global Variables at Static Memory

int TestFunction(int x, int y)
//Input parameters and return value at Stack Memory
{
    static int c = 0; //Static Variables at Static Memory
    int z;           //Local Variables at Stack Memory
    z = x + y + c + g;
    c++;
    return z;
}

void main()
{
    int a = 5, b = 6, c;//Local Variables at Stack Memory
    g = 10;
    c = TestFunction(a, b);
    printf("c = %d\r\n", c);
}

```

## 8.2. Static Memory

All local, static, and global variables are static memory. Static memory size is defined while programming and there is no way to change it at runtime. Following example illustrate the usage of the static memory.

### Example 1: Average Temperature using Static Memory

It is required to evaluate the average temperature for all provided temperature values. The program should ask user to supply the temperature.

Clearly, with static memory, there is no way to write a program without limiting the maximum number of the temperature values.

```
#include "stdio.h"
#include "conio.h"

void main()
{
    float degrees[1000];
    int count = 0;
    float total = 0;
    int i;

    while(count<10)
    {
        printf("\nEnter a degree :");
        scanf("%f", &degrees[count++]);
        printf("do you want to add more degrees (y/n)?");
        if(getche() !='y')break;
    }

    for(i=0;i<count;i++)
        total += degrees[i];

    printf("\nAverage students degree : %f", total/count);
}
```

To allow entering large number of temperature values, the maximum value is sited to 1000. If the user supplies 10 values 990 values will be unused.

Static memory usually imposes a limitation on variables size and program functionality. If it is required to support large number of records, even if this happens rarely, large variables must be defined. Even with large variables, still there is a limit. Defining large variables may lead to consume the main computer memory and sometimes lead to the common errors “Insufficient memory” or “Low virtual memory”.

It is required to have a new type of memory that can grow and shrink. This type memory called the dynamic memory.

### **8.3. Dynamic Memory**

Dynamic memory provides a way to allocate any amount of flat memory and a way to free this amount. Following example solves the problem of the unlimited data entry of the temperature records.

#### **Example 2: Average Temperature using Dynamic Memory**

```
#include "stdio.h"
#include "stdlib.h"

void main()
{
    float *pDegrees;
    int count;
    float total = 0;
    int i;

    printf("\nHow many records you want to supply :");
    scanf("%d", &count);

    pDegrees = (float*)malloc(count*sizeof(float));

    for(i=0;i<count;i++)
    {
        printf("\nEnter a degree :");
        scanf("%f", &pDegrees[i]);
    }

    for(i=0;i<count;i++)
        total += pDegrees[i];

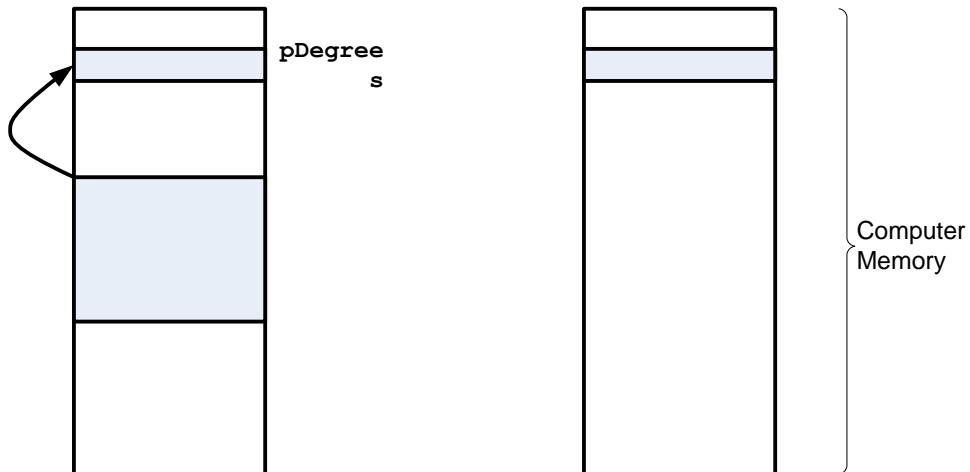
    printf("\nAverage students degree : %f\n", total/count);

    free(pDegrees);
}
```

`pDegrees = malloc(count*sizeof(float))` means allocating a flat memory of size equals to `(count*sizeof(float))` and retuning the address of the memory. The address is then assigned to the pointer (`pDegrees`).

`free(pDegrees)` means freeing the allocated memory and return it back to the system.

```
pDegrees =
(float*)malloc(count*sizeof(float));      free(pDegrees);
```



### Example 3: Simple String Replace Implementation

It is required to make string replace function that replaces any given sentence with a new sentence, longer or shorter.

For example if it is required to replace the word “Computer” with the longer word “Programming” in the text “Welcome to Computer world. Welcome to Computer world.”.

The new text should be “Welcome to Programming world. Welcome to Programming world.” which is longer and requires more memory space.

```
#include "stdio.h"
#include "string.h"
#include "stdlib.h"

char* Replace(char* inText, char* oldString, char* newString)
{
    char* outText = NULL;
    char* pText = inText;
    char* pTempText = NULL;
    int sectionLength = 0;

    outText = (char*)malloc(1);
    outText[0] = '\0';

    while (pTempText = strstr(pText, oldString))
    {
```

```

        sectionLength = (int)pTempText - (int)pText;
        outText = (char*)realloc(outText,
            strlen(outText)+sectionLength+strlen(newString)+1);

        strncat(outText, pText, sectionLength);
        strcat(outText, newString);

        pText = pTempText+strlen(oldString);
    }

    outText = (char*)realloc(outText,
        strlen(outText)+strlen(pText)+strlen(newString)+1);
    strcat(outText, pText);

    return outText;
}

void main()
{
    char orgText[] = "Welcome to Computer world.
                        Welcome to Computer world.";

    char* newText = NULL;
    newText = Replace(orgText, "Computer", "Programming");

    printf("Old Text: %s\n\n", orgText);
    printf("New Text: %s\n\n", newText);

    free(newText);
}

```



**free(pDegrees)** means allocating new memory with a new size and maintaining the original contents.

#### 8.4. Memory Functions

C language provides several useful memory functions such as:

**memcpy**: copy the required size from memory block to another.

**memset**: fills the required size of memory with certain value.

**memcmp:** compares the required size of memory with another memory. It returns 0 for identical matching, negative value if the first memory value is lower, positive value if the first memory is higher.

**sprintf:** prints a formatted text and numbers in a text buffer.

**scanf:** reads formatted text and numbers from a text buffer.

#### Example 4: Comparing and Exchanging Two Arrays

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"

void main()
{
    double X[5] = {87.1, 29.7, 23.8, 74.6, 28.7};
    double Y[5] = {12.1, 88.6, 37.9, 21.5, 98.2};
    double T[5];

    if(memcmp(X,Y,sizeof(X))==0)
    {
        printf("Arrays are identical,
               no need for exchange\r\n");
    }
    else
    {
        memcpy(T, X, sizeof(X));
        memcpy(X, Y, sizeof(X));
        memcpy(Y, T, sizeof(X));
        printf("Exchange is completed successfully\r\n");

        memset(T, 0, sizeof(T));
        //Fill the T array with zeros
    }
}
```

**memcpy(X, Y, sizeof(X))** means coping the memory contents from the address **Y** to the address **X** with the size of (**sizeof(X)**).

#### Example 5: Converting Student Record to/from Text

It is required to write or read student information without using **printf** or **scanf** functions.

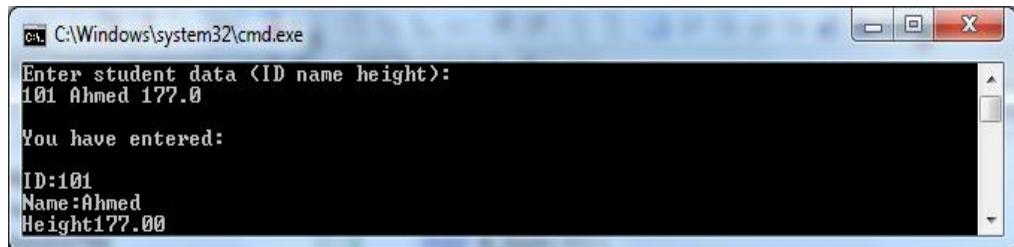
```
#include "stdio.h"
#include "conio.h"
#include "process.h"
```

```
#include "string.h"
#include "stdlib.h"

struct SStudent
{
    int m_ID;
    char m_Name[40];
    float m_Height;
};

void main()
{
    struct SStudent student;
    char text[100];
    puts("Enter student data (ID name height):");
    gets(text);
    sscanf(text, "%d %s %f",
           &student.m_ID, student.m_Name, &student.m_Height);

    sprintf(text, "ID:%d\nName:%s\nHeight%3.2f",
            student.m_ID, student.m_Name, student.m_Height);
    puts("\nYou have entered:\n");
    puts(text);
}
```



## 8.5. Files

File is a storage area managed by the operating system. Files normally stored on the hard drive. Most of the files has a name and an optional extension separated by a dot ‘.’, for example “records.txt” or “mountain.jpg”. The extension normally informs the type of the file. Common file types are images (BMP or JPG or GIF or PNG), documents (DOC or TXT or PDF), executable (EXE or DLL or COM or BAT). The file location is called the path, for example “C:\Test\Example\myfile.dat” means the file “myfile.dat” is located inside the folder “Example” inside the folder “Test” inside the hard drive partition “C”.

C language provides many functions to create, open, read, write and delete files. Also there are other functions responsible on managing folders.

Files like memory contains a flat binary data, however there is some difference?

*Files* must be accessed sequentially, which means you must seek to the required place in the file then performs the read or write operation. For that reason reading from different places in the file takes different time.

*Memory* is a random access device; you can read memory contents at any location in the same access time.

### Example 6: Creating Folders and Files

Following example creates the following file structure:

- C Partition
  - Test Folder
    - Test 1 Folder
    - Test 2 Folder
      - mydata1.txt File
      - mydata2.txt File

```
#include "stdio.h"
#include "direct.h"
#include "conio.h"

void main()
{
    FILE* pFile = NULL;
    printf("Create Folders and Files, press
           any key to continue:\r\n");
    getch();

    mkdir("C:\\Test");
    mkdir("C:\\Test\\Test 1");
    mkdir("C:\\Test\\Test 2");
    pFile = fopen("C:\\Test\\Test 2\\mydata1.txt", "w");
    fclose(pFile);
    pFile = fopen("C:\\Test\\Test 2\\mydata2.txt", "w");
    fclose(pFile);
```

```

printf("Creation completed, press any key
           to continue:\r\n");
getch();
printf("Remove created Folders and Files,
           press any key to continue:\r\n");
getch();

remove("C:\\Test\\Test 2\\mydata1.txt");
remove("C:\\Test\\Test 2\\mydata2.txt");
rmdir("C:\\Test\\Test 1");
rmdir("C:\\Test\\Test 2");
rmdir("C:\\Test");

printf("Removal completed, press any
           key to continue:\r\n");
getch();
}

```

**mkdir ("C:\\Test")** create the folder “C:\\Test”.

**rmdir ("C:\\Test")** removes an existing folder “C:\\Test”.

**pFile = fopen("C:\\Test\\Test 2\\mydata1.txt", "w")** creates a file “C:\\Test\\Test 2\\mydata1.txt” with write-mode and returns a pointer to the file information structure. Opened file is normally locked, others programs could not access the opened file until it closed.

**fclose (pFile)** close the opened file “C:\\Test”.

**remove ("C:\\Test\\Test 2\\mydata1.txt");** removes the existing file “C:\\Test\\Test 2\\mydata1.txt”.

### Example 7: Writing and Reading Characters

It is required to make a program that read every printed character and write it in a file. The program also should read the file back and print it to screen.

There are two functions to write or read single letter to the opened file. Those functions are putc and getch.

```

#include "stdio.h"
#include "conio.h"
#include "process.h"

void main()
{
    FILE * pFile;

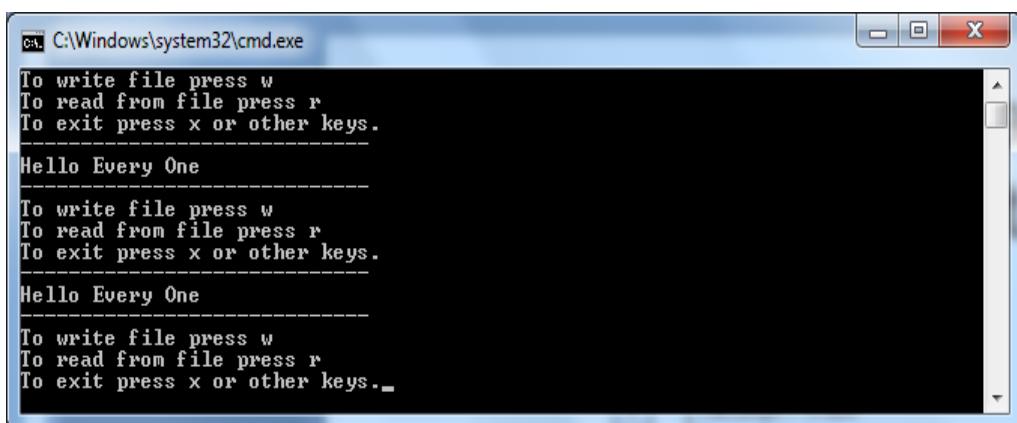
```

```

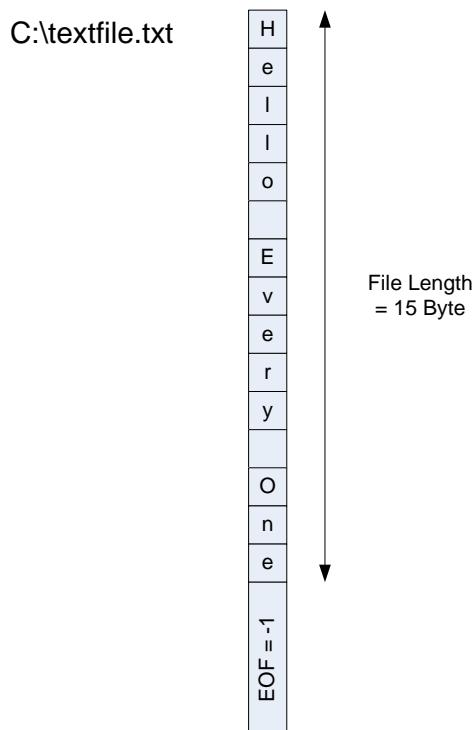
char ch;

while(1)
{
    printf("To write file press w\n To read from file
           press r\n To exit press x or other keys.");
    ch = getch();
    printf("\n-----\n");
    if(ch=='w')
    {
        pFile = fopen("c:\\textfile.txt", "w");
        while((ch=getche())!='r')
            putc(ch, pFile);
        fclose(pFile);
    }
    else if(ch=='r')
    {
        pFile = fopen("c:\\textfile.txt", "r");
        while((ch=getc(pFile))!=EOF)
            printf("%c", ch);
        fclose(pFile);
    }
    else
    {
        exit(0);
    }
    printf("\n-----\n");
}
}

```

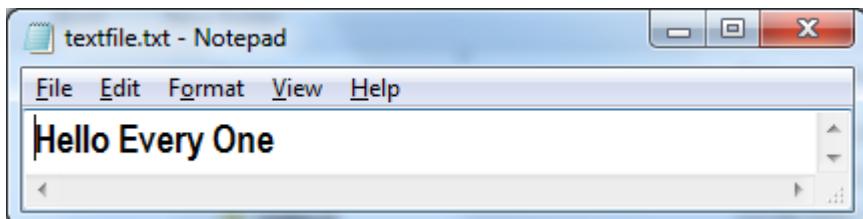


In above example the file “**c:\\textfile.txt**” should contain every typed letter until the user press the carriage return. Following figure describes the file internal structure.



The EOF record is placed by the operating system to recognize the end of file.

The file “**textfile.txt**” should be a normal text file.



### **Example 8: Writing and Reading Line of Strings**

It is required to make a program that read string lines entered by user and write it to a text file. The program also should read the file back line by line and prints it to the screen.

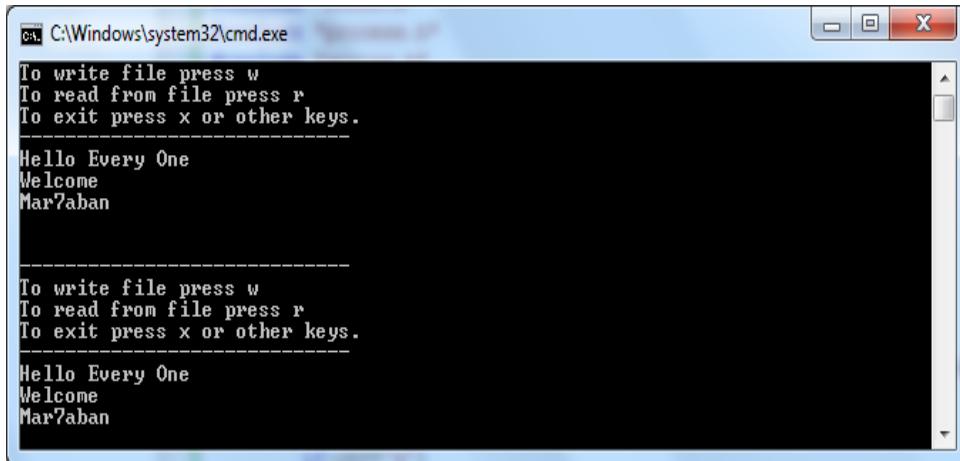
There are two functions to write or read single line to the opened file. Those functions are **fputs** and **fgets**.

```
#include "stdio.h"
#include "conio.h"
#include "process.h"
```

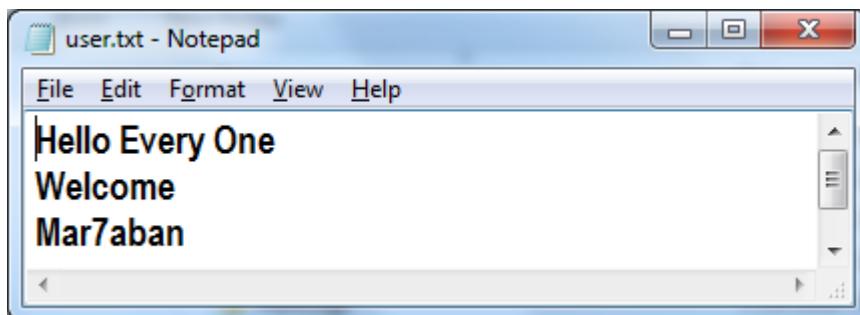
```
#include "string.h"

void main()
{
    FILE *pFile;
    char string[81];
    char ch;

    while(1)
    {
        printf("To write file press w\nTo read from file
               press r\nTo exit press x or other keys.");
        ch = getch();
        printf("\n-----\n");
        if(ch=='w')
        {
            pFile = fopen("c:\\user.txt", "w");
            while(strlen(gets(string))>0)
            {
                fputs(string, pFile);
                fputs("\n", pFile);
            }
            fclose(pFile);
        }
        else if(ch=='r')
        {
            pFile = fopen("c:\\user.txt", "r");
            while(fgets(string, 80, pFile)!=NULL)
            {
                printf("%s", string);
            }
            fclose(pFile);
        }
        else
        {
            exit(0);
        }
        printf("\n-----\n");
    }
}
```



The file “**user.txt**” should be a normal text file. Use any existing text editor like “Notepad, WordPad” to open it.



### **Example 9: Writing and Reading Formatted Strings and Numbers**

It is required to write the sport men records (id name height) in a text file. The program also should read the file back record by record and print it to screen.

There are two functions to write or read formatted data to the opened file. Those functions are **fprintf** and **fscanf**. They work exactly like printf and scanf function except that they deal with files instead of screen and keyboard.

```
#include "stdio.h"
#include "conio.h"
#include "process.h"
#include "string.h"

void main()
{
    FILE *pFile;
    char name[40];
    int id;
```

```
float height;
char ch;

while(1)
{
    printf("To write file press w\nTo read from file
           press r\nTo exit press x or other keys.");
    ch = getch();
    printf("\n-----\n");
    if(ch=='w')
    {
        pFile = fopen("c:\\records.txt", "w");
        while(1)
        {
            printf("Type (id name height):");
            scanf("%d %s %f", &id, name, &height);
            if(id<=0)break;
            fprintf(pFile, "%d %s %f ",
                    id, name, height);
        }
        fclose(pFile);
    }
    else if(ch=='r')
    {
        pFile = fopen("c:\\records.txt", "r");
        while(fscanf(pFile, "%d %s %f ",
                     &id, name, &height) != EOF)
        {
            printf("\n%03d %s %.2f", id, name, height);
        }
        fclose(pFile);
    }
    else
    {
        exit(0);
    }
    printf("\n-----\n");
}
```

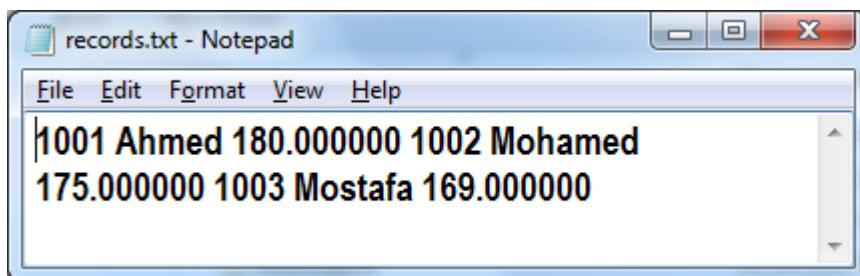
```
C:\Windows\system32\cmd.exe
To write file press w
To read from file press r
To exit press x or other keys.

Type <id name height>:1001 Ahmed 180
Type <id name height>:1002 Mohamed 175
Type <id name height>:1003 Mostafa 169
Type <id name height>:0 0 0

-----
To write file press w
To read from file press r
To exit press x or other keys.

1001 Ahmed 180.00
1002 Mohamed 175.00
1003 Mostafa 169.00
-----
```

The file “**records.txt**” should be a normal text file. Use any text editor to open it.



### Example 10: Writing and Reading Binary Data

It is required to write students records (name age height) in a binary file. The program also should read the file back record by record and print it to screen.

There are two functions to write or read formatted data to the opened file. Those functions are **fwrite** and **fread**.

**fwrite(datapointer, datasize, numberofrecords, filepointer)** is a function that writes any binary data to file. It takes 4 parameters:

**fread(datapointer, datasize, numberofrecords, filepointer)** is a function that reads any binary data from file. It takes 4 parameters:

**datapointer**: pointer to the buffer/array or the variable to be written.  
**datasize**: the size of the buffer/array or the variable.  
**numberofrecords**: number of records and usually set to 1.  
**filepointer**: file pointer.

```
#include "stdio.h"
```

```

#include "conio.h"
#include "process.h"
#include "string.h"

void main()
{
FILE *pFile;
char name[40];
int age;
float height;
char ch;

while(1)
{
    printf("To write file press w\nTo read from file
           press r\nTo exit press x or other keys.");
    ch = getch();
    printf("\n-----\n");
    if(ch=='w')
    {
        pFile = fopen("c:\\students.bin", "wb");
        while(1)
        {
            printf("Enter Student Name Age Height:");
            scanf("%s %d %f", name, &age, &height);
            if(age==0)break;
            fwrite(name, sizeof(name), 1, pFile);
            fwrite(&age, sizeof(age), 1, pFile);
            fwrite(&height, sizeof(height), 1, pFile);
        }
        fclose(pFile);
    }
    else if(ch=='r')
    {
        pFile = fopen("c:\\students.bin", "rb");
        while(1)
        {
            if(fread(name, sizeof(name), 1, pFile)==0)
                break;
            if(fread(&age, sizeof(age), 1, pFile)==0)
                break;
            if(fread(&height, sizeof(height),
                     1, pFile)==0)
                break;
            printf("%s %d %f\n", name, age, height);
        }
        fclose(pFile);
    }
    else
    {
        exit(0);
    }
}

```

```
}  
    printf("\n-----\n");  
}  
}
```

The screenshot shows a Windows command-line window titled "C:\Windows\system32\cmd.exe". The window contains the following text:

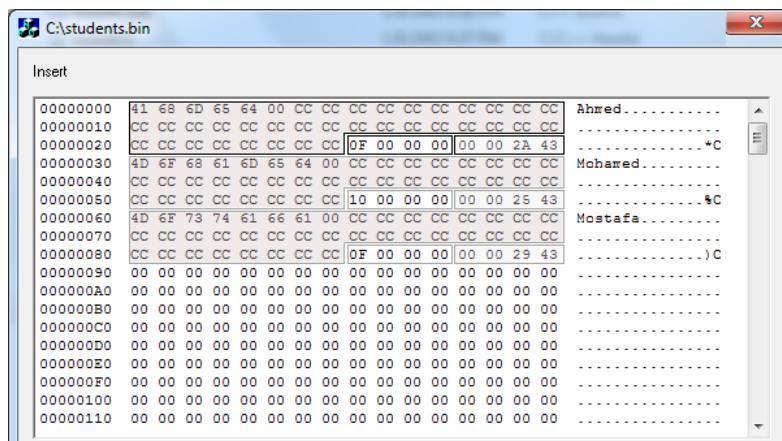
```
To write file press w  
To read from file press r  
To exit press x or other keys.  
  
Type <id name height>:1001 Ahmed 180  
Type <id name height>:1002 Mohamed 175  
Type <id name height>:1003 Mostafa 169  
Type <id name height>:0 0 0  
  
-----  
To write file press w  
To read from file press r  
To exit press x or other keys.  
  
1001 Ahmed 180.00  
1002 Mohamed 175.00  
1003 Mostafa 169.00
```

**fwrite(name, sizeof(name), 1, pFile)** means writing the array (**name**) of size (**sizeof(name)**) to the file.

Alternatively, this line can be replaced by: **fwrite(name, 1, sizeof(name), pFile)** considering that the array record size is (1) and the number of records is (**sizeof(name)**).

**fwrite(&height, sizeof(height), 1, pFile)** means writing the variable (**height**) by providing the variable pointer (**&height**) and size (**sizeof(height)**).

The file “**students.bin**” is a binary file. The contents of the binary files can be displayed using any binary editor as shown in the following figure.



It appears that the file contains three records each record contains the expected three data items (Name (40 byte), age (4 bytes) and height (4 bytes)).

### Example 11: Writing and Reading Structures

Following program writes and reads students data. Agent data is stored in a structure SStudent.

```
#include "stdio.h"
#include "conio.h"
#include "process.h"
#include "string.h"
#include "stdlib.h"

struct SStudent
{
    int m_ID;
    char m_Name[40];
    double m_Height;
};

void main()
{
FILE *pFile;
struct SStudent student;
char ch, temp[20];

while(1)
{
    printf("To write file press w\nTo read from file press r\n"
           "To exit press x or other keys.");
    ch = getch();
    printf("\n-----\n");
    if(ch=='w')
    {
        pFile = fopen("c:\\students.rec", "wb");
        do
        {
            printf("Enter ID: ");
            gets(temp);
            student.m_ID = atoi(temp);
            printf("Enter Name: ");
            gets(student.m_Name);
            printf("Enter Height: ");
            gets(temp);
            student.m_Height = atof(temp);

            fwrite(&student, sizeof(student), 1, pFile);
            printf("Add another agent (y/n)?\n");
        }
        while(getch()=='y');
        fclose(pFile);
    }
}
```

```

else if(ch=='r')
{
    pFile = fopen("c:\\students.rec", "rb");
    while(fread(&student, sizeof(student),
                1, pFile)==1)
    {
        printf("\nID: %03d", student.m_ID);
        printf("\nName: %s", student.m_Name);
        printf("\nHeight: %.2lf\n", student.m_Height);
    }
}
else
{
    exit(0);
}
printf("\n-----\n");
}
}

```

`fwrite(&student, sizeof(student), 1, pFile)` means writing single record to the file by providing student structure pointer (`&student`) and size (`sizeof(student)`).

`fread(&student, sizeof(student), 1, pFile)` means reading single record from file by providing student structure pointer (`&student`) and size (`sizeof(student)`).

### Example 12: Random Access Files

It is required to add advanced read operation to previous example. The advanced read views only the selected student record. The user should supply the index of the selected record.

```

#include "stdio.h"
#include "conio.h"
#include "process.h"
#include "string.h"
#include "stdlib.h"

struct SStudent
{
    int m_ID;
    char m_Name[40];
    double m_Height;
};

void main()
{
FILE *pFile;
struct SStudent student;
char ch, temp[20];
int recno, offset;

```

```

while(1)
{
    printf("To write file press w\nTo read all records press a\n"
          "To read one record press r\nTo exit press x or other keys.");
    ch = getch();
    printf("\n-----\n");
    if(ch=='w')
    {
        pFile = fopen("c:\\students.rec", "wb");
        do
        {
            printf("Enter ID: ");
            gets(temp);
            student.m_ID = atoi(temp);
            printf("Enter Name: ");
            gets(student.m_Name);
            printf("Enter Height: ");
            gets(temp);
            student.m_Height = atof(temp);

            fwrite(&student, sizeof(student), 1, pFile);
            printf("Add another agent (y/n)?\n");
        }
        while(getch()=='y');
        fclose(pFile);
    }
    else if(ch=='a')
    {
        pFile = fopen("c:\\students.rec", "rb");
        while(fread(&student, sizeof(student), 1, pFile)==1)
        {
            printf("\nID: %03d", student.m_ID);
            printf("\nName: %s", student.m_Name);
            printf("\nHeight: %.2lf\n", student.m_Height);
        }
    }
    else if(ch=='r')
    {
        pFile = fopen("c:\\students.rec", "rb");

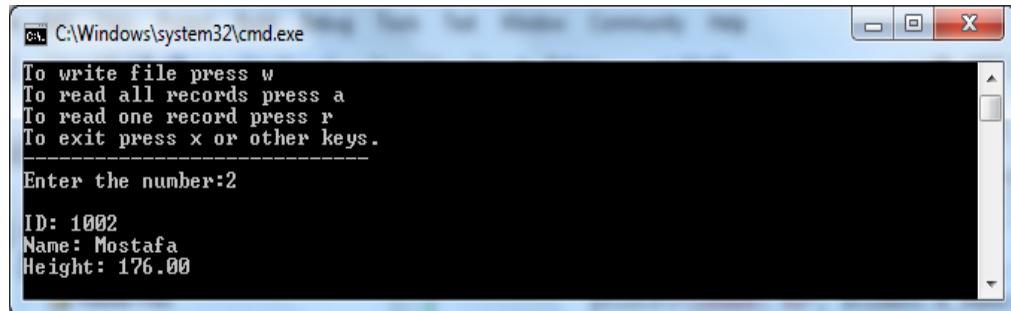
        printf("Enter the number:");
        scanf("%d", &recno);
        offset = (recno-1) * sizeof(student);
        fseek(pFile, offset, SEEK_SET);
        if(fread(&student, sizeof(student), 1, pFile)==1)
        {
            printf("\nID: %03d", student.m_ID);
            printf("\nName: %s", student.m_Name);
            printf("\nHeight: %.2lf\n", student.m_Height);
        }
        else
        {
            printf("\nRecord not found\n");
        }
    }
    else
    {
}
}

```

```

        exit(0);
    }
    printf("\n-----\n");
}
}

```



`fseek(pFile, offset, SEEK_SET)` means moving to the given offset from the beginning of the file (`SEEK_SET`).

### Example 13: Student Database Example

It is required to build a student database program, the program should support following operations:

1. Adding new student
2. Deleting certain student data
3. Listing all students
4. Saving all students data to file
5. Loading students data from file

**Program headers, definition, prototypes and global variables:**

```

#include "stdio.h"
#include "conio.h"
#include "process.h"
#include "string.h"
#include "stdlib.h"

#define TRUE 1

void AddStudent();
void DeleteStudent();
void ViewStudents();
void SaveStudents();
void LoadStudents();
void DeleteAll();

struct SStudent
{

```

```

    int m_ID;
    char m_Name[40];
    float m_Height;
};

struct SStudent *gStudents = NULL;
int gCount = 0;

```

**gStudents** is a global pointers point to all students' records.

**gCount** is a global number indicated the number of student's records.

**Main program:**

```

void main()
{
    while(TRUE)
    {
        printf("\n'a' add new student\n'd' delete
               existing student\n've' list all students");
        printf("\n'w' save to file \n'l' load from
               file \n'q' exit\n");

        switch(getch())
        {
            case 'a': AddStudent(); break;
            case 'd': DeleteStudent(); break;
            case 'v': ViewStudents(); break;
            case 'w': SaveStudents(); break;
            case 'l': LoadStudents(); break;
            case 'q': DeleteAll(); exit(0); break;
            default:
                puts("\nEnter only selection listed.");
        }

        puts("\nPress any key to continue.");
        getch();
        system("cls");
    }
}

```

**system("cls")** invokes a system command "cls", which means clearing the council screen.

**Adding new student:**

```

void AddStudent()
{

```

```

char tempText[40];
gStudents = (struct SStudent*)realloc(gStudents,
                                         (gCount+1) * sizeof(struct SStudent));
if(gStudents==NULL)
{
    printf("\nfailed to allocate more memore");
    return;
}
printf("\nEnter Record %d.\nEnter ID: ", gCount+1);
gets(tempText);
gStudents[gCount].m_ID = atoi(tempText);
printf("Enter name: ");
gets(gStudents[gCount].m_Name);
printf("Enter height: ");
gets(tempText);
gStudents[gCount].m_Height = atof(tempText);
gCount++;
}

```

**Deleting existing student:**

This function searches for the given student ID and delete it. It shift up all following records and overwrite the deleted record, then it re allocate the reserved memory with smaller size.

```

void DeleteStudent()
{
    char tempText[40];
    int i, ID;
    printf("\nEnter Student ID to be Deleted:");
    gets(tempText);
    ID = atoi(tempText);

    for(i=0;i<gCount;i++)
    {
        if(gStudents[i].m_ID==ID)
        {
            memcpy(&gStudents[i], &gStudents[i+1],
                   (gCount-(i+1))*sizeof(struct SStudent));

            gCount--;

            gStudents = (struct SStudent*)
realloc(gStudents, gCount*sizeof(struct SStudent));
            return;
        }
    }

    printf("\nCannot find Student ID.");
}

```

**View students:**

```
void ViewStudents()
{
    int j;
    if(gCount<1)printf("\nEmpty list.\n");
    for(j=0;j<gCount;j++)
    {
        printf("\nRecord number %d\n", j+1);
        printf(" ID: %d\n", gStudents[j].m_ID);
        printf(" Name: %s\n", gStudents[j].m_Name);
        printf(" Height: %4.2f\n", gStudents[j].m_Height);
    }
}
```

**Delete all students:**

```
void DeleteAll()
{
    if(gStudents)free(gStudents);
    gStudents = NULL;
}
```

**Save students' records to a file:**

```
void SaveStudents()
{
    FILE *pFile;

    if(gCount<1){printf("\nEmpty list.\n");return;}
    if((pFile = fopen("agents.rec", "wb"))==NULL)
        {printf("\nCannot open file.\n");return;}
    if(fwrite(gStudents, sizeof(struct SStudent),
              gCount, pFile)!=gCount)
        {printf("\nCannot write to file.\n");return;}
    fclose(pFile);
    printf("\nFile of %d records written.", gCount);
}
```

**Load students' records from a file:**

```
void LoadStudents()
```

```

long file_size;
FILE *pFile;

DeleteAll();

if((pFile = fopen("agents.rec", "rb"))==NULL)
{
    printf("\nCannot open file.\n");
    return;
}

fseek(pFile, 0, SEEK_END);
file_size = ftell(pFile);
fseek(pFile, 0, SEEK_SET);
if((gStudents = (struct SStudent*)malloc(file_size))==NULL)
    {printf("\nCannot allocate memory.\n");return;}

gCount = file_size / sizeof(struct SStudent);
if(fread(gStudents, sizeof(struct SStudent), gCount, pFile)
   !=gCount)
{
    printf("\nCannot read from file.\n");
    return;
}

fclose(pFile);
printf("\nFile read. Total records : %d.", gCount);
}

```

**fseek(pFile, 0, SEEK\_END)** means seeking to the end of the file.

**ftell(pFile)** returns the current position in the file. In this case it returns the file size, because previous step seeks to the end of the file.

The function calculates the number of records inside the file by dividing the measured file size by the single record size (**gCount = file\_size / sizeof(SStudent)**).

## 8.6. Exercise

### Q1.

Write a simple text editor, the program should take the text contents line by line. After entering a complete line the program should add it to an internal buffer. If the user supplies two empty lines, the program should ends the editing and ask the user for the path of the file to save the text.

Hint: you must use a dynamic memory to store the text while editing.

### Q2.

Modify the program in Q1 to provide an initial menu that offer following options:

1. New Text File
2. View Existing Text File
3. Save Text File
4. Exit

### Q3.

Write a program that copy a file to another file, the program should ask for the existing file name and the new file name then performs the copying. The program should inform user with the completion and the spent time.

The program should not copy the contents at once; it should divide it into sections with certain size.

### Q4.

Write a program that navigates through the file system. The program should list the files and folders under any folder. The program should allow used to navigate between folders and drives.

Hint 1: (“..”) means the parent director, (“.”) means the current directory.

Hint 2: (**getcwd**) retrieve the current system directory. (**chdir**) changes the current working directory. (**\_findfirst**, **\_findnext**, **\_findclose**) get all files in a directory. (**\_getdrives**, **\_getdrive**, **\_chdrive**) to list and move between drives.

### Q5.

Considering file copying and directory navigation questions, write a program that copies a complete folder from a location to another.



## Chapter 9: Projects and Libraries

## 9.1. Introduction

Actual programs may consist of hundreds of functions performing different operations and written by different developers. It is applicable in C to distribute the code in different files. Also it is applicable to write external libraries and reuse it in different programs.

This chapter describes how to construct a C project, how to develop static and dynamic C libraries.

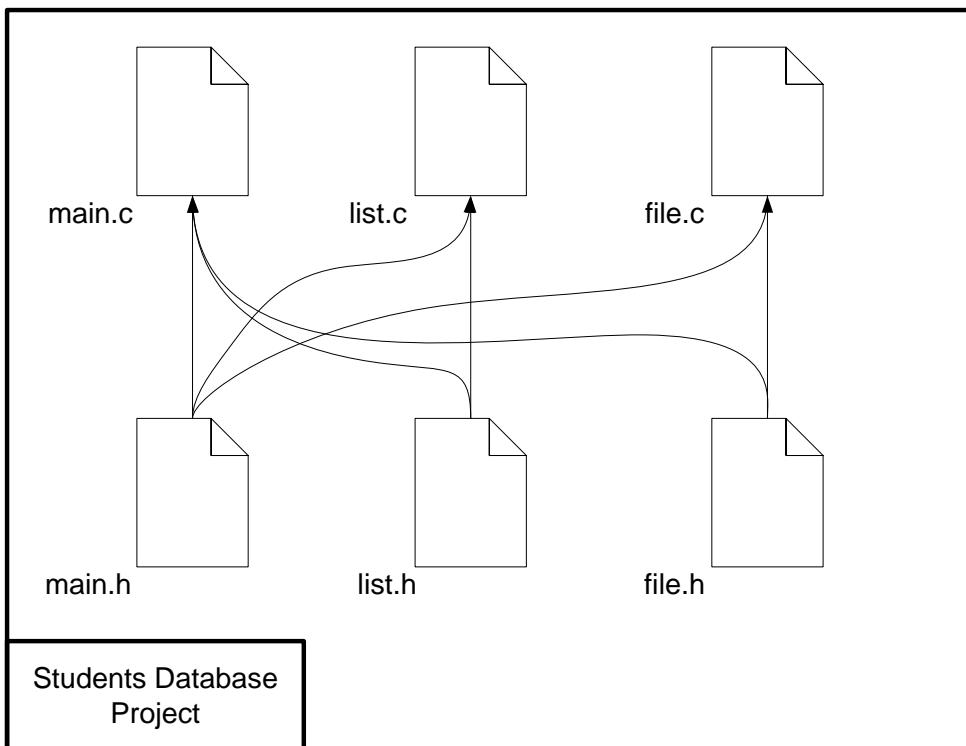
## 9.2. C Project

C Project consists of several code and header files. Code file contains a subset of code providing a set of related services. Header files contain the definitions of common functions and data types that may be used by other code files. Local functions and data types can be defined inside the code files.

Considering “Student Database” example in chapter 8, it is required to divide the code into separate modules each in a separate file, then place the common definition at heard files. The program can be divided into three modules

1. STUDENTS: Main module containing the functions (**main**)
2. LIST: Students list management module containing the functions (**ViewStudents**, **AddStudent**, **DeleteStudent**, **DeleteAll**)
3. FILE: File management module containing the functions (**SaveStudents**, **LoadStudents**)

Each module can have one header file providing their functions prototype. Also it is required to have one common header file for common program definitions.



Above figure shows all required files:

1. Code Files:
  - a. "main.c" providing the functions (**main**) and the global variables.
  - b. "list.c" containing the functions (**ViewStudents**, **AddStudent**, **DeleteStudent**, **DeleteAll**).
  - c. "file.c" providing the functions (**SaveStudents**, **LoadStudents**).
2. Header Files
  - a. "main.h" providing the definition of (**SStudent**) structure, common C headers. This header is used in "main.c" and "list.c" and "file.c".
  - b. "list.h" providing the prototype of the functions (**ViewStudents**, **AddStudent**, **DeleteStudent**). Those functions are used in "main.c".
  - c. "file.h" providing the prototype of the functions (**SaveStudents**, **LoadStudents**). Those functions are used in "main.c".

Finally the contents of each file are listed below:

#### **MAIN Module: main.c**

```
#include "main.h"
#include "list.h"
#include "file.h"
```

```
struct SStudent *gStudents = NULL;
int gCount = 0;

void main()
{
    while(TRUE)
    {
        printf("\n'a' add new student\n'd' delete
               existing student\n've' list all students");
        printf("\n'w' save to file \n'l' load from
               file \n'q' exit\n");

        switch(getch())
        {
            case 'a': AddStudent(); break;
            case 'd': DeleteStudent(); break;
            case 'v': ViewStudents(); break;
            case 'w': SaveStudents(); break;
            case 'l': LoadStudents(); break;
            case 'q': DeleteAll(); exit(0); break;
            default:
                puts("\nEnter only selection listed.");
        }

        puts("\nPress any key to continue.");
        getch();
        system("cls");
    }
}
```

**MAIN Module: main.h**

```
#include "stdio.h"
#include "conio.h"
#include "process.h"
#include "string.h"
#include "stdlib.h"

struct SStudent
{
    int m_ID;
    char m_Name[40];
    float m_Height;
};

#define TRUE 1
extern struct SStudent *gStudents;
extern int gCount;
```

Know that the global variables (**gStudents**, **gCount**) is defined at the beginning of (main.c) and used in (main.c, list.c, file.c). The files (list.c, file.c) does not know if those variables are defined or not.

(**extern struct SStudent \*gStudents**) and (**extern int gCount**) means the global variable (**gStudents**, **gCount**) is defined somewhere in the project. This definition is seen by (list.c and file.c) throw the header file (main.h).

**LIST Module: list.c**

```
#include "main.h"
#include "list.h"

void AddStudent()
{
    char tempText[40];
    gStudents = (struct SStudent*)
        realloc(gStudents,
            (gCount+1) * sizeof(struct SStudent));
    if(gStudents==NULL)
        {printf("\nfailed to allocate more memore"); return;}

    printf("\nRecord %d.\nEnter ID: ", gCount+1);
    gets(tempText);
    gStudents[gCount].m_ID = atoi(tempText);
    printf("Enter name: ");
    gets(gStudents[gCount].m_Name);
    printf("Enter height: ");
    gets(tempText);
```

```

gStudents[gCount].m_Height = atof(tempText);
gCount++;
}

void DeleteStudent()
{
    char tempText[40];
    int i, ID;
    printf("\nEnter Student ID to be Deleted:");
    gets(tempText);
    ID = atoi(tempText);

    for(i=0;i<gCount;i++)
    {
        if(gStudents[i].m_ID==ID)
        {
            memcpy(&gStudents[i], &gStudents[i+1],
                   (gCount-(i+1))*sizeof(struct SStudent));

            gCount--;

            gStudents = (struct SStudent*)
                realloc(gStudents,
                        gCount*sizeof(struct SStudent));
            return;
        }
    }

    printf("\nCannot find Student ID.");
}

void ViewStudents()
{
    int j;
    if(gCount<1)printf("\nEmpty list.\n");
    for(j=0;j<gCount;j++)
    {
        printf("\nRecord number %d\n", j+1);
        printf(" ID: %d\n", gStudents[j].m_ID);
        printf(" Name: %s\n", gStudents[j].m_Name);
        printf(" Height: %4.2f\n", gStudents[j].m_Height);
    }
}

void DeleteAll()
{
    if(gStudents)free(gStudents);
    gStudents = NULL;
}

```

**LIST Module: list.h**

```
void AddStudent();
void DeleteStudent();
void ViewStudents();
void DeleteAll();
```

**FILE Module: file.c**

```
#include "main.h"
#include "file.h"

void SaveStudents()
{
    FILE *pFile;

    if(gCount<1){printf("\nEmpty list.\n");return;}
    if((pFile = fopen("agents.rec", "wb"))==NULL)
        {printf("\nCannot open file.\n");return;}
    if(fwrite(gStudents, sizeof(struct SStudent), gCount,
pFile)!=gCount)
        {printf("\nCannot write to file.\n");return;}
    fclose(pFile);
    printf("\nFile of %d records written.", gCount);
}

void LoadStudents()
{
    long file_size;
    FILE *pFile;

    if(gStudents)
    {
        free(gStudents);
        gStudents = NULL;
    }

    if((pFile = fopen("agents.rec", "rb"))==NULL)
    {
        printf("\nCannot open file.\n");return;
    }
    fseek(pFile, 0, SEEK_END);
    file_size = ftell(pFile);
    fseek(pFile, 0, SEEK_SET);
    if((gStudents = (struct SStudent*)malloc(file_size))==NULL)
    {
        printf("\nCannot allocate memory.\n");return;
    }
    gCount = file_size / sizeof(struct SStudent);
    if(fread(gStudents, sizeof(struct SStudent), gCount,
```

```

        pFile) !=gCount)
{
    printf("\nCannot read from file.\n");return;
}
fclose(pFile);
printf("\nFile read. Total records : %d.", gCount);
}

```

**LIST Module: list.h**

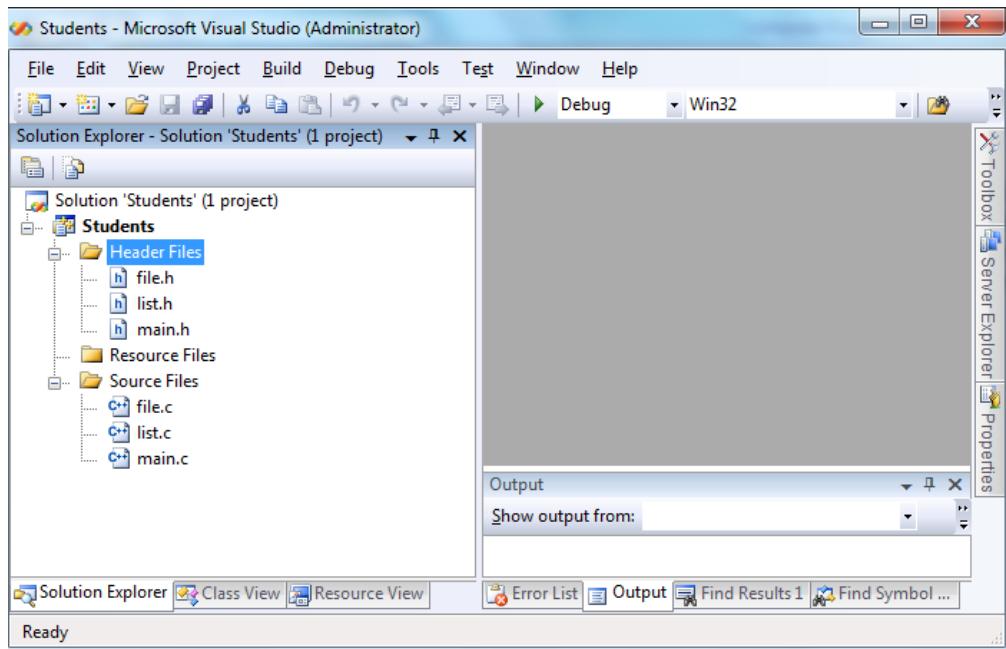
```

void SaveStudents();
void LoadStudents();

```

**Example 1: Student Database Project using Visual Studio 2008**

1. From menu create new project
2. Choose Visual C++ / Win32 / Win32 Consol Application
3. Set the project name to “Students”
4. Select “Consol Application” and “Empty Project” options
5. Add the Code files (main.c, list.c, file.c) to the source files
6. Add the Header files (main.h, list.h, file.h) to the header files
7. All code and header files must be inside the project folder
8. Build and run the project



### 9.3. Libraries

If some program modules are extensively used and can be used by other programmers. It is recommended to place those module in a separate files (Code and Header), and provide them to be integrated inside other projects.

For example if you develop and test a sort function. It is recommended to place it in a files “sort.c” and “sort.h”. Those files can be used later in your future programs or you may provide them to other programmers.

Sometimes it is not accepted to provide the open source code. Sometimes it is required to hide the C code in order to protect the copyrights of the owner and to prevent others from making changes. To make this you must convert your code to a dynamic or static library.

Library is a compiled code which can be completely used without disclosing the code behind. There are two types of libraries, static and dynamic.

Static library can be integrated inside any program. Final program does not need any additional files after the compilation because the static library is embedded inside the final application executable file. Static libraries are provided as (.lib) file.

Dynamic library can be integrated with any program. Final program must have the library file beside the final application executable file. Dynamic libraries are provided as (.dll) file.

	Open Source	Static Library	Dynamic Library
Code State	Open, Readable	Binary, Unreadable	Binary, Unreadable
Modification	Applicable	Not Applicable	Not Applicable
Reverse Engineering	Very Easy	Very Hard	Very Hard
Distribution Files	sort.c sort.h	sort.lib sort.h	sort.dll sort.h <i>optional abstract</i> sort.lib

To demonstrate the library idea, let us build a library that provides a very common service, the Sort function. Define the Sort function and place it into Code and Header files.

“sort.c”

```
#include "sort.h"

void Sort(float* values, int nValues)
{
    float tempValue;
    int i,j;

    for(i=0;i<nValues;i++)
        for(j=0;j<nValues;j++)
        {
            if(values[i]>values[j])
                {
```

```

    {
        tempValue = values[i];
        values[i] = values[j];
        values[j] = tempValue;
    }
}

```

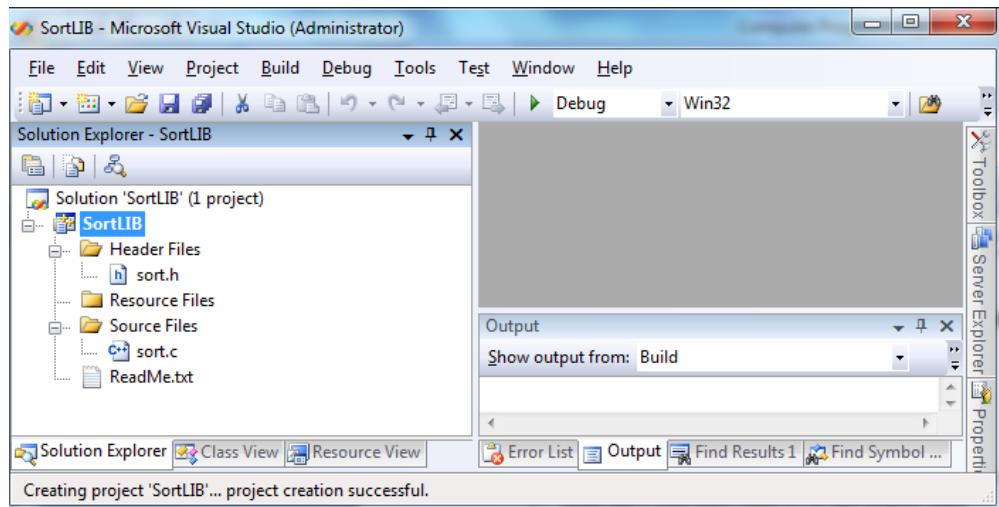
“sort.h”

```
void Sort(float* values, int nValues);
```

## Example 2: Building and Using Static Library using Visual Studio 2008

First: Building the Library

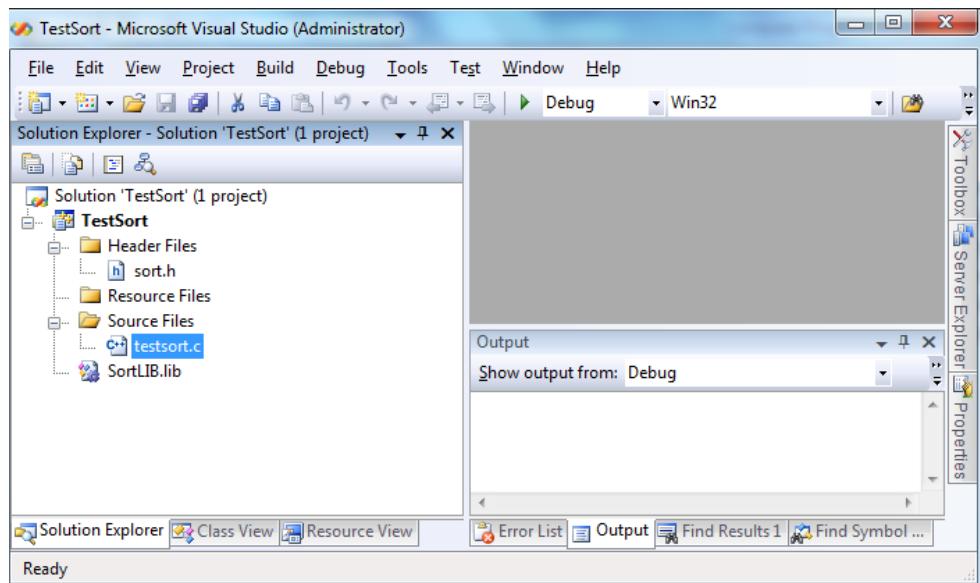
1. From menu create new project
2. Choose Visual C++ / Win32 / Win32 Project
3. Set the project name to “SortLIB”
4. Select “Static Library” option (and uncheck All other options).
5. Add the Code files (sort.c) to the source files
6. Add the Header files (sort.h) to the header files
7. All code and header files must be inside the project folder
8. Build the project
9. You will get a file (sortlib.lib) in the project folder



Second: Using the Library

1. From menu create new project
2. Choose Visual C++ / Win32 / Win32 Consol Application

3. Set the project name to “TestSort”
4. Select “Consol Application” and “Empty Project” options
5. Copy the file (sortlib.lib and sort.h) inside the project folder
6. Add the library files to the project
7. Add the Code files (testsort.c) to the source files
8. All code, header and libraries files must be inside the project folder
9. Modify the (testsort.c) as shown below
10. Build and run the project
11. You will get a file (testsort.exe) in the project folder; this file is the final application.



“testsort.c”

```
#include "stdio.h"
#include "sort.h"

void main()
{
    int i;
    float values[] = {86.2, 18.7, 36.1, 72.8,
                      64.8, 72.3, 16.8, 74.3};

    Sort(values, sizeof(values)/sizeof(float));

    for(i=0;i<sizeof(values)/sizeof(float);i++)
    {
        printf("%2.1f\n", values[i]);
    }
}
```

### Example 3: Building and Using Dynamic Library using Visual Studio 2008

First: Building the Library

1. From menu create new project
2. Choose Visual C++ / Win32 / Win32 Project
3. Set the project name to “SortDLL”
4. Select “DLL” and “Empty Project” options.
5. Add the Code files (sort.c) to the source files
6. Add the Header files (sort.h) to the header files
7. Modify the (sort.h and sort.cpp) as shown below
8. All code and header files must be inside the project folder
9. Build the project
10. You will get a file (sortdll.lib, sortdll.dll) in the project folder

“sort.c”

```
#include "sort.h"

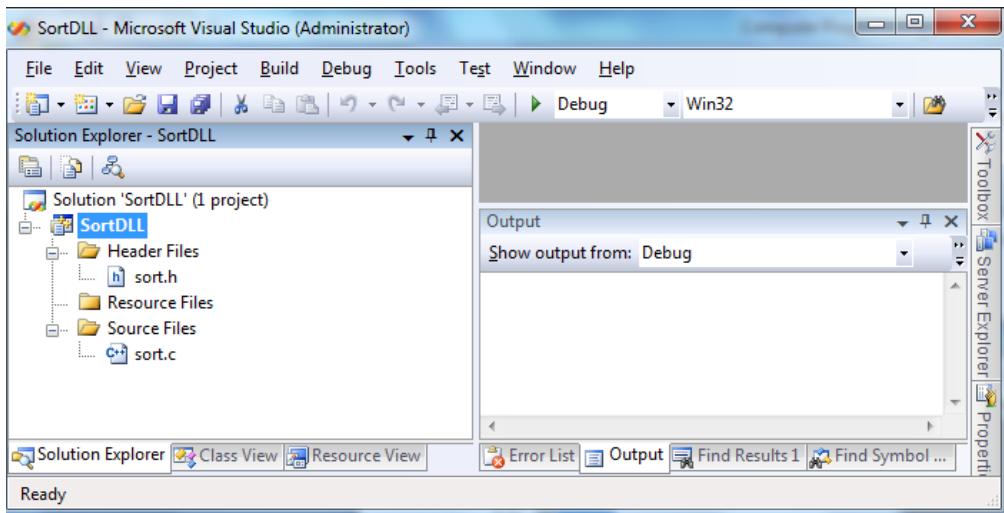
extern "C" __declspec(dllexport) void Sort(float* values, int nValues)
{
    float tempValue;
    int i,j;

    for(i=0;i<nValues;i++)
        for(j=0;j<nValues;j++)
    {
        if(values[i]>values[j])
        {
            tempValue = values[i];
            values[i] = values[j];
            values[j] = tempValue;
        }
    }
}
```

“sort.h”

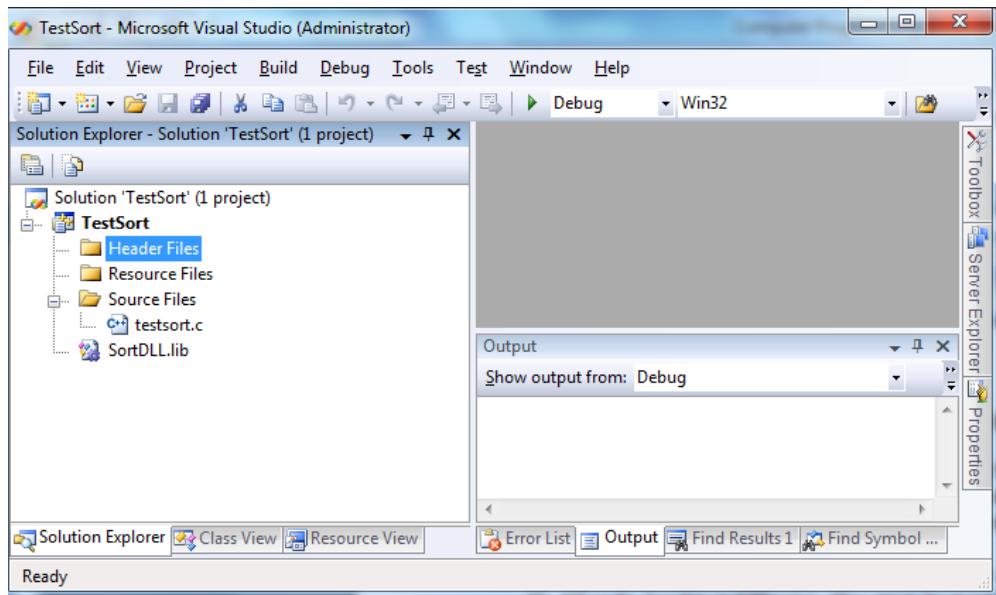
```
extern "C" __declspec(dllexport) void Sort(float* values, int nValues);
```

**\_\_declspec(dllexport)** means is a directive to indicate that the function will be exported outside the DLL, and will be available for external use.



## Second: Using the Library

12. From menu create new project
13. Choose Visual C++ / Win32 / Win32 Consol Application
14. Set the project name to “TestSort”
15. Select “Consol Application” and “Empty Project” options
16. Copy the file (sortdll.lib) inside the project folder
17. Copy the file (sortdll.dll) inside the project folder beside the “testsort.exe”
18. Add the library files (sortdll.lib) to the project
19. Add the Code files (testsort.c) to the source files
20. All code, header and libraries files must be inside the project folder
21. Modify the (testsort.c) as shown below
22. Build and run the project
23. You will get a file (testsort.exe) in the project folder; this file is the final application.



"testsort.c"

```
#include "stdio.h"

extern "C" __declspec(dllexport) void Sort(float* values, int nValues);

void main()
{
    int i;
    float values[] = {86.2, 18.7, 36.1, 72.8,
                      64.8, 72.3, 16.8, 74.3};

    Sort(values, sizeof(values)/sizeof(float));

    for(i=0;i<sizeof(values)/sizeof(float);i++)
    {
        printf("%2.1f\n", values[i]);
    }
}
```

**\_\_declspec(dllexport)** is a directive to indicate that the function will be imported from an external DLL and not defined inside the program.

## 9.4. Command Line

Command line is parameters that can be passed before executing the program; normally you can provide the command from the command line window.

### Example 4: Printing Default Parameters of Command Line

```
#include "stdio.h"

void main(int argc, char *argv[])
{
    printf("%d\n", argc);
    printf("%s\n", argv[0]);
}
```



(`int argc`) is the number of command line parameters.

(`char *argv[]`) is an array of strings contain all supplied commands.

By default the number of command line parameters is 1 which is the executable file path as shown in the output window.

### Example 5: Command Line Sort

Following program will sort all given numeric values and print it on screen.

```
#include "stdio.h"
#include "stdlib.h"

void main(int argc, char *argv[])
{
    int i, j, n;
    double* values;
    double temp;

    n = argc - 1;

    if(n==0)
    {
        printf("There is no values\n");
        exit(0);
    }
```

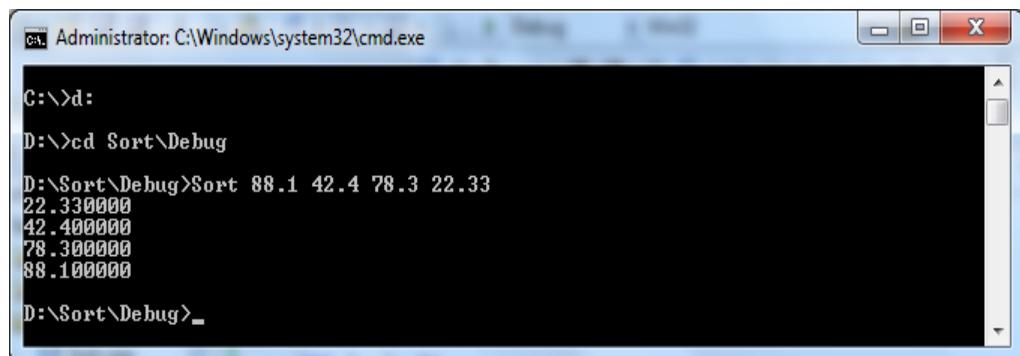
```
values = (double*)malloc(n*sizeof(double));

for(i=0;i<n;i++)
    values[i] = atof(argv[i+1]);

for(i=0;i<n-1;i++)
    for(j=i+1;j<n;j++)
    {
        if(values[i]>values[j])
        {
            temp = values[i];
            values[i] = values[j];
            values[j] = temp;
        }
    }

for(i=0;i<n;i++)
{
    printf("%lf\n", values[i]);
}
}
```

To try above program, use “cmd.exe” utility and move to the folder where your executable file “Sort.exe” is located. Then run the command as shown in the following figure.



## **9.5. Exercise**

### **Q1.**

Write a program that performs folder copy. The program should use command line to get the source and the destination paths. Alternatively, if the command line is empty, the program should ask user to supply the required paths.

### **Q2.**

Write static and dynamic library that provide following mathematical operations on complex numbers:

1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Amplitude
6. Phase



## Chapter 10: Solving Engineering Problems, Part 2

## 10.1. Introduction

This chapter introduces more engineering problems and provides appropriate solutions using C language. Following problems are introduced:

- Advanced Students Database using Linked Lists
- Matrix Algebra
- Finding roots using Newton Raphson method
- Numerical Differentiation
- Numerical Integration
- Solving Differential Equations

## 10.2. Dynamic Linked Lists

### 10.2.1. Problem Statement

Consider Students Database program, it appears that the program uses (`realloc`) when adding or deleting student member. Using (`realloc`) may solve the problem, especially if the structure size and the number of records are small.

Actually (`realloc`) function

1. Creates a new buffer with the new size
2. Copies the original contents
3. Deletes the original buffer
4. Returns the address of the new buffer

Consider a complicated SStudent structure containing all student information and his courses degrees as shown below:

```
struct SDate
{
    int m_Day;
    int m_Month;
    int m_Year;
};

struct SStudent
{
    char m_Name[256];
    char m_Description[8192];
    SDate m_BirthDate;
    double m_Degrees[10];
    double m_TotalDegrees;
};
```

Above structure size is 8548 byte, if it is required to build a program that supports up to 10,000 student. This means adding extra student will cost transferring following data size inside the computer:

$$10000 * 8548 = 85,480,000 \text{ Byte}$$

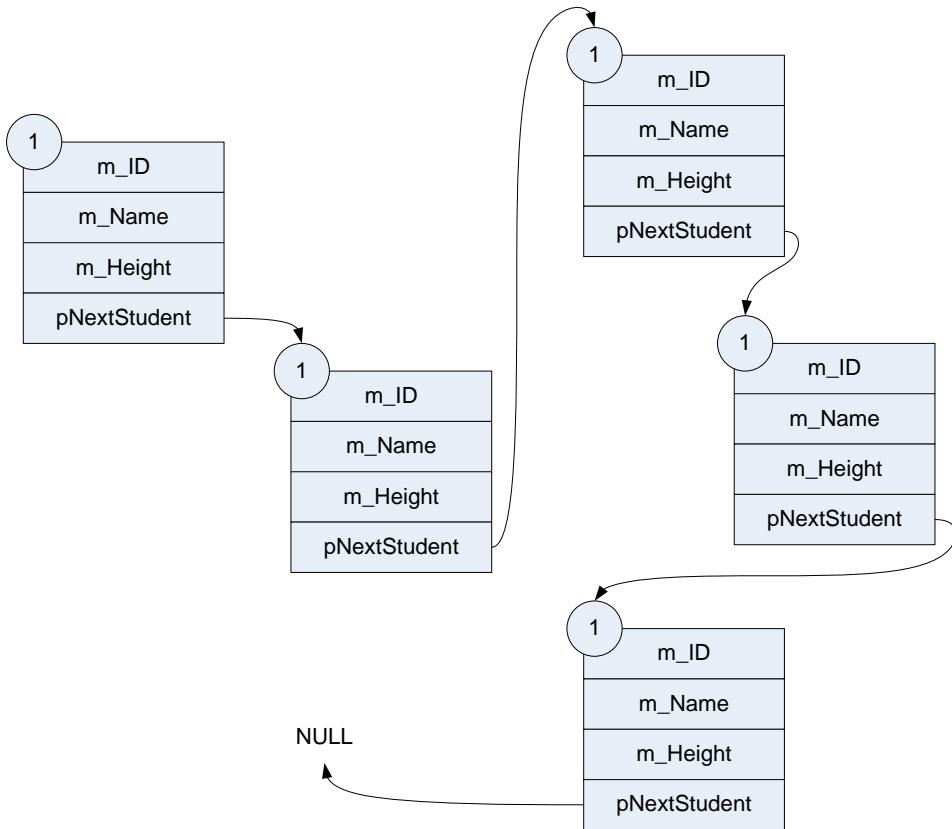
If it is required to transfere 1 byte 1 micosecond the above addition operation will take 85 second or 1.5 minute. This time is very long.

### 10.2.2. Understanding the Solution

Another technique is used, it depends on storing student information in a separte buffers and linking between them using a pointers. This technique called the Linked List method.

Assument the new structure SStudent after adding the member (**SStudent \*pNextStudent**). **pNextStudent** is a pointer containing the address of the next member of the list. Last member of the last have equals **pNextStudent** NULL.

```
struct SStudent
{
    int m_ID;
    char m_Name[40];
    float m_Height;
    SStudent *pNextStudent;
};
```



### 10.2.3. Write the Program

At the beginning of the program only it is required to have one empty pointer, indicating that there is no students added.

```
struct SStudent *gpFirstStudent = NULL;
```

To add a new record to the list:

1. Create New Record
  - a. If the list is empty
    - i. Create new record
    - ii. Assign **gpFirstStudent** to it
  - b. If the list contains records
    - i. Navigate until reaching the last record
    - ii. Create new record
    - iii. Assign the **pNextStudent** of the last record to it
2. Fill the record data
3. Set the next record to NULL

Following code implements the **AddStudent** function:

```

void AddStudent()
{
    struct SStudent* pLastStudent;
    struct SStudent* pNewStudent;
    char tempText[40];

    if(gpFirstStudent==NULL) //If the list is empty
    {
        //Create the first record
        pNewStudent =
            (struct SStudent*)malloc(sizeof(struct SStudent));
        //Assign gpFirstStudent to it
        gpFirstStudent = pNewStudent;
    }
    else //If the list contains records
    {
        // Navigate until reaching the last record
        pLastStudent = gpFirstStudent;
        while(pLastStudent->pNextStudent)
            pLastStudent = pLastStudent->pNextStudent;

        // Create new record and assign the pNextStudent
        // of the last record to it
        pNewStudent =
            (struct SStudent*)malloc(sizeof(struct SStudent));
        //Assign the pNextStudent of the last record to it
        pLastStudent->pNextStudent = pNewStudent;
    }

    //Fill the record data

    printf("\nEnter ID: ");
    gets(tempText);
    pNewStudent->m_ID = atoi(tempText);

    printf("Enter name: ");
    gets(pNewStudent->m_Name);

    printf("Enter height: ");
    gets(tempText);
    pNewStudent->m_Height = atof(tempText);

    //Set the next record to NULL
    pNewStudent->pNextStudent = NULL;
}

```

To delete certain record to the list:

1. Get selected record ID from user
2. Loop on all records starting from **gpFirstStudent**
3. For each record
  - a. Compare the record ID with selected ID
  - b. If it is the selected record
    - i. If previous record exit set the **pNextStudent** of the previous record to point to the next record
    - ii. If previous record not exit sets the **gpFirstStudent** to point to the next record
    - iii. Delete the record

Following code implements the **AddStudent** function:

```
void DeleteStudent()
{
    char tempText[40];
    int i, selectedID;

    //Get selected record ID from user
    printf("\nEnter Student ID to be Deleted:");
    gets(tempText);
    selectedID = atoi(tempText);

    if(gpFirstStudent)
    {
        struct SStudent* pPreviousStudent = NULL;
        struct SStudent* pSelectedStudent = gpFirstStudent;
        //Loop on all records starting from gpFirstStudent
        while(pSelectedStudent)
        {
            //Compare the record ID with selected ID
            if(pSelectedStudent->m_ID==selectedID)
            {
                if(pPreviousStudent)
                    pPreviousStudent->pNextStudent =
                    pSelectedStudent->pNextStudent;
                else
                    gpFirstStudent =
                    pSelectedStudent->pNextStudent;

                free(pSelectedStudent);
                return;
            }

            //Store previous record pointer
            pPreviousStudent = pSelectedStudent;
            pSelectedStudent =
                pSelectedStudent->pNextStudent;
        }
    }

    printf("\nCannot find Student ID.");
}
```

```
}
```

Following code implements the **ViewStudents** function:

```
void ViewStudents()
{
    int count = 0;
    struct SStudent* pCurrentStudent = gpFirstStudent;
    if(gpFirstStudent==NULL)printf("\nEmpty list.\n");

    while(pCurrentStudent)
    {
        printf("\nRecord number %d\n", count+1);
        printf(" ID: %d\n", pCurrentStudent->m_ID);
        printf(" Name: %s\n", pCurrentStudent->m_Name);
        printf(" Height: %4.2f\n", pCurrentStudent->m_Height);
        pCurrentStudent = pCurrentStudent->pNextStudent;
        count++;
    }
}
```

Following code implements the **DeleteAll** function:

```
void DeleteAll()
{
    struct SStudent* pCurrentStudent = gpFirstStudent;
    if(gpFirstStudent==NULL)printf("\nEmpty list.\n");

    while(pCurrentStudent)
    {
        struct SStudent* pTempStudent = pCurrentStudent;
        pCurrentStudent = pCurrentStudent->pNextStudent;
        free(pTempStudent);
    }
    gpFirstStudent = NULL;
}
```

Following code implements the **SaveStudents** function:

```
void SaveStudents()
{
    int count = 0;
    FILE *pFile;
    struct SStudent* pCurrentStudent = gpFirstStudent;

    if(gpFirstStudent==NULL)printf("\nEmpty list.\n");
```

```

if((pFile = fopen("agents.rec", "wb"))==NULL)
    {printf("\nCannot open file.\n");return;}

while(pCurrentStudent)
{
    fwrite(pCurrentStudent,
           sizeof(struct SStudent), 1, pFile);
    pCurrentStudent = pCurrentStudent->pNextStudent;
    count++;
}
fclose(pFile);

printf("\nFile of %d records written.", count);
}

```

Following code implements the **LoadStudents** function:

```

void LoadStudents()
{
    int i;
    int count;
    long fileSize;
    FILE *pFile;
    struct SStudent* pCurrentStudent;
    struct SStudent* pPreviousStudent = NULL;

    DeleteAll();

    if((pFile = fopen("agents.rec", "rb"))==NULL)
        {printf("\nCannot open file.\n");return;}
    fseek(pFile, 0, SEEK_END);
    fileSize = ftell(pFile);
    fseek(pFile, 0, SEEK_SET);
    count = fileSize / sizeof(struct SStudent);

    for(i=0;i<count;i++)
    {
        pCurrentStudent =
            (struct SStudent*)malloc(sizeof(struct SStudent));

        if(i==0)
            gpFirstStudent = pCurrentStudent;
        else
            pPreviousStudent->pNextStudent = pCurrentStudent;

        fread(pCurrentStudent, sizeof(struct SStudent),
              1, pFile);
        pCurrentStudent->pNextStudent = NULL;
        pPreviousStudent = pCurrentStudent;
    }
}

```

```

    fclose(pFile);
    printf("\nFile read. Total records : %d.", count);
}

```

And finally the main function:

```

void main()
{
    while(TRUE)
    {
        printf("\n'a' add new student\n'd' delete
               existing student\n've' list all students");
        printf("\n'w' save to file \n'l' load from
               file \n'q' exit\n");

        switch(getch())
        {
            case 'a': AddStudent(); break;
            case 'd': DeleteStudent(); break;
            case 'v': ViewStudents(); break;
            case 'w': SaveStudents(); break;
            case 'l': LoadStudents(); break;
            case 'q': DeleteAll(); exit(0); break;
            default:
                puts("\nEnter only selection listed.");
        }

        puts("\nPress any key to continue.");
        getch();
        system("cls");
    }
}

```

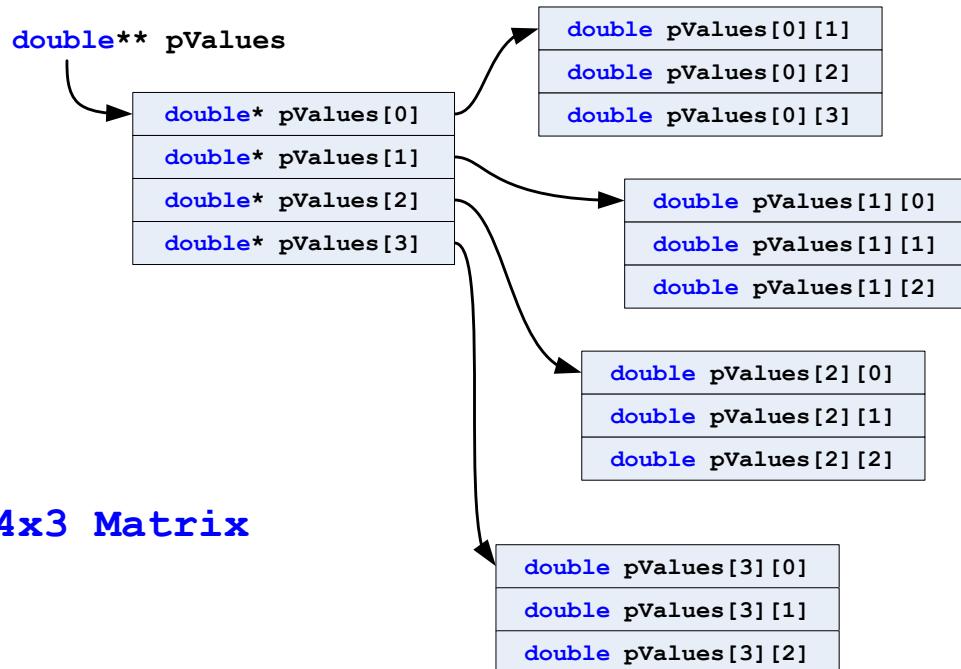
## 10.3. Matrix Algebra

### 10.3.1. Problem Statement

In chapter 5 the matrix implemented using 2 dimension array with static size. It is required to define a dynamic matrix that can support any size and implement operations like addition, subtraction, multiplication and transposing.

### 10.3.2. Understand the Solution

Double pointers or (pointer to pointer) can be used to define a dynamic two dimension arrays. The matrix values can be represented with a pointer to pointer as shown in the following figure:



### 10.3.3. Write the Program

The matrix is defined using following structure to place the values and the dimension information together.

```
struct SMatrix
{
    double** pValues; //Pointer to pointer to matrix values
    int nR;           //Number of rows
    int nC;           //Number of columns
};
```

It is required to define following functions:

1. **CreateMatrix**: Create the matrix by providing the dimension and the an initial filling type. Following filling types will be supported (None, zeros, ones, identity, random)
2. **DestroyMatrix**: Destroy all allocated memory for the given matrix
3. **PrintMatrix**: Prints matrix contents as rows and columns

4. **MADD**: Add two matrices and return the result in a new matrix
5. **MSUB**: Subtract two matrices and return the result in a new matrix
6. **MMUL**: Multiplicate two matrices and return the result in a new matrix
7. **MTRS**: Transpose the given matrices and return the result in a new matrix

**CreateMatrix:**

```

enum      MFILLTYPE{MFT_NONE,           MFT_ZEORS,          MFT_ONES,          MFT_RAND,
MFT_IDENTITY};

struct SMatrix* CreateMatrix(int nR, int nC, enum MFILLTYPE fillType)
{
    int r, c;
    struct SMatrix* pRM;

    pRM = (struct SMatrix*)malloc(sizeof(struct SMatrix));

    pRM->nR = nR;
    pRM->nC = nC;

    //Allocate the matrix, the matrix is an array
    //of pointer to rows pointers
    pRM->pValues = (double**)malloc(nR * sizeof(double*));

    //Allocate the rows, each row will point to all row elements
    for(r=0;r<nR;r++)
        pRM->pValues[r] = (double*)malloc(nC * sizeof(double));

    if(fillType!=MFT_NONE)
    {
        for(r=0;r<nR;r++)
            for(c=0;c<nC;c++)
            {
                switch(fillType)
                {
                    case MFT_ZEORS:
                        pRM->pValues[r][c] = 0; break;
                    case MFT_ONES:
                        pRM->pValues[r][c] = 1; break;
                    case MFT_RAND:
                        pRM->pValues[r][c] = (rand()%100)/100.0;
                        break;
                    case MFT_IDENTITY:
                        pRM->pValues[r][c] = (r==c)?1:0; break;
                }
            }
    }

    return pRM;
}

```

**DestroyMatrix:**

```
void DestroyMatrix(struct SMatrix* pMatrix)
```

```
{
    int r;

    //delete each row pointer
    for(r=0;r<pMatrix->nR;r++)
        free(pMatrix->pValues[r]);

    //delete the matrix
    free(pMatrix->pValues);

    free(pMatrix);
}
```

**PrintMatrix:**

```
void PrintMatrix(struct SMatrix* pM)
{
    int r, c;

    for(r=0;r<pM->nR;r++)
    {
        for(c=0;c<pM->nC;c++)
            printf("%5.2lf\t", pM->pValues[r][c]);

        printf("\n");
    }
}
```

**MADD:**

```
struct SMatrix* MADD(struct SMatrix* pM1, struct SMatrix* pM2)
{
    int r, c;
    struct SMatrix* pRM;

    if(pM1->nC!=pM2->nC || pM1->nR!=pM2->nR)
        {printf("matrix error");exit(0);}
    pRM = CreateMatrix(pM1->nR, pM1->nC, MFT_NONE);

    for(r=0;r<pRM->nR;r++)
    {
        for(c=0;c<pRM->nC;c++)
            pRM->pValues[r][c] =
                pM1->pValues[r][c] + pM2->pValues[r][c];
    }

    return pRM;
}
```

## MSUB:

```

struct SMatrix* MSUB(struct SMatrix* pM1, struct SMatrix* pM2)
{
    int r, c;
    struct SMatrix* pRM;

    if(pM1->nC!=pM2->nC || pM1->nR!=pM2->nR)
        {printf("matrix error");exit(0);}
    pRM = CreateMatrix(pM1->nR, pM1->nC, MFT_NONE);

    for(r=0;r<pRM->nR;r++)
    {
        for(c=0;c<pRM->nC;c++)
            pRM->pValues[r][c] =
                pM1->pValues[r][c] - pM2->pValues[r][c];
    }

    return pRM;
}

```

## MMUL:

```

struct SMatrix* MMUL(struct SMatrix* pM1, struct SMatrix* pM2)
{
    int r, c, i;
    struct SMatrix* pRM;

    if(pM1->nC!=pM2->nR) {printf("matrix error");exit(0);}
    pRM = CreateMatrix(pM1->nR, pM2->nC, MFT_NONE);

    for(r=0;r<pRM->nR;r++)
    {
        for(c=0;c<pRM->nC;c++)
        {
            pRM->pValues[r][c] = 0;
            for(i=0;i<pM1->nC;i++)
            {
                pRM->pValues[r][c] +=
                    pM1->pValues[r][i] * pM2->pValues[i][c];
            }
        }
    }

    return pRM;
}

```

## MTRS:

```

struct SMatrix* MTRS(struct SMatrix* pM)
{

```

```

int r, c;
struct SMatrix* pRM;

pRM = CreateMatrix(pM->nC, pM->nR, MFT_NONE);

for(r=0;r<pRM->nR;r++)
{
    for(c=0;c<pRM->nC;c++)
    {
        pRM->pValues[r][c] = pM->pValues[c][r];
    }
}

return pRM;
}

```

**Main:**

```

void main()
{
    int r, c, value = 0;
    struct SMatrix *pM1, *pM2, *pM3, *pRM;

    pM1 = CreateMatrix(4, 4, MFT_RAND);
    pM2 = CreateMatrix(4, 4, MFT_RAND);
    pM3 = CreateMatrix(4, 7, MFT_RAND);

    printf("\nM1:\n"); PrintMatrix(pM1);
    printf("\nM2:\n"); PrintMatrix(pM2);
    printf("\nM3:\n"); PrintMatrix(pM3);

    pRM = MADD(pM1, pM2);
    printf("\nM3 = M1 + M2:\n");
    PrintMatrix(pRM); DestroyMatrix(pRM);

    pRM = MSUB(pM1, pM2);
    printf("\nM3 = M1 - M2:\n");
    PrintMatrix(pRM); DestroyMatrix(pRM);

    pRM = MMUL(pM1, pM3);
    printf("\nM3 = M1 * M2:\n");
    PrintMatrix(pRM); DestroyMatrix(pRM);

    pRM = MTRS(pM3);
    printf("\nM3 = Transpose of M1:\n");
    PrintMatrix(pRM); DestroyMatrix(pRM);

    DestroyMatrix(pM1);
    DestroyMatrix(pM2);
    DestroyMatrix(pM3);
}

```

{}

## 10.4. Finding Roots

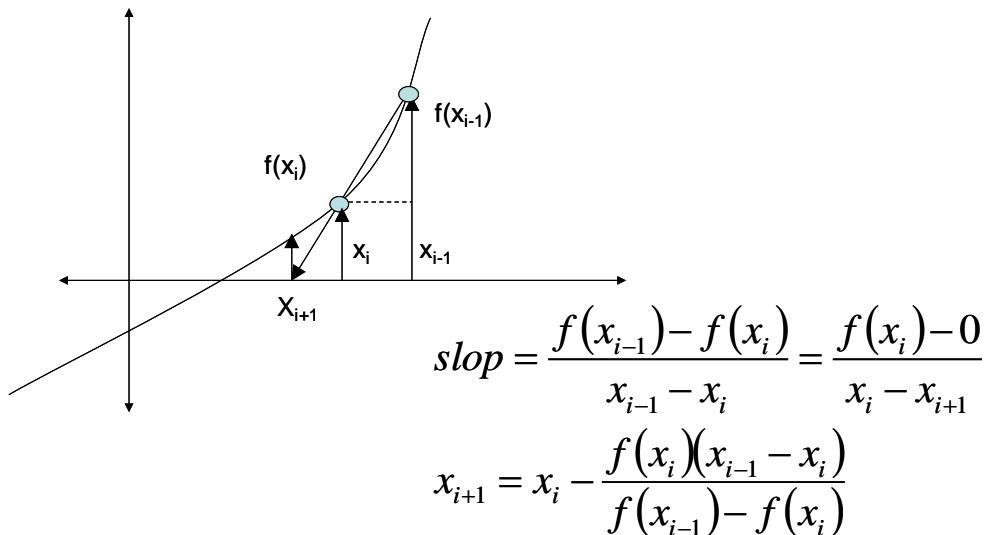
### 10.4.1. Problem Statement

It is required to finds the first real root of any given equations.

### 10.4.2. Understanding the Solution

Secant method can be used to find a real root of any given equation.

Using any two initial values  $x_{i-1}$  and  $x_i$  a new value  $x_{i+1}$  can be derived as shown in the following figures:



The algorithm can be summarized in the following steps:

STEP 1: Choose any  $x_{i-1}$  and  $x_i$

STEP 2: If  $(f(x_{i-1}) - f(x_i)) = 0$  Then Exit With Error

STEP 3: Calculate  $x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$

STEP 4: Calculate  $\varepsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| \cdot 100\%$

If  $\varepsilon_a > \varepsilon_s$  Then Goto STEP 2

If  $\varepsilon_a \leq \varepsilon_s$  Then Root =  $x_{i+1}$  (Terminate)

Important: During the iterations if  $f(x_{i-1}) - f(x_i)$  equals zero, this means no root can be found with this initial value or maybe there is no roots at all.

#### 10.4.3. Write the Program

```
#include "stdio.h"
#include "math.h"

double f(double x)
{
    return (x*x-x-2)/x;
}

double Secant(double x1, double x2, double es, int maxtrials)
{
    double px = x1, ppx = x2, x;

    for(int i = 0; i<maxtrials; i++)
    {
        x = px - f(px) * (ppx-px) / (f(ppx)-f(px));

        if(i>0)
        {
            double ea = fabs( (x-px)/x )*100;
            if(ea<=es)  return x;
        }

        ppx = px;
        px = x;
    }
    return x;
}

void main()
{
```

```

        printf("the root is %lf ", Secant(2.5, 3, 0.01, 100));
}

```

**Secant** function provide the first root for the specific equation  $\frac{x^2-x-2}{x}$ .

It is required to write another **Secant()** function that can find the first root of any given function. This simply can be made by providing a pointer to the mathematical function instead of calling certain function.

```

#include "stdio.h"
#include "math.h"

double f(double x)
{
    return (x*x-x-2)/x;
}

double Secant(double x1, double x2, double es, int maxtrials,
double (*pf)(double))
{
    double px = x1, ppx = x2, x;

    for(int i = 0; i<maxtrials; i++)
    {
        x = px - pf(px) * (ppx-px) / (pf(ppx)-pf(px));

        if(i>0)
        {
            double ea = fabs( (x-px)/x )*100;
            if(ea<=es)   return x;
        }

        ppx = px;
        px = x;
    }
    return x;
}

void main()
{
    printf("the root is %lf ", Secant(2.5, 3, 0.01, 100,f));
}

```

## 10.5. Advanced Root Finding Solutions

### 10.5.1. Complex Roots

Secant method can be extended to find complex roots. This can be achieved by initializing  $x_0$  with complex value and supporting complex calculation instead of real calculations.

### 10.5.2. Finding All Roots

Secant method can be extended to find all roots.

For any function  $f(x)$ :

Step 1: Find a root using Secant Method

Step 2: If the root is founded store it and proceed to the next step, otherwise go to Step 5

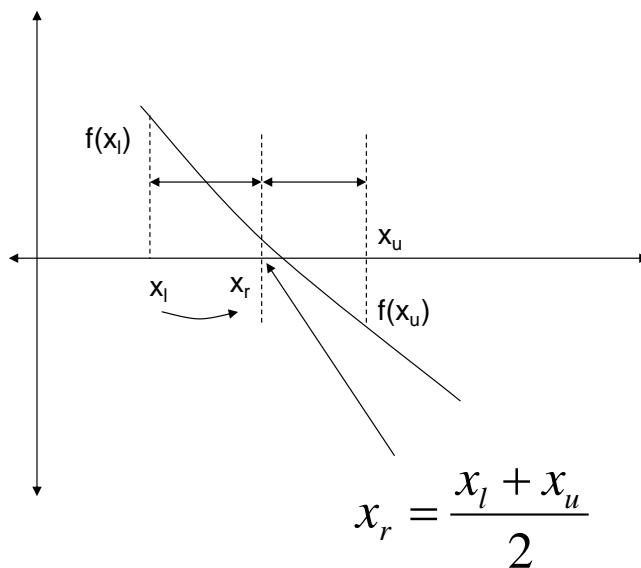
Step 3: Calculate  $\frac{f(x)}{(x-x_0)}$  to exclude the founded root and decrease the system order

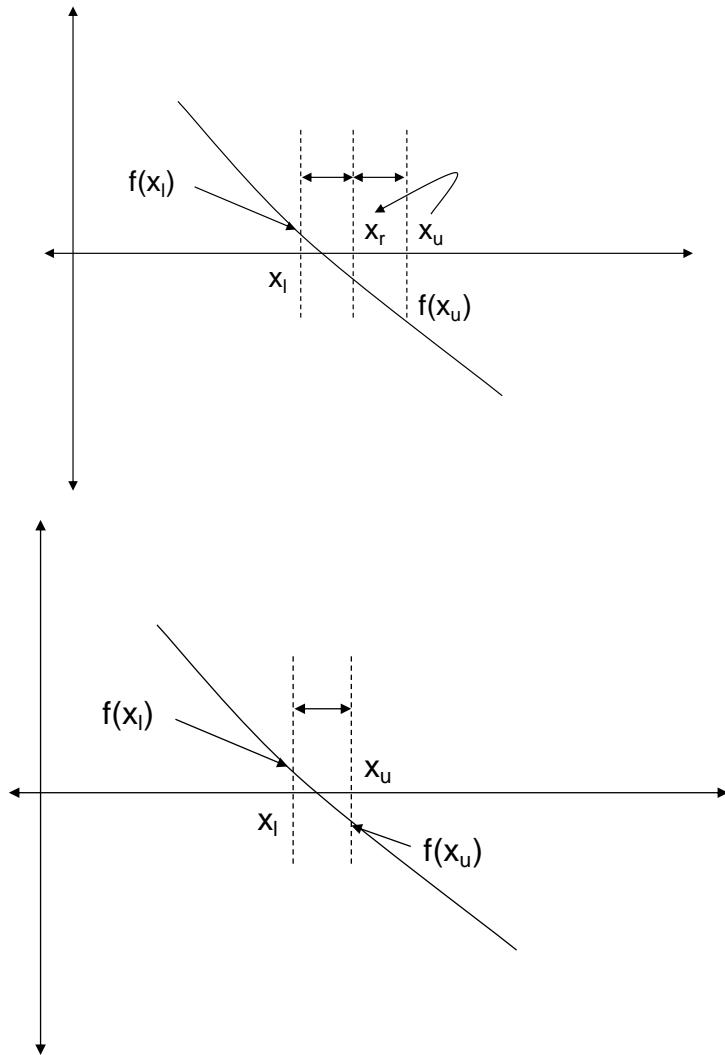
Step 4: Go to Step 1

Step 5: Print all founded roots if any

### 10.5.3. Bisection Method

Using two initial values  $x_l$  and  $x_u$  (where  $(x_l) * f(x_u) < 0$ ) a new value  $x_r$  can be derived as shown in the following figures:





The algorithm can be summarized in the following steps:

STEP 1: Choose  $x_l$  and  $x_u$  where  $f(x_l) \cdot f(x_u) < 0$

STEP 2: Calculate  $x_r = \frac{x_l + x_u}{2}$

Calculate  $f(x_r)$

STEP 3: If  $f(x_l) \cdot f(x_r) < 0$  Then Set  $x_u = x_r$

If  $f(x_l) \cdot f(x_r) > 0$  Then Set  $x_l = x_r$

If  $f(x_r) = 0$  Then Root =  $x_r$  (Terminate)

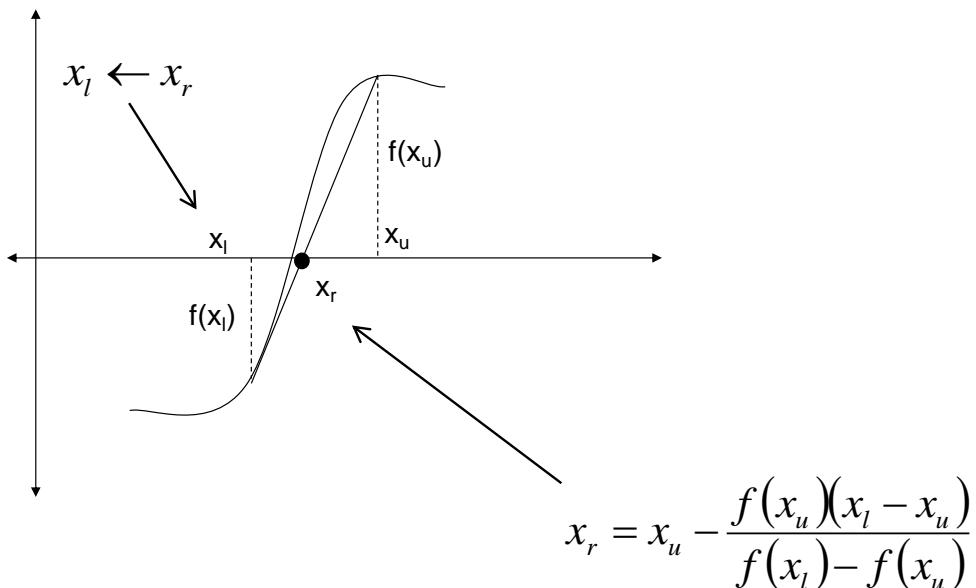
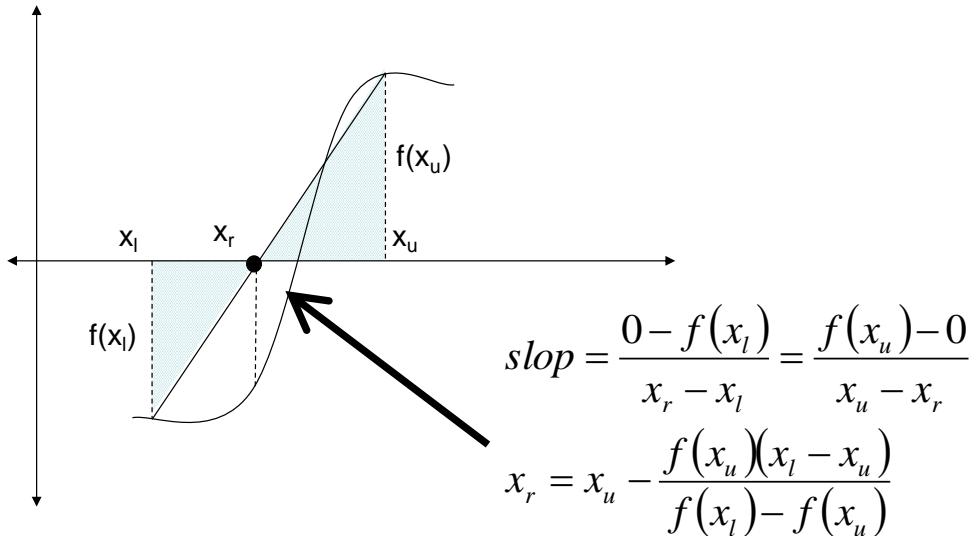
STEP 4: Calculate  $\varepsilon_a = \left| \frac{x_r^{new} - x_r}{x_r^{new}} \right| \cdot 100\%$

If  $\varepsilon_a > \varepsilon_s$  Then Goto STEP 2

If  $\varepsilon_a \leq \varepsilon_s$  Then Root =  $x_r$  (Terminate)

#### 10.5.4. False Position Method

Using two initial values  $x_l$  and  $x_u$  (where  $(x_l) * f(x_u) < 0$ ) a new value  $x_r$  can be derived as shown in the following figures:



The algorithm can be summarized in the following steps:

STEP 1: Choose  $x_l$  and  $x_r$  where  $f(x_l) \cdot f(x_u) < 0$

STEP 2: Calculate  $x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$

Calculate  $f(x_r)$

STEP 3: If  $f(x_l) \cdot f(x_r) < 0$  Then Set  $x_u = x_r$

If  $f(x_l) \cdot f(x_r) > 0$  Then Set  $x_l = x_r$

If  $f(x_r) = 0$  Then Root =  $x_r$  (Terminate)

STEP 4: Calculate  $\varepsilon_a = \left| \frac{x_r^{new} - x_r}{x_r^{new}} \right| \cdot 100\%$

If  $\varepsilon_a > \varepsilon_s$  Then Goto STEP 2

If  $\varepsilon_a \leq \varepsilon_s$  Then Root =  $x_r$  (Terminate)

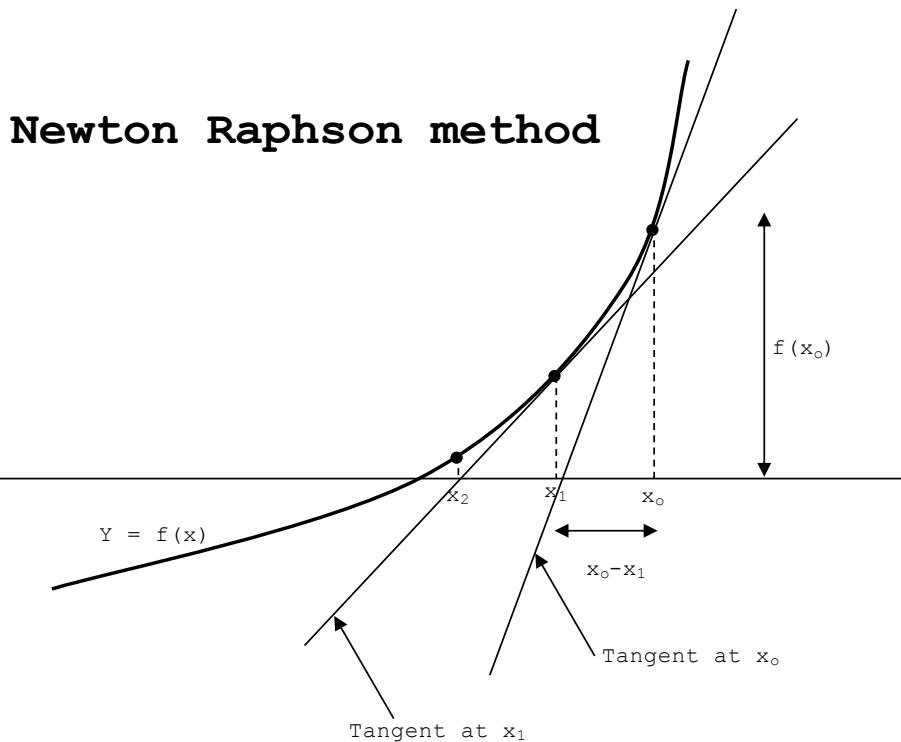
#### 10.5.5. Newton Raphson Method

For any function  $f(x)$  the derivative at  $x_o$  is  $f'(x_o) = \frac{f(x_o)}{(x_o - x_1)}$  and generally at  $x_{n+1}$

$f'(x_n) = \frac{f(x_n)}{(x_{n+1} - x_n)}$  which lead to:

Newton Raphson root finding formula  $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

This formula can be used by suggesting any initial  $x_o$  then find  $x_1$  and repeat the operation to get  $x_2, x_3 \dots$  until  $x_n \approx x_{n+1}$ . As shown in the following figure if  $(x_n \approx x_{n+1})$  this value will represent the founded root.



The algorithm can be summarized in the following steps:

STEP 1: Starting with any  $x_i$

STEP 2: If  $f'(x_i) = 0$  Then Exit With Error

STEP 3: Calculate  $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$

STEP 4: Calculate  $\varepsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| \cdot 100\%$

If  $\varepsilon_a > \varepsilon_s$  Then Goto STEP 2

If  $\varepsilon_a \leq \varepsilon_s$  Then Root =  $x_{i+1}$  (Terminate)

Important: During the iterations if  $f'(x)$  equals zero, this means no root can be found with this initial value or maybe there is no roots at all.

## 10.6. Numerical Differentiation

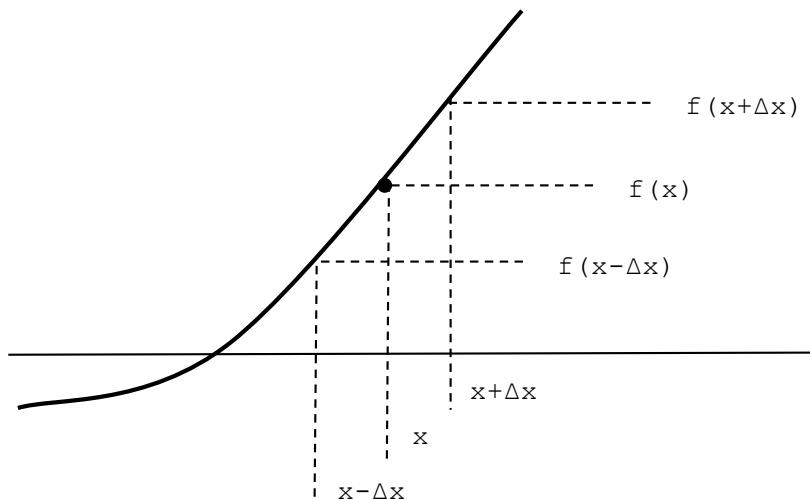
### 10.6.1. Problem Statement

It is required to finds the first derivative of any function  $f(x)$  at certain  $x$ .

### 10.6.2. Understanding the Solution

For any function  $f(x)$  the derivative can be defined as one of the following:

$$f'(x) = \begin{cases} \frac{f(x) - f(x - \Delta x)}{\Delta x} & \text{Backward Derivative} \\ \frac{f(x + \Delta x) - f(x - \Delta x)}{2 \cdot \Delta x} & \text{Center Derivative} \\ \frac{f(x + \Delta x) - f(x)}{\Delta x} & \text{Forward Derivative} \end{cases}$$



$\Delta x$  must be small enough to achieve accurate result, in the other hand if  $\Delta x$  is very small it may increase the numerical errors.

### 10.6.3. Write the Program

```
#include "stdio.h"
#include "math.h"

enum DERIVATIVE_METHOD
{DERIVATIVE_BACKWORD, DERIVATIVE_CENTER, DERIVATIVE_FORWARD};
```

```
double f(double x)
{
    return x*x*x-13*x*x+20*x+100;
}

double Derivative(double x, double dx, double(*pf)(double),
                  enum DERIVATIVE_METHOD method)
{
    double fd;

    switch(method)
    {
        case DERIVATIVE_BACKWORD:
            fd = (pf(x)-pf(x-dx))/dx; break;
        case DERIVATIVE_CENTER :
            fd = (pf(x+dx)-pf(x-dx)) / (2*dx); break;
        case DERIVATIVE_FORWARD :
            fd = (pf(x+dx)-pf(x))/dx; break;
    }

    return fd;
}

void main()
{
printf("Backwrderivative:%f\n",Derivative(3,0.01,f,DERIVATIVE_BACKWORD));
printf("Center derivative:%f\n",Derivative(3,0.01, f,
DERIVATIVE_CENTER));
printf("Forward derivative:%f\n",Derivative(3,0.01,
f,DERIVATIVE_FORWARD));
}
```

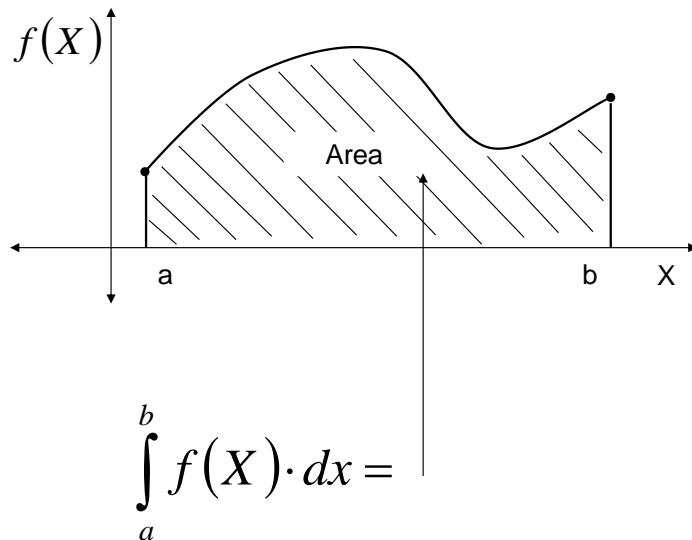
## 10.7. Numerical Integration

### 10.7.1. Problem Statement

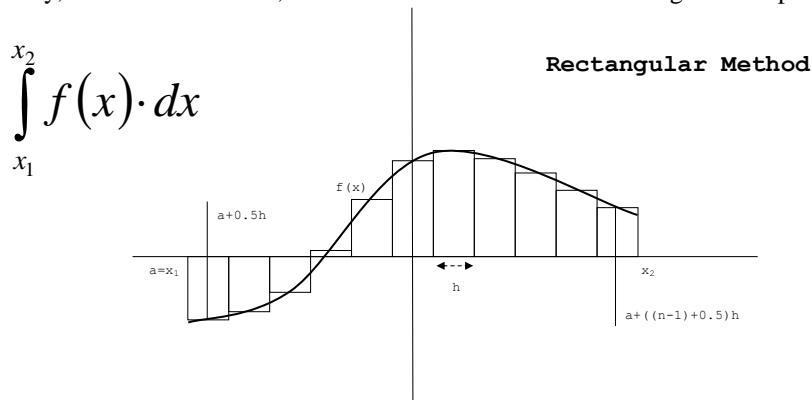
It is required to integrate any given equations at any given range.

### 10.7.2. Understanding the Solution

Integrating a function means calculating the area under the curve.



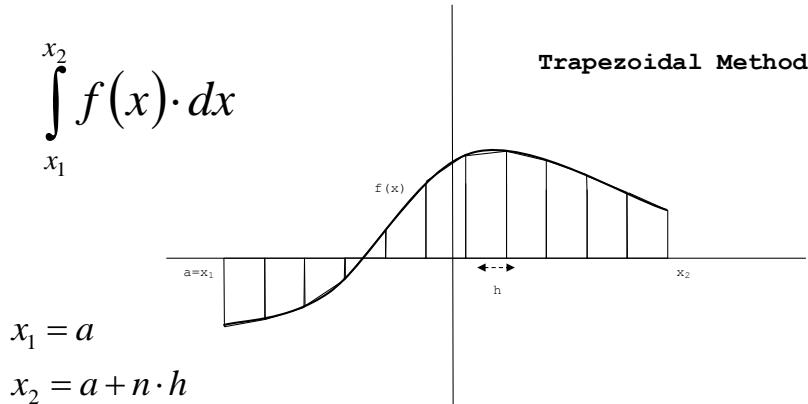
Numerically, to evaluate the area, it can be divided into a set of rectangles or trapezoids.



$$x_1 = a$$

$$x_2 = a + n \cdot h$$

$$\sum_{i=0}^{n-1} f\left(a + \left(i + \frac{1}{2}\right) \cdot h\right) \cdot h$$



$$\sum_{i=1}^n [f(a + (i-1) \cdot h) + f(a + i \cdot h)] \cdot \frac{h}{2}$$

(h) must be small enough to achieve accurate result, in the other hand if (h) is very small it may increase the numerical errors.

#### 10.7.3. Write the Program

```
#include "stdio.h"
#include "math.h"

enum INTEGRATION_METHOD {INTEGRATION_RECTANGULAR,
INTEGRATION_TRAPEZOIDAL};

double f(double x)
{
    return x*x*x-13*x*x+20*x+100;
}

double Integration(double x1, double x2, double h,
double (*pf)(double), enum INTEGRATION_METHOD method)
{
    double I, x;

    if (method==INTEGRATION_RECTANGULAR)
    {
        x = x1;
        I = 0;
        while (x<x2)
        {
            I += f(x) * h;
            x += h;
        }
    }
    else if (method==INTEGRATION_TRAPEZOIDAL)
    {
        x = x1;
        I = f(x);
        while (x<x2)
        {
            I += (f(x) + f(x+h)) * h / 2;
            x += h;
        }
    }
}
```

```
        }
    }
    else if(method==INTEGRATION_TRAPEZOIDAL)
    {
        x = x1;
        I = 0;
        while(x<x2)
        {
            I += (f(x)+f(x+h)) * (h/2);
            x += h;
        }
    }

    return I;
}

void main()
{
    printf("Rectangular integration:%f\n",
    Integration(-5, 5, 0.01, f, INTEGRATION_RECTANGULAR));
    printf("Trapezoidal integration:%f\n",
    Integration(-5, 5, 0.01, f, INTEGRATION_TRAPEZOIDAL));
}
```

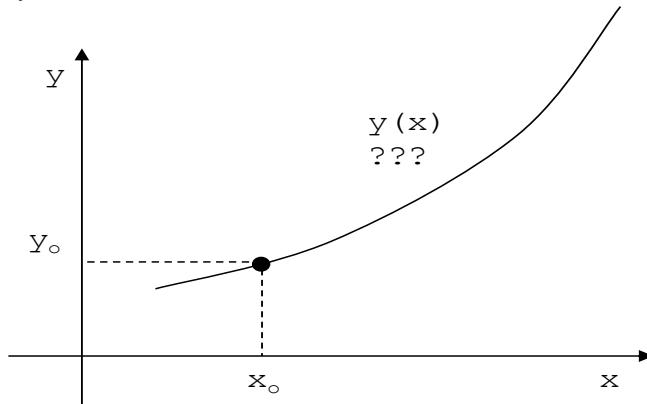
## 10.8. Solving First Order Differential Equations

### 10.8.1. Problem Statement

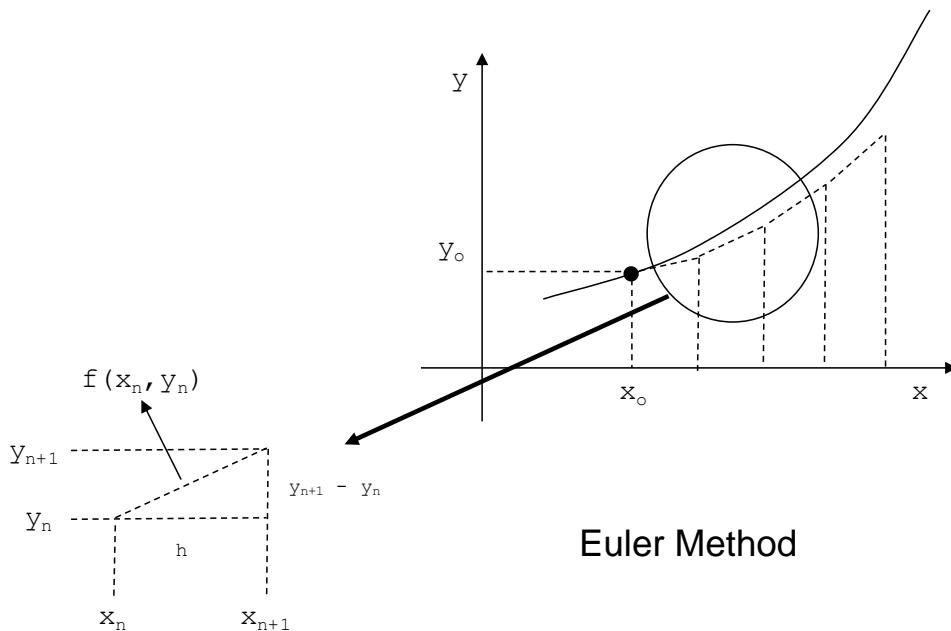
It is required to solve any given first order differential equation.

### 10.8.2. Understanding the Solution

For the first order differential equation  $\frac{dy}{dx} = f(x, y)$  it is required to evaluate  $y(x)$  given the initial value  $(x_0, y_0)$ .



Euler proposes a numerical method to solve any first order differential equation. Considering the following figure:



It appears that:

$$\frac{dy}{dx} = f(x, y)$$

$$\frac{y_{n+1} - y_n}{x_{n+1} - x_n} = f(x_n, y_n)$$

$$y_{n+1} = y_n + (x_{n+1} - x_n) \cdot f(x_n, y_n)$$

$$y_{n+1} = y_n + h \cdot f(x_n, y_n)$$

Euler method state that for any first order equation  $\frac{dy}{dx} = f(x, y)$  and the initial value  $(x_0, y_0)$  following formula can be used to evaluate  $y(x)$   $y_{n+1} = y_n + h \cdot f(x_n, y_n)$

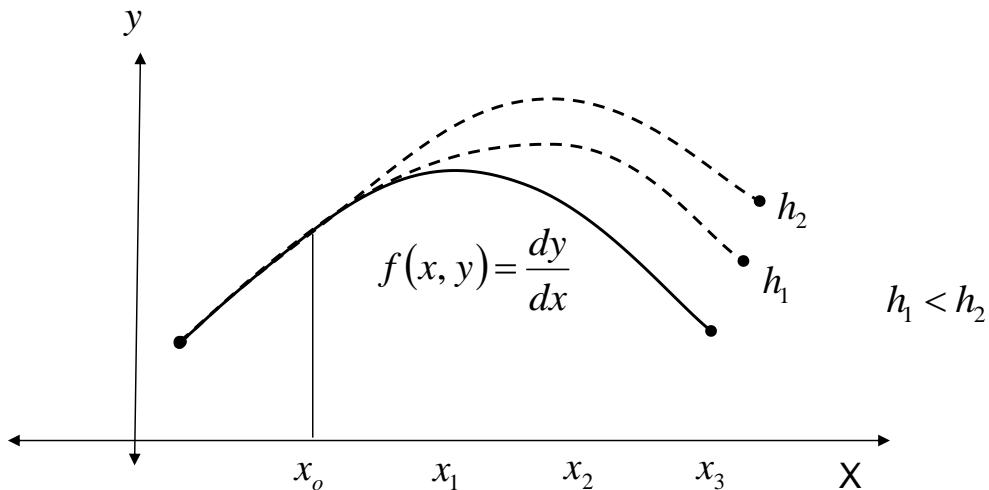
Step1: Using  $(x_0, y_0)$  find  $y_1$

Step2: Using  $(x_1, y_1)$  find  $y_2$

Step3: Using  $(x_2, y_2)$  find  $y_3$

Step4: ...

As the  $(h)$  value decreases as the accuracy of the result increases. In the other hand very small values of  $(h)$  will lead to increase the numerical errors.



Following example demonstrate the idea:

Given the following first order equation and the initial values, find  $y(x)$  analytically and numerically:

$$\frac{dy}{dx} = y - x^2 + 1$$

$$(x_0, y_0) = (0, 0.5)$$

**The Analytical Solution:**

$$\frac{dy}{dx} = y - x^2 + 1$$

$$\frac{dy}{dx} + p(x)y = g(x) \Rightarrow y = \frac{\int \mu(x)g(x)dx + c}{\mu(x)}, \mu(x) = ke^{\int p(x)dx}$$

$$p(x) = -1$$

$$g(x) = 1 - x^2$$

$$\mu(x) = ke^{\int -1 dx} = ke^{-x}$$

$$y = \frac{\int ke^{-x}(1-x^2)dx + c}{ke^{-x}} = \frac{ke^{-x}(x^2 + 2x + 1) + c}{ke^{-x}} = x^2 + 2x + 1 + me^x$$

$$x = 0, y = 0.5 \Rightarrow m = -0.5$$

$$y = x^2 + 2x + 1 - 0.5e^x$$

**The numerical solution:**

By choosing  $h = 0.2$

$$f(x, y) = \frac{dy}{dx} = y - x^2 + 1$$

$$\tilde{y}_{i+1} = y_i + h \cdot f(x_i, y_i) = y_i + 0.2 \cdot f(x_i, y_i)$$

i	X <sub>i</sub>	Y <sub>i</sub> (Numerical)	Y <sub>i</sub> (Actual)	e <sub>t</sub>
0	0	0.5	0.5	0
1	0.2	0.8	0.82929	3.53294
2	0.4	1.152	1.21409	5.11393
3	0.6	1.5504	1.64894	5.97599
4	0.8	1.98848	2.12723	6.52255
5	1	2.45818	2.64086	6.91756
6	1.2	2.94981	3.17994	7.23694
7	1.4	3.45177	3.7324	7.51866
8	1.6	3.95013	4.28348	7.78235
9	1.8	4.42815	4.81518	8.03756
10	2	4.86578	5.30547	8.28743

### Predictor – Corrector Method

$$\frac{dy}{dx} = f(x, y)$$

$$y_{n+1} = y_{n-1} + 2h \cdot f(x_n, y_n)$$

The numerical solution can be made as following:

$$y_1 = y_0 + h \cdot f(x_0, y_0) \text{ Euler}$$

$$y_2 = y_0 + 2h \cdot f(x_1, y_1)$$

$$y_3 = y_1 + 2h \cdot f(x_2, y_2)$$

$$y_4 = y_2 + 2h \cdot f(x_3, y_3)$$

Know that the first step is made using Euler and the later steps is made by Predictor Corrector method.

#### 10.8.3. Write the Program

```
#include "stdio.h"
#include "math.h"

enum ODE_METHOD{ODE_EULER, ODE_PREDICTORCORRECTOR};

double f(double x,double y)
{
    return 4*x*x+2*y+3*x+x*y;
}

void SolveODE(double* xValues, double* yValues, int nValues,
double h, double(*pf)(double,double), enum ODE_METHOD method)
{
    int i;
    double x, y, pPy, py;

    if(method==ODE_EULER)
    {
        x = xValues[0];
        y = yValues[0];

        for(i=0;i<nValues;i++)
        {
            y = y + h * pf(x,y);
        }
    }
    else
    {
        x = xValues[0];
        y = yValues[0];
        pPy = y;
        py = y;

        for(i=1;i<nValues;i++)
        {
            y = pPy + 2*h * pf(x,y);
            pPy = py;
            py = y;
            x = x + h;
        }
    }
}
```

```

        x += h;
        xValues[i] = x;
        yValues[i] = y;
    }
}
else if(method==ODE_PREDICTORCORRECTOR)
{

    x = xValues[0];
    ppy = yValues[0];

    py = ppy + h * pf(x,ppy);
    x += h;
    xValues[1] = x;
    yValues[1] = py;

    for(i=2;i<nValues;i++)
    {
        y = ppy + 2*h * pf(x-h,py);
        x += h;
        ppy = py;
        py = y;
        xValues[i] = x;
        yValues[i] = y;
    }
}
void main()
{
    int i;
    double xValues[100] = {3};
    double yValues[100] = {3};

    SolveODE(xValues, yValues, 100, 0.01, f, ODE_EULER);

    for(i=0;i<100;i++)
    {
        printf("\nEULER(x,y) : (%lf,%lf)",
               xValues[i], yValues[i]);
    }

    SolveODE(xValues, yValues, 100, 0.01,
              f, ODE_PREDICTORCORRECTOR);

    for(i=0;i<100;i++)
    {
        printf("\nPREDICTORCORRECTOR(x,y) : (%lf,%lf)",
               xValues[i], yValues[i]);
    }
}

```

## 10.9. Advanced ODE Solutions

### 10.9.1. Runge-Kutta 3 Method

$$\frac{dy}{dx} = f(x, y)$$

$$k_1 = h \cdot f(x_n, y_n)$$

$$k_2 = h \cdot f(x_n + h/2, y_n + k_1/2)$$

$$k_3 = h \cdot f(x_n + h, y_n - k_1 + 2k_2)$$

$$y_{n+1} = y_n + (k_1 + 4k_2 + k_3)/6$$

The numerical solution can be made as following:

$$k_1 = h \cdot f(x_0, y_0)$$

$$k_2 = h \cdot f(x_0 + h/2, y_0 + k_1/2)$$

$$k_3 = h \cdot f(x_0 + h, y_0 - k_1 + 2k_2)$$

$$y_1 = y_0 + (k_1 + 4k_2 + k_3)/6$$

$$k_1 = h \cdot f(x_1, y_1)$$

$$k_2 = h \cdot f(x_1 + h/2, y_1 + k_1/2)$$

$$k_3 = h \cdot f(x_1 + h, y_1 - k_1 + 2k_2)$$

$$y_2 = y_1 + (k_1 + 4k_2 + k_3)/6$$

$$y_3 = y_2 + (k_1 + 4k_2 + k_3)/6$$

### 10.9.2. Runge-Kutta 4 Method

$$\frac{dy}{dx} = f(x, y)$$

$$k_1 = h \cdot f(x_n, y_n)$$

$$k_2 = h \cdot f(x_n + h/2, y_n + k_1/2)$$

$$k_3 = h \cdot f(x_n + h/2, y_n + k_2/2)$$

$$k_4 = h \cdot f(x_n + h, y_n + k_3)$$

$$y_{n+1} = y_n + (k_1 + 2k_2 + 2k_3 + k_4)/6$$

The numerical solution can be made as following:

$$\begin{aligned}
 k_1 &= h \cdot f(x_0, y_0) \\
 k_2 &= h \cdot f(x_0 + h/2, y_0 + k_1/2) \\
 k_3 &= h \cdot f(x_0 + h/2, y_0 + k_2/2) \\
 k_4 &= h \cdot f(x_0 + h, y_0 + k_3) \\
 y_1 &= y_0 + (k_1 + 2k_2 + 2k_3 + k_4)/6
 \end{aligned}$$

$$\begin{aligned}
 k_1 &= h \cdot f(x_1, y_1) \\
 k_2 &= h \cdot f(x_1 + h/2, y_1 + k_1/2) \\
 k_3 &= h \cdot f(x_1 + h/2, y_1 + k_2/2) \\
 k_4 &= h \cdot f(x_1 + h, y_1 + k_3) \\
 y_2 &= y_1 + (k_1 + 2k_2 + 2k_3 + k_4)/6
 \end{aligned}$$

$$y_3 = y_2 + (k_1 + 2k_2 + 2k_3 + k_4)/6$$

### 10.9.3. Solving Higher Order Differential Equations

Any differential can be re-written as a set of first order differential equations:

$$5y^{(4)} + 3y^{(3)} + 2y^{(2)} + 9y^{(1)} + 2y^{(0)} = 4 \cdot x$$

$$\begin{aligned}
 y_1 &= y^{(0)} \\
 y_1^{(1)} &= y_2 = y^{(1)} \\
 y_2^{(1)} &= y_3 = y^{(2)} \\
 y_3^{(1)} &= y_4 = y^{(3)} \\
 y_4^{(1)} &= y^{(4)} = 4 \cdot x^2 - (3y^{(3)} + 2y^{(2)} + 9y^{(1)} + 2y^{(0)})/5
 \end{aligned}$$

$$\begin{aligned}
 y_1^\bullet &= y_2 \\
 y_2^\bullet &= y_3 \\
 y_3^\bullet &= y_4 \\
 y_4^\bullet &= 4 \cdot x^2 - (3y_4 + 2y_3 + 9y_2 + 2y_1)/5
 \end{aligned}$$

The numerical solution can be applied to all equation separately considering the knowing the initial point ( $x_{(0)}$ ,  $y_{1(0)}$ ,  $y_{2(0)}$ ,  $y_{3(0)}$ ,  $y_{4(0)}$ ).

## 10.10. Numerical Errors

Numerical errors come from different sources:

1. Number Truncation
2. Mathematical Approximations

### 10.10.1. Number Truncation

Small fractions of numerical values are truncated due to the limitation of storage area. For example if only 5 digits are allowed to represent the number and it is required to represent the value 0.988938292. The number will be truncated to 0.98893 and the fraction 0.000008282 is considered as a truncation error. This happens also in binary values because different data types have limited storage area.

Example:

It is required to represent the number 3.25 using binary representation and it is only allowed to use 7 bit for exponent and 7 bit for mantissa.

$$(3.25)_{10} \rightarrow (11.011001100110011)_2 \rightarrow (0000011.0110011)_2 \rightarrow (3.1875)_{10} \rightarrow E_r = 0.0625 = 0.39\%$$

### 10.10.2. Mathematical Approximation

Most of the numerical methods perform some sort of approximation. All illustrated numerical methods uses approximate solutions to simplify the solution.

For example, numerical Integration, calculates the area under the curve by dividing the area into a set of rectangles which may lead to approximate calculations of actual area value.

Another example, the exponent function `exp()` uses approximate for of Taylor expansion to calculate the exponent value.

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \dots$$

Above Taylor expansion for exponent value is infinite; there is no way to evaluate the actual value.

## 10.11. Exercise

### Q1.

For student program, write a function that reverses the arrangement of the list. This means that the last element will be the head and the first element will be the tail.

### Q2.

For student program, write a function that sort the list based on student name.

### Q3.

Write a function that calculates the inverse of any given matrix.

Hint: Use Gauss–Jordan elimination method

### Q3.

Write a program that performs following matrix calculation using complex numbers:

1. Add two complex matrices
2. Subtract two complex matrices
3. Multiply two complex matrices

### Q4.

Write a function that finds the complex root of the function:

$$f(x) = x^2 + 4$$

### Q5.

Write a function that multiplies two polynomials, by taking the order and the coefficients of each. For example:

$$f_1(x) = 3x - 2$$

$$f_2(x) = 2x^2 - 4x + 1$$

$$f_1(x) * f_2(x) = 3x * (2x^2 - 4x + 1) - 2 * (2x^2 - 4x + 1)$$

$$f_1(x) * f_2(x) = 6x^3 - 12x^2 + 3x - 4x^2 + 8x - 2$$

$$f_1(x) * f_2(x) = 6x^3 - 16x^2 + 11x - 2$$

### Q6.

Write a function that finds all roots of any given function. For example:

$$f(x) = x^5 - x^4 + 9x^3 + x^2 + 8x + 4$$

### Q7.

Write a program that evaluates the derivative of following function at x=10:

$$f(x) = x^5 - x^4 + 9x^3 + x^2 + 8x + 4$$

Also it is required to:

1. Try following  $\Delta x$  values (0.1, 0.01, 0.001, 0.0001, 0.00001)
2. Compare the numerical result with the analytical result
3. Comment on the result

**Q8.**

Write a program that integrates following function in the range (0, 10):

$$f(x) = x^5 - x^4 + 9x^3 + x^2 + 8x + 4$$

Also it is required to:

1. Try following h values (0.1, 0.01, 0.001, 0.0001, 0.00001)
2. Compare the numerical result with the analytical result
3. Comment on the result

**Q9.**

Write a program that solves the following first order differential equation, given that  $x_0= 1$ ,  $y_0=7$ :

$$\frac{dy}{dx} = f(x, y) = \frac{y + x}{x}$$

Also it is required to:

1. Try following h values (0.1, 0.01, 0.001, 0.0001, 0.00001)
2. Compare the numerical result with the analytical result
3. Comment on the result

Hint: The analytic solution of this equation with given initial conditions is:

$$y(x) = x \ln |x| + 7x$$

**Q10.**

Write a program that solves the following differential equation:

$$\begin{aligned} x^2 y'' + 7xy' + 8y &= 0 \\ y'_0 &= 1, y_0 = 1, x_0 = 1 \end{aligned}$$

Also it is required to:

1. Try following h values (0.1, 0.01, 0.001, 0.0001, 0.00001)
2. Compare the numerical result with the analytical result
3. Comment on the result

Hint: The analytic solution of this equation is:

$$y = \frac{c_1}{2}x^{-2} + c_2x^{-4}$$

**Q11. (Report)**

Write a function that divides a polynomial to other polynomial, the function should produce the division result and reminder polynomials. Use this function to modify Q6.

## Chapter 11: GUI/GDI Programming for Windows

## 11.1. Basic GUI/GDI Programming

GUI stands for Graphical User Interface. GUI programming means building visual programs that interact with user using windows, buttons, text boxes, menus, trees, lists....

GDI stands for Graphical Device Interface. GDI programming means drawing shapes and strings to the connected graphical device like screens and printers.

C language does not provide built in GUI/GDI libraries. In each operating system there is different GUI/GDI libraries, for example Windows operating system uses Windows APIs Libraries, Linux operating system uses Cairo/GTK/QT GUI Libraries. This book will use GUI/GDI libraries in Windows environment.

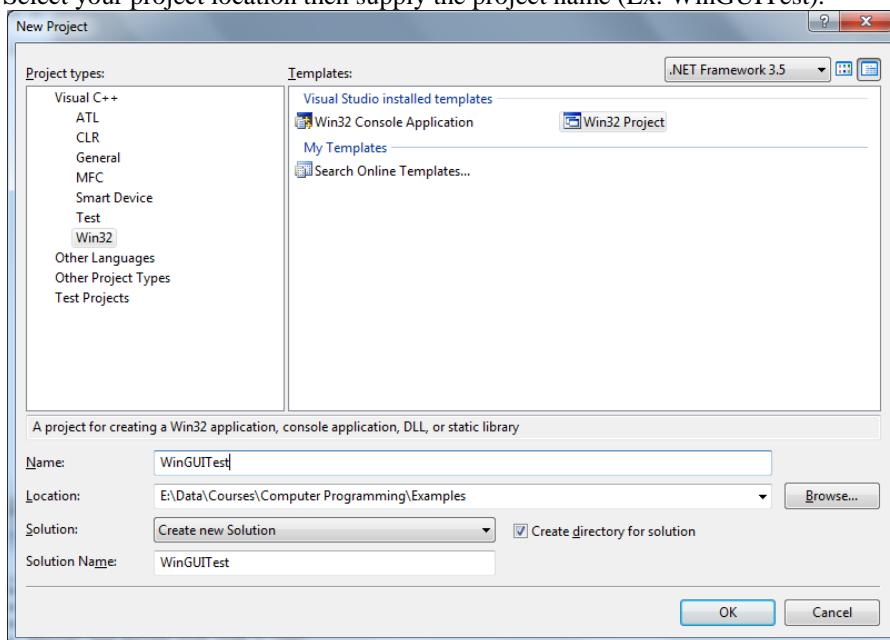
Windows GUI/GDI libraries consist of a hundreds of functions called APIs (Application Programming Interface).

Using GUI/GDI libraries requires expert knowledge of C language, for that reason this book introduces a very little wrapper “wingui.h” to simplify the GUI/GDI programming. The reader can understand the implementation of this wrapper starting from chapter 8. All defined routines in the “wingui.h” is started with WG letters (EX: WGCreateMainWindow).

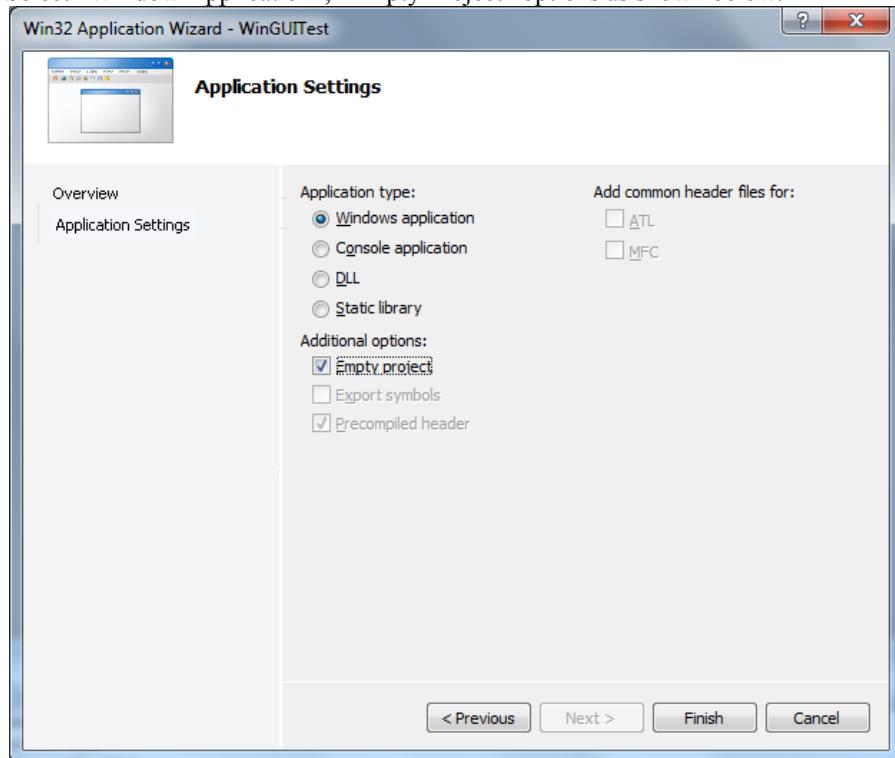
### 11.1.1. Making Empty Window Program

Now we will build a very simple windows application that shows an empty window on your screen. Simply perform the following instructions:

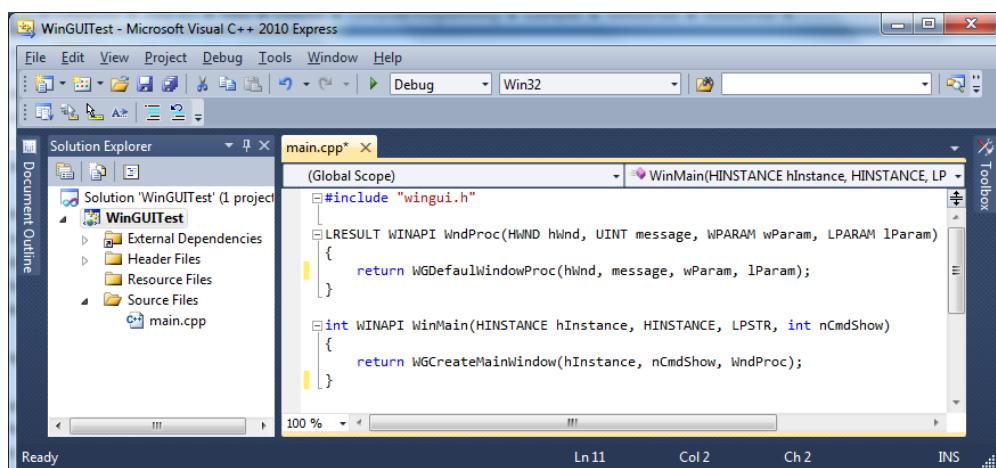
1. Select New Project.
2. Select Visual C++ \ Win32 Project.
3. Select your project location then supply the project name (Ex: WinGUITest).



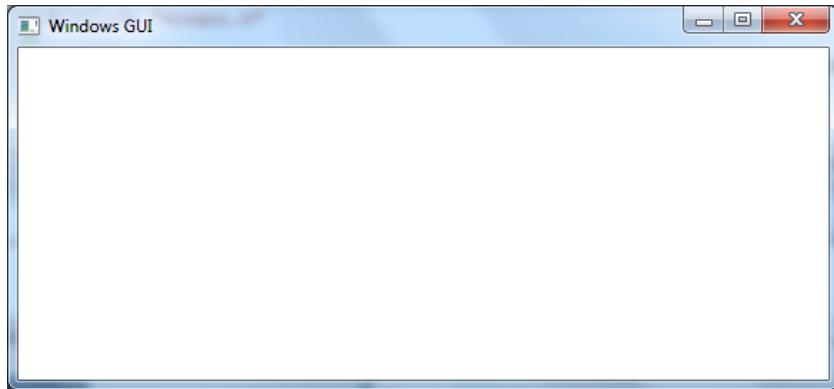
4. In the displayed dialog press “Next”.
5. Select “Window Application”, “Empty Project” options as shown below.



6. Click Finish.
7. The project is created.
8. Add new file “main.cpp”.
9. Copy the file “wingui.h” inside your project folder beside the “main.cpp” file.
10. Enter the following code.



11. The program displays an empty window with title “Windows GUI”.



12. In the following sections we will:
- Modify window Title and Size.
  - Draw Some Graphics.
  - Capture Mouse Event.
  - Capture Keyboard Event.
  - Make Some Animations.

### 11.1.2. Understanding the WINGUI Code

Above program uses following code:

```
#include "wingui.h"

LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE, LPSTR, int nCmdShow)
{
    return WGCreateMainWindow(hInstance, nCmdShow, WndProc);
}
```

**WinMain** is the main program function; it is used instead of the “**main**” function.

The **WGCreateMainWindow** creates the main application window, displays it on the screen, monitor all system and user events and finally return when the user or system close this window.

Whenever any system or user event is triggered the **WGCreateMainWindow** function catch the event then call the supplied function **WndProc** to process any further actions.

**WndProc** function processes all windows events, it has four parameters identifying all event information, and those parameters are:

<b>HWND hWnd</b>	The handle of the window that receive the event.
<b>UINT message</b>	The event type for example: <ul style="list-style-type: none"> <li>• WM_CREATE: The window is created but not displayed yet.</li> </ul>

	<ul style="list-style-type: none"> <li>• WM_SHOWWINDOW: The window will be displayed.</li> <li>• WM_PAINT: The window will be repainted.</li> <li>• WM_MOUSEMOVE: The mouse changes its position.</li> <li>• WM_LBUTTONDOWN: The left mouse button is pressed down.</li> <li>• WM_LBUTTONUP: The left mouse button is released up.</li> <li>• WM_KEYDOWN: A keyboard key is pressed down.</li> <li>• WM_KEYUP: A keyboard key is released up.</li> <li>• ...</li> </ul>
<b>WPARAM wParam</b> <b>LPARAM lParam</b>	Thos parameters have different information depending on the event type. For example in mouse events they contain the mouse location on the screen. Another example in keyboard events they contain which key is pressed in the keyboard.

**WGDefaultWindowProc** function must be called to handle advanced event processing instead of you.

### 11.1.3. Windows APIs data types

As you can see; windows APIs uses data types with new names like HWND, LPARAM, UNIT and more. Actually nothing new, all those names are aliases of existing C data types. For example the UINT data type means “**unsigned long**”. This can be simply mad using “**typedef**” directive.

```
typedef unsigned long UINT;
```

Further examples are mentioned in the following table:

<b>Win API data type</b>	<b>Description</b>
<b>BYTE</b>	Numeric 8 Bit unsigned value
<b>WORD</b>	Numeric 16 Bit unsigned value
<b>DWORD</b>	Numeric 32 Bit unsigned value
<b>LPSTR</b>	Address of string value
<b>UINT</b>	Numeric integer value
<b>LPARAM, WPARAM</b>	Numeric integer value holds the event information
<b>HWND</b>	Numeric integer value holds the window handle
<b>HDC</b>	Numeric integer value holds the graphics device context handle
<b>HBRUSH</b>	Numeric integer value holds the drawing brush handle
<b>HPEN</b>	Numeric integer value holds the drawing pen handle
<b>HFONT</b>	Numeric integer value holds the drawing font handle
<b>HBITMAP</b>	Numeric integer value holds the bitmap handle
<b>HINSTANCE</b>	Numeric integer value holds the application handle

Windows APIs uses data type aliases to clarify the purpose of the data type. HWND and HINSTANCE is the same data type but the first one is used to hold the window handle, while the other is used to hold the application handle.

#### 11.1.4. Modify Window Title

Simply to modify windows title, we can use the windows API function:

```
BOOL SetWindowText(HWND, LPCTSTR);
```

The function takes the window handle and the new title text and returns true in case of success and FALSE in case of failure.

The question is where to use the **SetWindowText** function?

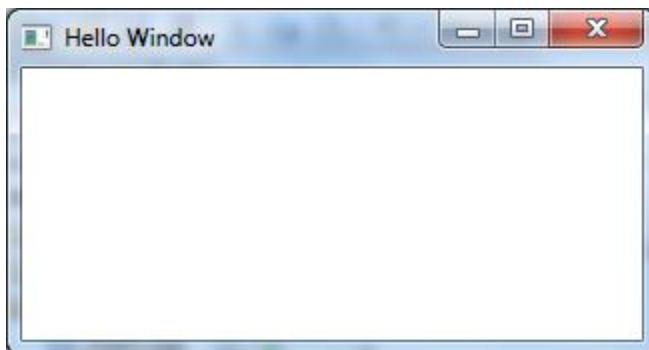
To answer this question, we need to change the window title after it has been created and before it will be displayed. This means we can write our code inside WM\_CREATE event because this event is invoked directly after creating the window and before displaying it.

Try the following code:

```
LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
        case WM_CREATE:
            SetWindowText(hWnd, "Hello Window");
            break;
    }

    return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}
```

Above code produce the following window.



#### 11.1.5. Modify Window Size

By default the application window placed at random location and has a random size. To change the window position and size try the following code:

```
LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
```

```
{
    switch(message)
    {
        case WM_CREATE:
            SetWindowText(hWnd, "Hello Window");
            MoveWindow(hWnd, 200, 100, 400, 300, FALSE);
            break;
    }

    return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}
```

Above code produce the following window. The window is moved to the position 200, 100 and resized to 400 and 300. The mention dimension is measured in pixels.

#### 11.1.6. Make a Full Screen Window

By default the application window has a caption and a thick border used for resizing. This configuration called a window style, this style is called “**WS\_OVERLAPPEDWINDOW**”. This style consists of a set of styles to add caption, border and other properties to the Overlapped window. Overlapped window style is the base window style required by any application window.

```
#define WS_OVERLAPPEDWINDOW (WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU
| WS_THICKFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX)
```

Full Screen application fills the whole screen and has no caption or border. Also full screen application placed over other windows applications even the windows task bar.

```
LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int screenWidth, screenHeight;

    switch(message)
    {
        case WM_CREATE:
        {
            int screenWidth = GetSystemMetrics(SM_CXSCREEN);
            int screenHeight = GetSystemMetrics(SM_CYSCREEN);
            SetWindowLong(hWnd, GWL_STYLE, WS_OVERLAPPED);
            SetWindowPos(hWnd, HWND_TOP, 0, 0,
            screenWidth, screenHeight, SWP_SHOWWINDOW);
        }
        break;
    }

    return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}
```

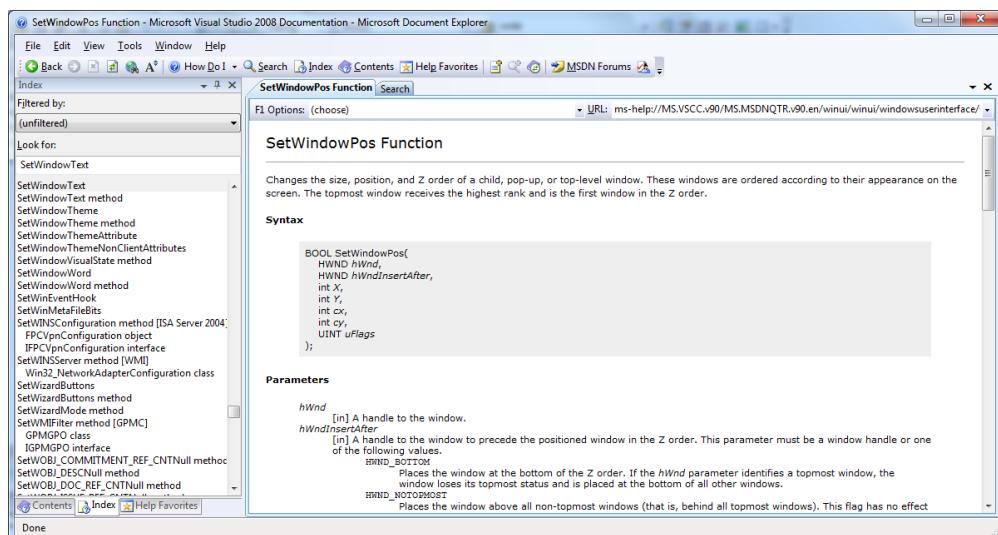
Above code retrieve the dimension of the screen using **GetSystemMetrics** function. **SetWindowLong** function changes the window style to **WS\_OVERLAPPED** only; this will

remove the other styles like the Caption and the Thick border. Finally the **SetWindowPos** function sets the window at top of the other applications and change the size and the position to fill the whole screen.

### 11.1.7. Getting Help on Windows APIs

Windows libraries contain hundreds of functions called API (Application Programming Interface). Explaining each function details is outside the scope of this book; however you can find the APIs details at MSDN (Microsoft Software Development Network). You must install MSDN software to access the windows APIs help, otherwise you can search the online MSDN at: <http://www.msdn.com/>.

Assuming you have installed the MSDN, move the text cart [] inside the required function then press F5, the MSDN is opened showing up the whole function details.



### 11.1.8. Draw Simple Graphics

To perform any permanent drawing in your window, you need to place your code inside the **WM\_PAINT** event. **WM\_PAINT** event is triggered whenever the system decides to re-paint the entire window or part of it. The system decides to re-paint the windows in many situations, for examples:

- When the window is resized.
- When the window is activated after being covered with other applications.
- When the window is displayed after being hidden or after the creation.
- When the programmer commands the system that he need to repaint the window in order to make some animation.

To start drawing you need to retrieve a handle to the graphics context object, this handle is required by all drawing functions. Graphics Context plays as the painter who performs the painting on the Window Clint Area. Try the following code.

```

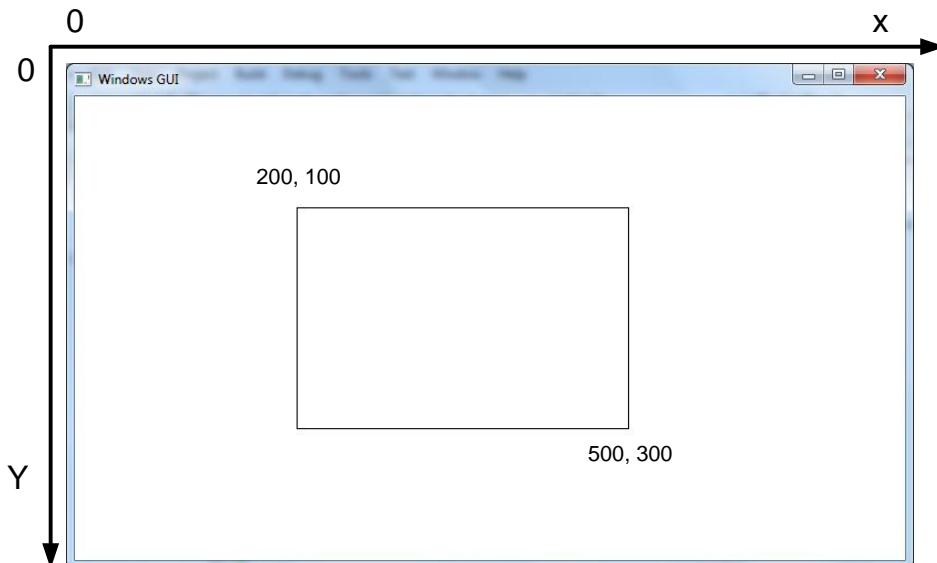
HDC hDC;

switch(message)
{
case WM_PAINT:
{
    hDC = GetDC(hWnd);
    Rectangle(hDC, 200, 100, 500, 300);
    ReleaseDC(hWnd, hDC);
}
break;
}

```

**GetDC** starts the graphics mode and returns the device context handle. You must call **ReleaseDC** to end the graphics mode.

**Rectangle** draws a rectangle on the window client area using the device context handle. This example draws a rectangle located at (left = 200, top = 100, right = 500, bottom = 300).



### 11.1.9. Draw More Graphics

Using windows graphics APIs you can draw lines, rectangles, ellipses, polygons and text. Try following example:

```

switch(message)
{
case WM_PAINT:
{

```

```

HDC hDC = GetDC(hWnd);
Rectangle(hDC, 100, 100, 300, 300);

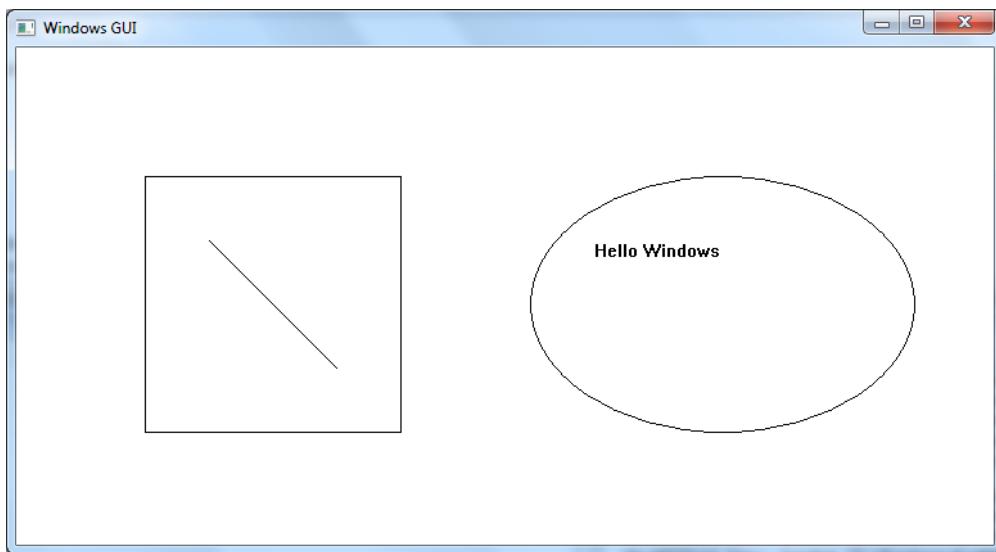
Ellipse(hDC, 400, 100, 700, 300);

MoveToEx(hDC, 150, 150, NULL);
LineTo(hDC, 250, 250);

TextOut(hDC, 450, 150, "Hello Windows", 13);
ReleaseDC(hWnd, hDC);
}

break;
}

```



**Ellipse** draws an ellipse inside the given rectangle. To draw a line you must move to the starting location by **MoveToEx** function then call **LineTo** to draw the line to the specified location.

Printing text using **printf** does not work with windows applications. Alternatively you can use **TextOut** function. Know that **TextOut** can print the supplied text only; however it is not able to format numeric values. In the next sections we will propose an alternative solution.

#### 11.1.10. Changing the Brush, the Pen and the Font

Graphics Device Context (DC) uses Brushes to fill any drawing object. Using windows APIs you can change the brush color and shape. Brush has many shapes like solid, cross hatched, diagonal hatched and more.

DC uses Pens to draw the border of any drawing object. Using windows APIs you can change the Pen color, thickness and style. Pen has many styles like solid, dotted and more.

DC uses Fonts to draw the text. Using windows APIs you can change the Font color, family, size, orientation, and style. Font has many styles like bold, italic, underline and strike out.

Initially the DC contains a default brush, pen and font objects, to change them you need to perform the following scenario:

Assume you need to change the brush color and shape

1. **Create** the New Brush
2. **Select** the New Brush into the DC and Store the Old Brush in a temporary location
3. **Draw** whatever you want using the New Brush
4. When you have finished **Select** the Old Brush into the DC
5. Finally, **Destroy** the New Brush

Do the same scenario to change the Pen or Font.

Finally it is important to understand how to specify the color value. As we know from physics Colors consists of three components (**RED**, **GREEN**, **BLUE**), although to specify colors in your program you must specify the value of each component. Old computers were able to view a limited set of colors (2 colors or monochrome mode, 16/256 color or indexed mode). Now all computers support true color mode (24 bit color mode provides  $2^{24}$  color alternatives, 32 bit color mode provides  $2^{32}$  color alternatives).

Computer with 24 Bit color capability represents the color with a **DWORD (unsigned int)**, three bytes are used to specify the three color components, and the fourth one is not used and must be set with 0. Each color components takes up to 256 values, use small values to increase the darkness or large values to increase the lightness.

0	Blue	Green	Red
---	------	-------	-----

Windows APIs rename the color data type from (**unsigned int**) to **COLOREF**. Following examples illustrate how to specify different color values:

```
COLORREF C;
C = 0x000000FF;      //Red Color
C = 0x0000FF00;      //Green Color
C = 0x00FF0000;      //Blue Color
C = 0x00550000;      //Dark Blue Color
C = 0x00000000;      //Black Color
C = 0x00FFFFFF;      //White Color
C = 0x00888888;      //Gray Color
```

Alternatively, you can use the **RGB(,,)** macro to construct the color value. For example:

```
COLORREF C = RGB(255, 255, 0); //Yellow Color
```

Finally, try the following code:

```
HDC hDC;
HBRUSH hOldBrush, hBrush;
```

```

switch(message)
{
case WM_PAINT:
{
    hDC = GetDC(hWnd);

    //1
    hBrush = WGCreateBrush(RGB(255, 0, 0), HS_SOLID);

    //2: First Rectangle
    Rectangle(hDC, 50, 50, 150, 150);

    //3
    hOldBrush = (HBRUSH) SelectObject(hDC, hBrush);

    //4: Second Rectangle
    Rectangle(hDC, 200, 50, 300, 150);

    //5
    SelectObject(hDC, hOldBrush);

    //6: Third Rectangle
    Rectangle(hDC, 350, 50, 450, 150);

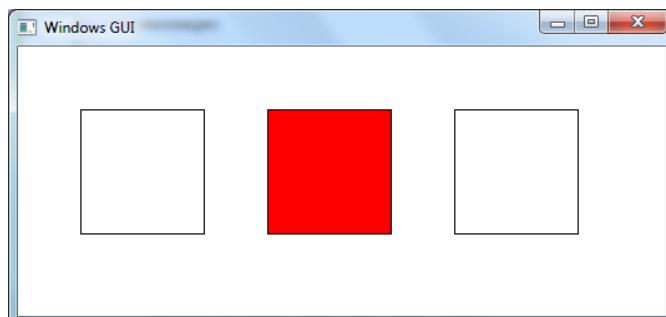
    //7
    DeleteObject(hBrush);

    ReleaseDC(hWnd, hDC);
}

break;
}

```

Above example draw three rectangles as shown below:



The drawing operations are performed as follows:

1. The function **WGCreateBrush** creates a red and solid brush. The new brush is stored in the **hBrush** variable.
2. The first rectangle is drawn using the default DC brush.
3. The **SelectObject(hBrush)** replaces the existing brush with **hBrush** and return back the existing one. The old brush is saved in a temporary variable **hOldBrush**.

4. The second rectangle is drawn using the new **hBrush**.
5. The **SelectObject(hOldBrush)** replaces the existing brush with **hOldBrush** and return back the exiting one.
6. The third rectangle is drawn using the default DC brush.
7. The function **DeleteObject(hBrush)** destroys the created brush.

Following example shows how to use Brushes, Pens and Fonts.

```

HDC hDC;
HBRUSH hOldBrush, hBrush1, hBrush2, hBrush3;
HPEN hOldPen, hPen1, hPen2, hPen3;
HFONT hOldFont, hFont1, hFont2;

switch(message)
{
case WM_PAINT:
{
    hDC = GetDC(hWnd);

    SetPixel(hDC, 10, 10, RGB(255, 0, 0));
    SetPixel(hDC, 11, 11, RGB(255, 0, 0));
    SetPixel(hDC, 12, 12, RGB(255, 0, 0));

    hBrush1 = WGCCreateBrush(RGB(255, 0, 0), HS_SOLID);
    hBrush2 = WGCCreateBrush(RGB(128, 0, 0), HS_CROSS);
    hBrush3 = WGCCreateBrush(0, HS_NULL);

    hPen1 = WGCCreatePen(RGB(128, 128, 0), 5, PS_SOLID);
    hPen2 = WGCCreatePen(0, 0, PS_NULL);
    hPen3 = WGCCreatePen(RGB(0, 128, 0), 1, PS_DOT);

    hFont1 = WGCCreateFont("Arial", 20, FS_ITALIC, 0);
    hFont2 = WGCCreateFont("Calibri", 40, FS_BOLD, -450);

    hOldBrush = (HBRUSH)SelectObject(hDC, hBrush1);
    hOldPen = (HPEN)SelectObject(hDC, hPen1);
    Rectangle(hDC, 50, 50, 150, 150);

    SelectObject(hDC, hBrush2);
    SelectObject(hDC, hPen2);
    Rectangle(hDC, 50, 200, 150, 300);

    SelectObject(hDC, hBrush3);
    SelectObject(hDC, hPen3);
    Ellipse(hDC, 250, 50, 350, 150);

    hOldFont = (HFONT)SelectObject(hDC, hFont1);
    SetTextColor(hDC, RGB(128, 225, 0));
    TextOut(hDC, 250, 200, "Hello Windows", 13);

    SelectObject(hDC, hFont2);
    SetTextColor(hDC, RGB(0, 225, 128));
    TextOut(hDC, 300, 220, "GUI", 3);

    SelectObject(hDC, hOldBrush);
    SelectObject(hDC, hOldPen);
}
}

```

```

    SelectObject(hDC, hOldFont);

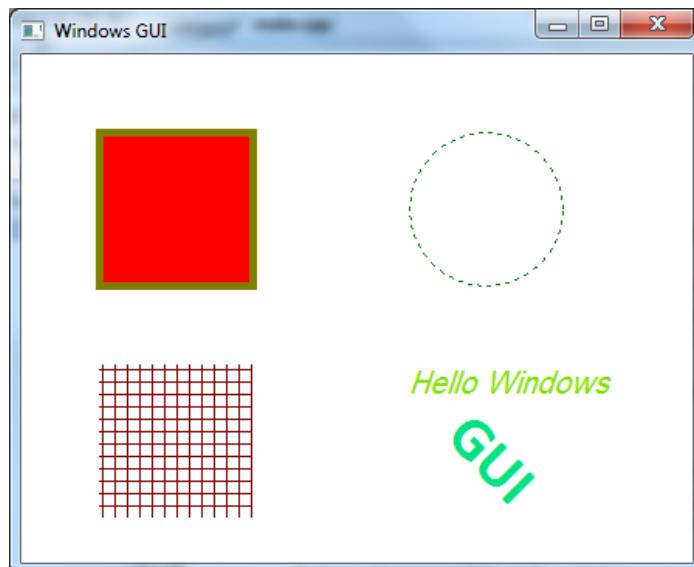
    DeleteObject(hBrush1);
    DeleteObject(hBrush2);
    DeleteObject(hBrush3);

    DeleteObject(hPen1);
    DeleteObject(hPen2);
    DeleteObject(hPen3);

    DeleteObject(hFont1);
    DeleteObject(hFont2);

    ReleaseDC(hWnd, hDC);
}
break;
}

```



**WGCreateBrush** function creates a brush by specifying the color and the hatch style, following hatch styles can be specified:

HS_SOLID	: solid
HS_BDIAGONAL	: 45-degree upward left-to-right hatch
HS_CROSS	: Horizontal and vertical crosshatch
HS_DIAGCROSS	: 45-degree crosshatch
HS_FDIAGONAL	: 45-degree downward left-to-right hatch
HS_HORIZONTAL	: Horizontal hatch
HS_VERTICAL	: Vertical hatch
HS_SOLID	: Solid color with no hatch
HS_NULL	: Hollow brush

**WGCreatePen** function creates a pen by specifying the color, the thickness in pixels and the style, know that non-solid styles will work only if the specified thickness is 0 or 1. Also 0 thickness assumed as 1, following pen styles can be specified:

```

PS_SOLID           : The pen is solid.
PS_DASH            : The pen is dashed.
PS_DOT             : The pen is dotted.
PS_DASHDOT         : The pen has alternating dashes and dots.
PS_DASHDOTDOT     : The pen has alternating dashes and 2 dots.
PS_NULL            : The pen is invisible.

```

**WGCreateFont** function creates a font by specifying the name, the size, the style and the orientation. Font name specifies the type of the font family; you must use one of the installed fonts in your system. There are many common fonts like “Arial”, “Times New Roman”, “Courier New”, “Calibri” and “Verdana”. Font size is specified in logical screen units, logical units are related to the monitor size and the resolution. Font style can take one or more of the following values: **FS\_BOLD**, **FS\_ITALIC**, **FS\_UNDERLINE**, **FS\_STRIKOUT**.

**SetTextColor** function sets the current font color.

### 11.1.11. Draw Formatted Text

**TextOut** function does not support formatting of numeric values like integer and real. “wingui.h” library introduce a wrapper function **WGPrintf** which provide a similar behavior to **printf** function but in graphics mode. Try the following code:

```

HDC hDC;
HFONT hOldFont, hFont;

switch(message)
{
case WM_PAINT:
{
    hDC = GetDC(hWnd);

    hFont = WGCreateFont("Calibri", 50,
                         FS_BOLD|FS_ITALIC|FS_UNDERLINE, 0);
    hOldFont = (HFONT) SelectObject(hDC, hFont);

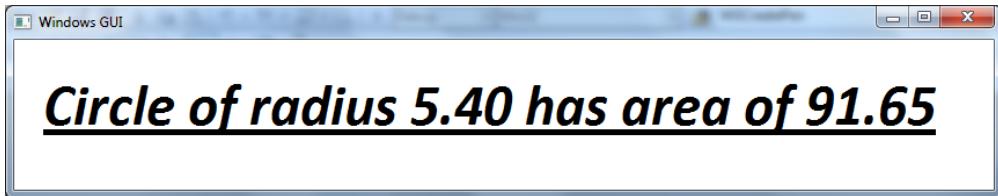
    double radius = 5.4;
    double area = 3.143 * radius * radius;
    WGPrintf(hDC, 25, 25,
              "Circle of radius %2.2lf has area of %2.2lf",
              radius, area);

    SelectObject(hDC, hOldFont);
    DeleteObject(hFont);

    ReleaseDC(hWnd, hDC);
}
break;
}

```

Above program shows the following window.



### 11.1.12. Capturing Mouse Events

The operating system is responsible on displaying the mouse cursor on the screen and sends various mouse events to the appropriate window. Following mouse events can be captured:

1. `WM_LBUTTONDOWN`, `WM_RBUTTONDOWN` and `WM_MBUTTONDOWN`: Mouse button is pressed.
2. `WM_LBUTTONUP`, `WM_RBUTTONUP` and `WM_MBUTTONUP`: Mouse button is released.
3. `WM_MOUSEMOVE`: Indicates that the mouse is moved over the window.
4. `WM_MOUSEWHEEL`: Indicates that the mouse wheel is rotated.

Mouse events are provided with the following information:

<code>lParam</code>	The highest word contains the Y value. The lowest word contains the X value.
<code>wParam</code>	<p>The highest word contains the wheel movement direction; positive value indicates forward rotation, negative value indicates backward rotation.</p> <p>The lowest word contains the current keyboard and mouse status:</p> <ul style="list-style-type: none"> <li><code>MK_CONTROL</code>: The control key is down.</li> <li><code>MK_SHIFT</code>: The shift key is down.</li> <li><code>MK_LBUTTON</code>: The left mouse button is down.</li> <li><code>MK_MBUTTON</code>: The middle mouse button is down.</li> <li><code>MK_RBUTTON</code>: The right mouse button is down.</li> </ul>

Following examples illustrates how to use different mouse events:

**Example 1:** following example prints the mouse position on the window.

```

switch(message)
{
    case WM_MOUSEMOVE:
    {
        int x = LOWORD(lParam);
        int y = HIWORD(lParam);

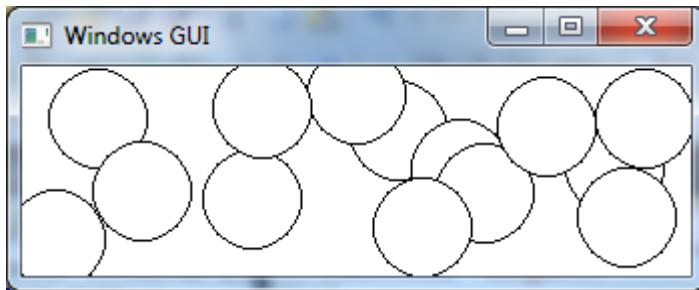
        HDC hDC = GetDC(hWnd);
        WGPrintf(hDC, 10, 10,
                 "The mouse is located at %d,%d", x, y);
        ReleaseDC(hWnd, hDC);
    }
    break;
}

```

**Example 2:** following example draw a circle at the mouse location when the left mouse button is pressed down.

```
switch(message)
{
case WM_LBUTTONDOWN:
{
    int x = LOWORD(lParam);
    int y = HIWORD(lParam);

    HDC hDC = GetDC(hWnd);
    Ellipse(hDC, x-25, y-25, x+25, y+25);
    ReleaseDC(hWnd, hDC);
}
break;
}
```

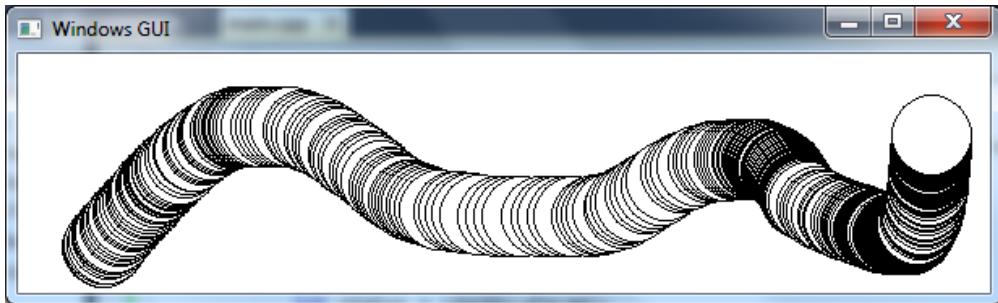


**Example 3:** following example lets the user to make a freehand drawing.

```
switch(message)
{
case WM_MOUSEMOVE:
{
    int status = LOWORD(wParam);

    if(status&MK_LBUTTON)
    {
        int x = LOWORD(lParam);
        int y = HIWORD(lParam);

        HDC hDC = GetDC(hWnd);
        Ellipse(hDC, x-25, y-25, x+25, y+25);
        ReleaseDC(hWnd, hDC);
    }
}
break;
}
```



### 11.1.13. Capturing Keyboard Events

The operating system sends the following keyboard events to the window having the input focus:

- **WM\_KEYDOWN:** Indicates that a keyboard button is pressed down.
- **WM\_KEYUP:** Indicates that a keyboard button is released up.
- **WM\_CHAR:** Indicates that keyboard sends a letter.

Keyboard events provided with the following information:

wParam	<p>For <b>WM_KEYDOWN</b> and <b>WM_KEYUP</b> events: Specifies which virtual keyboard button is pressed. Following list is part of the expected values: <b>VK_SHIFT, VK_ALT, VK_CONTROL, VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT, VK_HOME, VK_INSERT, VK_DELETE, VK_RETURN, VK_ESC, VK_NUMLOCK, VK_END.</b></p> <p>For <b>WM_CHAR</b> event: Specifies the character code.</p>
--------	--

Following examples illustrate how to use different keyboard events:

**Example 1:** following example prints the provided keyboard letter.

```

switch(message)
{
    case WM_CHAR:
    {
        //Generate a random position (x,y) inside the window rect
        int width = WGGetClientWidth(hWnd);
        int height = WGGetClientHeight(hWnd);
        int x = 50 + rand()%width - 100;
        int y = 50 + rand()%height - 100;

        HFONT hFont = WGCreateFont("Calibri", 40, FS_BOLD, 0);

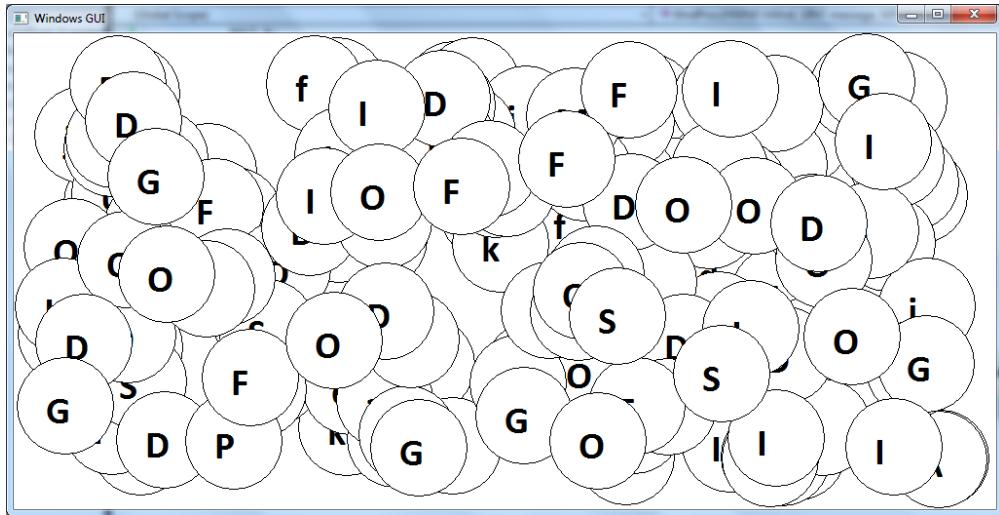
        HDC hDC = GetDC(hWnd);
        HFONT hOldFont = (HFONT)SelectObject(hDC, hFont);
        Ellipse(hDC, x-50, y-50, x+50, y+50);
        WPrintf(hDC, x-20, y-20, "%c", (char)wParam);
        SelectObject(hDC, hOldFont);
        ReleaseDC(hWnd, hDC);
    }
}

```

```

        DeleteObject(hFont);
    }
    break;
}

```



**Example 2:** following example moves a shape up, down, right and left.

```

int gX = 200;
int gY = 200;

HRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
        case WM_KEYDOWN:
        {
            switch(wParam)
            {
                case VK_UP: gY -= 5; break;
                case VK_DOWN: gY += 5; break;
                case VK_RIGHT: gX += 5; break;
                case VK_LEFT: gX -= 5; break;
            }

            InvalidateRect(hWnd, NULL, FALSE);
        }
        break;

        case WM_PAINT:
        {
            HDC hDC = GetDC(hWnd);
            Ellipse(hDC, gX-50, gY-50, gX+50, gY+50);
            ReleaseDC(hWnd, hDC);
        }
        break;
    }
}

```

```

    return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}

```

The **InvalidateRect** function with **FALSE** forces the system to erase the window then trigger the **WM\_PAINT** event.

#### 11.1.14. Making Animation

Animation is making rapid changes of the drawing contents along the time. Timer events can be used to produce the animation. You can create several timers events to control the animation scenario.

Following examples illustrate how to use the timers to create some animation.

**Example 1:** following example creates several counters using different timers.

```

int gCounter1 = 0;
int gCounter2 = 0;
int gCounter3 = 0;

HRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
        case WM_CREATE:
        {
            SetTimer(hWnd, 1, 10, NULL);
            SetTimer(hWnd, 2, 100, NULL);
            SetTimer(hWnd, 3, 1000, NULL);
        }
        break;

        case WM_TIMER:
        {
            HDC hDC = GetDC(hWnd);
            if(wParam==1) WGPrintf(hDC, 50, 50, "%d", gCounter1++);
            if(wParam==2) WGPrintf(hDC, 150, 50, "%d", gCounter2++);
            if(wParam==3) WGPrintf(hDC, 250, 50, "%d", gCounter3++);
            ReleaseDC(hWnd, hDC);
        }
        break;
    }

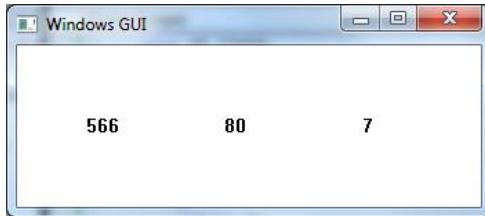
    return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}

```

The function **SetTimer(hWnd, 1, 10)** creates a timer event with identifier = 1 and time interval = 10 milliseconds.

In above example three timers is created with identifiers equal 1, 2 and 3 and time intervals equal 10, 100, 1000 milliseconds.

**WM\_TIMER** event is called whenever any timer is triggered; the wParam contains the timer identifier.



**Example 2:** Bouncing ball, version 1.0.

```

int gX = 0;
int gY = 0;
int gDX = 10;
int gDY = 10;

LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
        case WM_CREATE:
            {
                SetTimer(hWnd, 100, 1, NULL);
            }
            break;

        case WM_TIMER:
            {
                int width = WGGetClientWidth(hWnd);
                int height = WGGetClientHeight(hWnd);

                if(gX<0 || gX>width)gDX *= -1;
                if(gY<0 || gY>height)gDY *= -1;

                gX += gDX;
                gY += gDY;

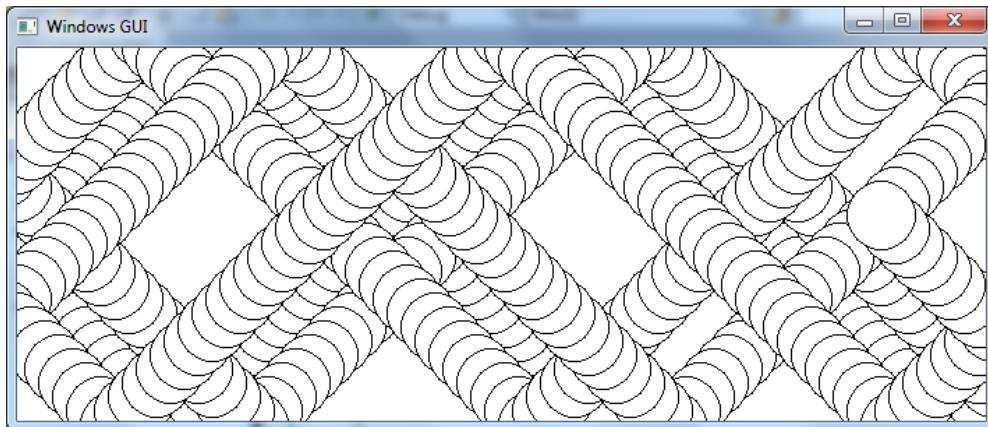
                HDC hDC = GetDC(hWnd);
                Ellipse(hDC, gX - 25, gY - 25, gX + 25, gY + 25);
                ReleaseDC(hWnd, hDC);
            }
            break;
    }

    return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}

```

Above program shows a bouncing ball, whenever it reaches to the window edge it changes its direction to the opposite direction. The program draws the ball at each timer event. Following figure shows the program output.

`WGGetClientWidth`, `WGGetClientHeight` function retrieves the client area width and height respectively. Client area represents the whole window area except the caption, the border and the scrollbars if exist.



What if you need to show only the moving ball at the new position only? This means we have to erase the whole screen every drawing time.

### Example 3: Bouncing ball, version 2.0.

Following example:

1. Draws only the ball at the new position.
2. Shows the program in full screen mode.
3. Displays a colored text showing the current ball position.
4. Displays another text beside the moving ball.

```
int gX = 0;
int gY = 0;
int gDX = 10;
int gDY = 10;

LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
        case WM_CREATE:
        {
            //Show Window in Full Screen Mode
            int screenWidth = GetSystemMetrics(SM_CXSCREEN);
            int screenHeight = GetSystemMetrics(SM_CYSCREEN);
            SetWindowLong(hWnd, GWL_STYLE, WS_OVERLAPPED);
            SetWindowPos(hWnd, HWND_TOP, 0, 0,
                        screenWidth, screenHeight, SWP_SHOWWINDOW);

            SetTimer(hWnd, 1, 50, NULL);
        }
        break;

        case WM_TIMER:
        if(wParam==1)
        {
```

```

int width = WGGetClientWidth(hWnd);
int height = WGGetClientHeight(hWnd);

if(gX<0 || gX>width)gDX *= -1;
if(gY<0 || gY>height)gDY *= -1;

gX += gDX;
gY += gDY;

HDC hDC = GetDC(hWnd);
PatBlt(hDC, 0, 0, width, height, BLACKNESS);

HFONT hFont = WGCreateFont("Calibri", 40, FS_BOLD, 0);
HFONT hOldFont = (HFONT)SelectObject(hDC, hFont);
SetBkMode(hDC, TRANSPARENT);
SetTextColor(hDC, RGB(255,0,0));
WGPrintf(hDC, 50, 50,
    "Bouncing Ball Position %d,%d", gX, gY);
SetTextColor(hDC, RGB(255,255,0));
WGPrintf(hDC, gX + 60, gY, "Bouncing Ball");
SelectObject(hDC, hOldFont);
DeleteObject(hFont);

Ellipse(hDC, gX - 50, gY - 50, gX + 50, gY + 50);

ReleaseDC(hWnd, hDC);
}
break;
}

return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}

```

**PatBlt** function fills the specified area in the DC with (**BLACKNESS**).

The **SetBkMode** function removes the colored background behind the text.



SetBkMode(hDC, TRANSPARENT);

SetBkMode(hDC, OPAQUE);

By default windows uses **OPAQUE** mode, unless you change it by yourself.

## 11.2 GDI Programming

### 11.2.1. Bouncing Balls Screen Saver

Following program creates bouncing balls with random color. Whenever the user clicks on the mouse, the program creates a new ball at the mouse position and assigns to it a random color. User can create as much balls up to 1000. Bouncing ball information is stored in several arrays. User can control the speed using the Up and Down arrow, or by rotating the mouse wheel. User can use keyboard keys ('s' and 'r') to stop and to resume the animation.

```

int gInterval = 50;
int gnShapes = 0;
int gX[1000];
int gY[1000];
int gDX[1000];
int gDY[1000];
int gColors[1000];

LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
        case WM_CREATE:
        {
            int screenWidth = GetSystemMetrics(SM_CXSCREEN);
            int screenHeight = GetSystemMetrics(SM_CYSCREEN);
            SetWindowLong(hWnd, GWL_STYLE, WS_OVERLAPPED);
            SetWindowPos(hWnd, HWND_TOP, 0, 0,
                         screenWidth, screenHeight, SWP_SHOWWINDOW);

            SetTimer(hWnd, 1, 50, NULL);
        }
        break;
        case WM_LBUTTONDOWN:
        {
            if(gnShapes==1000)break;

            gX[gnShapes] = LOWORD(lParam);
            gY[gnShapes] = HIWORD(lParam);
            gDX[gnShapes] = (rand()%2)?10:-10;
            gDY[gnShapes] = (rand()%2)?10:-10;
            gColors[gnShapes] =
                RGB(rand()%256,rand()%256,rand()%256);
            gnShapes++;
            InvalidateRect(hWnd, NULL, FALSE);
        }
        break;
        case WM_TIMER:
        {
            InvalidateRect(hWnd, NULL, FALSE);
        }
        break;
        case WM_MOUSEWHEEL:
        {
            short delta = HIWORD(wParam);
            if(delta>0)gInterval--;
        }
    }
}

```

```

        if(delta<0)gInterval++;
        if(gInterval<1)gInterval = 1;
        if(gInterval>1000)gInterval = 1000;
        KillTimer(hWnd, 1);
        SetTimer(hWnd, 1, gInterval, NULL);
    }
    break;
case WM_KEYDOWN:
{
    if(wParam==VK_DOWN)gInterval--;
    if(wParam==VK_UP)gInterval++;
    if(gInterval<1)gInterval = 1;
    if(gInterval>1000)gInterval = 1000;
    KillTimer(hWnd, 1);
    SetTimer(hWnd, 1, gInterval, NULL);
}
break;
case WM_CHAR:
{
    if(wParam=='s'||wParam=='S')KillTimer(hWnd, 1);
    if(wParam=='r'||wParam=='R')
        SetTimer(hWnd, 1, gInterval, NULL);
}
break;
case WM_PAINT:
{
    HDC hDC = GetDC(hWnd);
    int width = WGGetClientWidth(hWnd);
    int height = WGGetClientHeight(hWnd);

    PatBlt(hDC, 0, 0, width, height, BLACKNESS);

    SetBkMode(hDC, TRANSPARENT);
    SetTextColor(hDC, RGB(255,255,0));
    WGPrintf(hDC, 10, 10, "Interval %d", gInterval);

    for(int i=0;i<gnShapes;i++)
    {
        HBRUSH hBrush = WGCreateBrush(gColors[i], HS_SOLID);
        HBRUSH hOldBrush = (HBRUSH)SelectObject(hDC, hBrush);

        if(gX[i]>R.right || gX[i]<R.left)gDX[i] *= -1;
        if(gY[i]>R.bottom || gY[i]<R.top)gDY[i] *= -1;

        gX[i] += gDX[i];
        gY[i] += gDY[i];
        Ellipse(hDC, gX[i]-50, gY[i]-50, gX[i]+50, gY[i]+50);

        SelectObject(hDC, hOldBrush);
        DeleteObject(hBrush);
    }
    ReleaseDC(hWnd, hDC);
}
break;
}

return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}

```

### 11.2.2. Simple Drawing

Following program provides a simple drawing with some editing facilities. User can add random colored circles by pressing the right mouse button. User can select any displayed circle by clicking on it using the left mouse button then drag/move it. While selecting the circle, user can use the mouse wheel to increase or to decrease the radius of the circle.

```

int gnShapes = 0;
int gX[1000];
int gY[1000];
int gS[1000];
int gColors[1000];
int gSelected = -1;

LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
        case WM_CREATE:
        {
            int screenWidth = GetSystemMetrics(SM_CXSCREEN);
            int screenHeight = GetSystemMetrics(SM_CYSCREEN);
            SetWindowLong(hWnd, GWL_STYLE, WS_OVERLAPPED);
            SetWindowPos(hWnd, HWND_TOP, 0, 0,
                         screenWidth, screenHeight, SWP_SHOWWINDOW);
        }
        break;
        case WM_LBUTTONDOWN:
        {
            if(gSelected>=0)
            {
                gSelected = -1;
            }
            else
            {
                short x = LOWORD(lParam);
                short y = HIWORD(lParam);

                gSelected = -1;
                for(int i= gnShapes-1;i>=0;i--)
                {
                    double distance =
                        sqrt(pow((x-gX[i]), 2.0) + pow(y-gY[i], 2.0));
                    if(distance<gS[i])
                    {
                        gSelected = i;
                        break;
                    }
                }
            }
        }
        break;
        case WM_MOUSEMOVE:
    }
}

```

```

{
    if(gSelected>=0)
    {
        short x = LOWORD(lParam);
        short y = HIWORD(lParam);
        gX[gSelected] = x;
        gY[gSelected] = y;
        InvalidateRect(hWnd, NULL, FALSE);
    }
    break;
case WM_RBUTTONDOWN:
{
    if(gSelected>=0 || gnShapes==1000)break;

    gX[gnShapes] = LOWORD(lParam);
    gY[gnShapes] = HIWORD(lParam);
    gS[gnShapes] = 50;
    gColors[gnShapes] =
        RGB(rand()%256,rand()%256,rand()%256);
    gnShapes++;
    InvalidateRect(hWnd, NULL, FALSE);
}
break;
case WM_MOUSEWHEEL:
{
    if(gSelected>=0)
    {
        short delta = HIWORD(wParam);
        if(delta>0)gS[gSelected]+=5;
        else gS[gSelected]-=5;
        if(gS[gSelected]>250)gS[gSelected]=250;
        if(gS[gSelected]<5)gS[gSelected]=5;
        InvalidateRect(hWnd, NULL, FALSE);
    }
}
break;
case WM_PAINT:
{
    HDC hDC = GetDC(hWnd);
    int width = WGGetClientWidth(hWnd);
    int height = WGGetClientHeight(hWnd);

    PatBlt(hDC, 0, 0, width, height, BLACKNESS);

    for(int i=0;i<gnShapes;i++)
    {
        HBRUSH hBrush =
            WGCreateBrush(gColors[i], HS_SOLID);
        HBRUSH hOldBrush =
            (HBRUSH)SelectObject(hDC, hBrush);

        Ellipse(hDC, gX[i]-gS[i], gY[i]-gS[i],
                gX[i]+gS[i], gY[i]+gS[i]);

        SelectObject(hDC, hOldBrush);
        DeleteObject(hBrush);
    }
}

```

```

        ReleaseDC(hWnd, hDC);
    }
    break;
}

return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}

```

Execute above program then try to add a large number of circles, then select a circle from the bottom, then increase its radius, then move it in the window, you will notice a lot of flickers on the screen.

Flickers happen because the drawing operation is performed in front of your eyes. With large number of drawing objects, your eye will notice the drawing operations. This is called a flicker.

To resolve the flickers windows GDI libraries offers the **Memory Device Context** to allow user to complete the drawing far away from user eye then render the final result at once to the **Physical Device Context**.

Following program illustrate this idea, only change the **WM\_PAINT** message in the above program.

```

case WM_PAINT:
{
    int width = WGGetClientWidth(hWnd);
    int height = WGGetClientHeight(hWnd);

    HDC hDC = GetDC(hWnd);
    HDC hMemDC = CreateCompatibleDC(hDC);
    HBITMAP hBitmap = CreateCompatibleBitmap(hDC, width, height);
    HBITMAP hOldBitmap = (HBITMAP)SelectObject(hMemDC, hBitmap);

    PatBlt(hMemDC, 0, 0, width, height, BLACKNESS);

    for(int i=0;i<gnShapes;i++)
    {
        HBRUSH hBrush = WGCreateBrush(gColors[i], HS_SOLID);
        HBRUSH hOldBrush = (HBRUSH)SelectObject(hMemDC, hBrush);

        Ellipse(hMemDC, gX[i]-gS[i], gY[i]-gS[i],
                gX[i]+gS[i], gY[i]+gS[i]);

        SelectObject(hMemDC, hOldBrush);
        DeleteObject(hBrush);
    }

    BitBlt(hDC, 0, 0, width, height, hMemDC, 0, 0, SRCCOPY);
    SelectObject(hMemDC, hOldBitmap);
    DeleteObject(hBitmap);
    DeleteDC(hMemDC);

    ReleaseDC(hWnd, hDC);
}
break;

```

`hMemDC = CreateCompatibleDC(hDC)` function creates another Device Context in the memory similar to the provided DC. This device is called **Memory Device Context**.

The new `hMemDC` is empty with (0, 0) size, it is required to supply it with a bitmap object having a size equals to the window size.

`hBitmap = CreateCompatibleBitmap(hDC, width, height)` function creates the required bitmap. Finally you need to select the bitmap inside the new DC (`hMemDC`) using the `SelectObject` function.

After preparing the memory DC you can draw whatever you want. Once you have finished you can use the `BitBlt` function to copy the entire contents from the Memory DC (`hMemDC`) to the Physical DC (`hDC`).

At the end do not forget the delete the created DC (`hMemDC`) and the created Bitmap (`hBitmap`) objects.

### 11.2.3. Dealing with Bitmaps

Windows GDI APIs provide a set of functions to deal with bitmaps. Bitmap is a type of image format supported by Microsoft windows. You can load and draw bitmap images like any other drawing objects.

```
HBITMAP hBitmap;

LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
        case WM_CREATE:
        {
            hBitmap = (HBITMAP)LoadImage(NULL, "C:\\image.bmp",
                                         IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE);
        }
        break;
        case WM_DESTROY:
        {
            DeleteObject(hBitmap);
        }
        break;
        case WM_PAINT:
        {
            int width = WGGetClientWidth(hWnd);
            int height = WGGetClientHeight(hWnd);

            BITMAP bmpInfo;
            GetObject(hBitmap, sizeof(bmpInfo), &bmpInfo);
            int bmpWidth = bmpInfo.bmWidth;
            int bmpHeight = bmpInfo.bmHeight;

            HDC hDC = GetDC(hWnd);
            HDC hMemDC = CreateCompatibleDC(hDC);
            HBITMAP hOldBitmap = (HBITMAP)SelectObject(hMemDC,
```

```

        hBitmap);

    StretchBlt(hDC, 0, 0, width, height, hMemDC,
                0, 0, bmpWidth, bmpHeight, SRCCOPY);

    SelectObject(hMemDC, hOldBitmap);
    DeleteDC(hMemDC);
}
break;
}

return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}

```

**LoadImage** function loads the bitmap image from the given path; the function returns the handle of the loaded bitmap. Bitmap handle can be used to determine the bitmap size and to draw the bitmap.

**GetObject** function retrieves the bitmap object information. Bitmap information contains the width, the height and other information.

To draw the bitmap it must be selected into a Memory DC (**hMemDC**) then copied to the actual DC (**hDC**) using one of the block transfer functions like (**BitBlt** and **StretchBlt**).

#### 11.2.4. Working with Regions

Using windows GDI you can restrict the drawing inside or outside any shape. Those restricted shapes called a regions. Following program allows user to make a free drawing inside an elliptic area.

```

int gX[1024];
int gY[1024];
int gnShapes = 0;
LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch(message)
    {
        case WM_LBUTTONDOWN:
        {
            if(gnShapes==1000)break;

            gX[gnShapes] = LOWORD(lParam);
            gY[gnShapes] = HIWORD(lParam);
            gnShapes++;
            InvalidateRect(hWnd, NULL, FALSE);
        }
        break;
    case WM_PAINT:
    {
        HDC hDC = GetDC(hWnd);
        HRGN hRgn = CreateEllipticRgn(50, 50, 500, 400);
        SelectClipRgn(hDC, hRgn);
        for(int i=0;i<gnShapes;i++)
        {

```

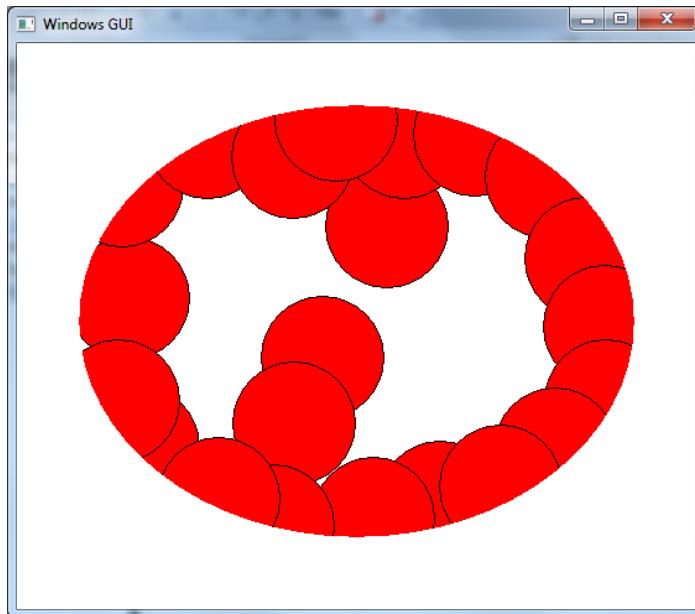
```

HBRUSH hBrush = WGCreateBrush(RGB(255,0,0), HS_SOLID);
HBRUSH hOldBrush = (HBRUSH)SelectObject(hDC, hBrush);
Ellipse(hDC, gX[i]-50, gY[i]-50, gX[i]+50, gY[i]+50);
SelectObject(hDC, hOldBrush);
DeleteObject(hBrush);
}
DeleteObject(hRgn);
ReleaseDC(hWnd, hDC);
}
break;
}

return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}

```

**CreateEllipticRgn** function creates an elliptic region. By selecting this region using **SelectClipRgn** function, the drawing will be restricted only inside the region area.



You can create more complex region by combining several regions. Try the following code:

```

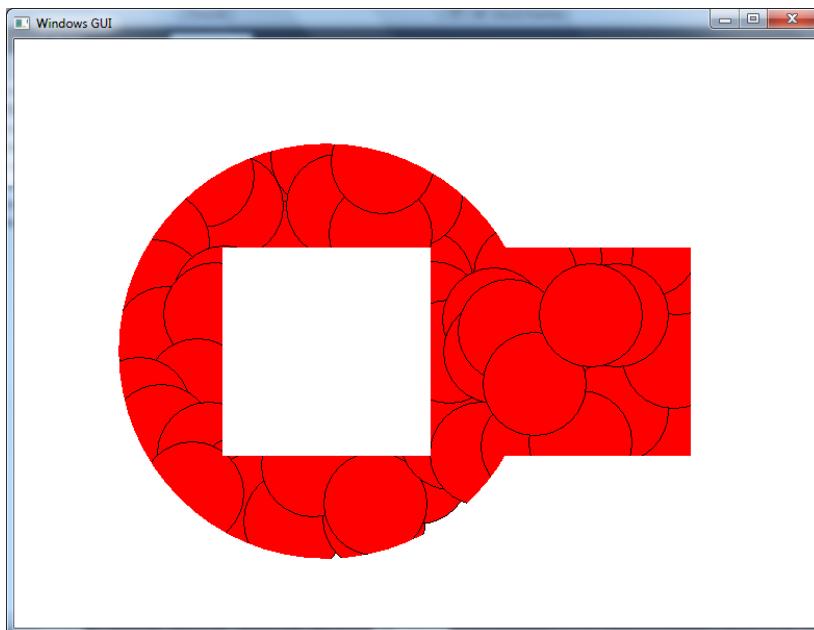
case WM_PAINT:
{
    HDC hDC = GetDC(hWnd);
    HRGN hRgn1 = CreateEllipticRgn(100, 100, 500, 500);
    HRGN hRgn2 = CreateRectRgn(200, 200, 400, 400);
    HRGN hRgn3 = CreateRectRgn(450, 200, 650, 400);
    ExtSelectClipRgn(hDC, hRgn1, RGN_COPY);
    ExtSelectClipRgn(hDC, hRgn2, RGN_DIFF);
    ExtSelectClipRgn(hDC, hRgn3, RGN_OR);
    for(int i=0;i<gNShapes;i++)
    {
        HBRUSH hBrush = WGCreateBrush(RGB(255,0,0), HS_SOLID);

```

```

        HBRUSH hOldBrush = (HBRUSH)SelectObject(hDC, hBrush);
        Ellipse(hDC, gX[i]-50, gY[i]-50, gX[i]+50, gY[i]+50);
        SelectObject(hDC, hOldBrush);
        DeleteObject(hBrush);
    }
    DeleteObject(hRgn1);
    DeleteObject(hRgn2);
    DeleteObject(hRgn3);
    ReleaseDC(hWnd, hDC);
}
break;

```



## 11.3 GUI Programming

### 11.3.1. Showing Windows Messages

Following program displays several message boxes to the user.

```

switch(message)
{
case WM_LBUTTONDOWN:
{
    MessageBox(hWnd, "You pressed the left mouse button.",
               "Any Title...", MB_OK|MB_ICONINFORMATION);
}
break;
case WM_CLOSE:
{

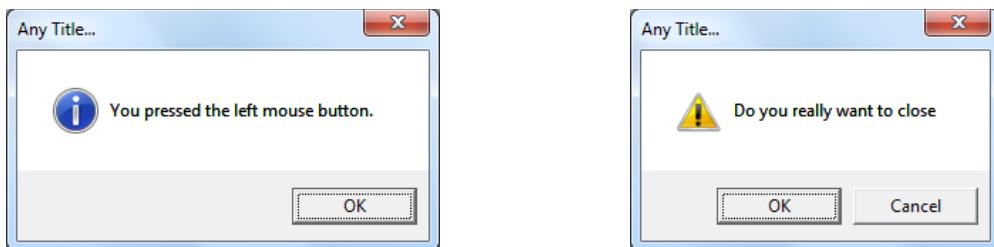
```

```

if(MessageBox(hWnd, "Do you really want to close",
    "Any Title...", MB_OKCANCEL|MB_ICONEXCLAMATION)==IDCANCEL)
{
    return 0;
}
break;
}

```

**MessageBox** function shows a message box, you can specify the message text, the title, the buttons, and the icon.



### 11.3.2. Working with Controls

Windows GUI consists of many controls like button, text boxes, list boxes, combo boxes, check box, radio button, menus, list controls and tree control.

```

switch(message)
{
case WM_CREATE:
{
    HWND hButton;
    hButton = CreateWindow("Button", "", WS_CHILD|WS_VISIBLE,
        10, 10, 200, 50, hWnd, (HMENU)(UINT_PTR)1, NULL, NULL);
    SetWindowText(hButton, "Button 1");

    hButton = CreateWindow("Button", "", WS_CHILD|WS_VISIBLE,
        10, 70, 200, 50, hWnd, (HMENU)(UINT_PTR)2, NULL, NULL);
    SetWindowText(hButton, "Button 2");
}
break;
case WM_COMMAND:
{
    WORD controlEvent = HIWORD(wParam);
    WORD controlID = LOWORD(wParam);
    HWND hControl = (HWND)lParam;
    if(controlEvent==BN_CLICKED)
    {
        HDC hDC = GetDC(hWnd);
        WGPrintf(hDC, 250, 10,
            "You pressed button %d", controlID);
        ReleaseDC(hWnd, hDC);
    }
}
break;
}

```

```
}
```

**CreateWindow** function creates whatever type of windows control; you can specify the control class (type), style, position, size, parent window and identifier.

The **WM\_COMMAND** message catches most of the control events. In this example you can catch the **BN\_CLICKED** event and determine from which button it came.

Class Name determines the type of the control. Following program illustrate how to create and to use different controls classes.

```

switch(message)
{
case WM_CREATE:
{
    HWND hControl;

    hControl = CreateWindow("Button", "", WS_CHILD|WS_VISIBLE,
                           10, 10, 200, 50, hWnd, (HMENU)(UINT_PTR)1, NULL, NULL);
    SetWindowText(hControl, "Button");

    hControl = CreateWindow("EDIT", "",
                           WS_CHILD|WS_VISIBLE|WS_BORDER|ES_MULTILINE|ES_WANTRETURN,
                           10, 70, 200, 50, hWnd, (HMENU)(UINT_PTR)2, NULL, NULL);
    SetWindowText(hControl, "Type any text...");

    hControl = CreateWindow("STATIC", "",
                           WS_CHILD|WS_VISIBLE,
                           10, 130, 200, 50, hWnd, (HMENU)(UINT_PTR)3, NULL, NULL);
    SetWindowText(hControl, "Static Label");

    hControl = CreateWindow("COMBOBOX", "",
                           WS_CHILD|WS_VISIBLE|WS_BORDER|CBS_DROPDOWNLIST, 10, 190,
                           200, 150, hWnd, (HMENU)(UINT_PTR)4, NULL, NULL);
    SendMessage(hControl, CB_ADDSTRING, 0, (LPARAM)"Item1");
    SendMessage(hControl, CB_ADDSTRING, 0, (LPARAM)"Item2");
    SendMessage(hControl, CB_ADDSTRING, 0, (LPARAM)"Item3");

    hControl = CreateWindow("LISTBOX", "",
                           WS_CHILD|WS_VISIBLE|WS_BORDER|LBS_NOTIFY, 10, 250,
                           200, 50, hWnd, (HMENU)(UINT_PTR)5, NULL, NULL);
    SendMessage(hControl, LB_INSERTSTRING, 0, (LPARAM)"Item1");
    SendMessage(hControl, LB_INSERTSTRING, 1, (LPARAM)"Item2");
    SendMessage(hControl, LB_INSERTSTRING, 2, (LPARAM)"Item3");

    hControl = CreateWindow("Button", "",
                           WS_CHILD|WS_VISIBLE|BS_AUTOCHECKBOX, 250, 10,
                           200, 20, hWnd, (HMENU)(UINT_PTR)6, NULL, NULL);
    SetWindowText(hControl, "Check Box");

    hControl = CreateWindow("Button", "",
                           WS_CHILD|WS_VISIBLE|BS_AUTORADIOBUTTON, 250, 70,
                           200, 20, hWnd, (HMENU)(UINT_PTR)7, NULL, NULL);
    SetWindowText(hControl, "Radio 1");

    hControl = CreateWindow("Button", "",
                           WS_CHILD|WS_VISIBLE|BS_AUTORADIOBUTTON,
                           250, 90, 200, 20, hWnd, (HMENU)(UINT_PTR)8, NULL, NULL);
}

```

```

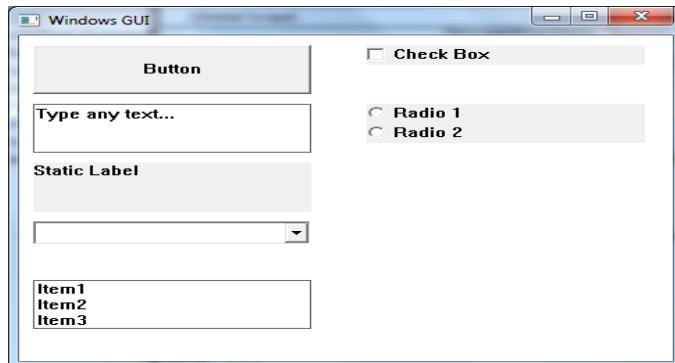
        SetWindowText(hControl, "Radio 2");
    }
    break;
case WM_COMMAND:
{
    WORD controlEvent = HIWORD(wParam);
    WORD controlID = LOWORD(wParam);
    HWND hControl = (HWND)lParam;
    switch(controlID)
    {
        case 1://button
        {
            if(controlEvent==BN_CLICKED)
                MessageBox(hWnd,"Button Clicked", "", MB_OK);
        }
        break;
    case 2://Edit
    {
        if(controlEvent==EN_CHANGE)
        {
            char text[1024];
            int length = GetWindowText(hControl, text, 1024);
            HDC hDC = GetDC(hWnd);
            TextOut(hDC, 550, 10, text, length);
            ReleaseDC(hWnd, hDC);
        }
        break;
    case 4://Cobmo
    {
        if(controlEvent==CBN_SELCHANGE)
        {
            char text[1024];
            int selection =
                SendMessage(hControl, CB_GETCURSEL, 0, 0);
            SendMessage(hControl, CB_GETLBTEXT,
                        (WPARAM)selection, (LPARAM)text);
            MessageBox(hWnd, text, "", MB_OK);
        }
        break;
    case 5://List
    {
        if(controlEvent==LBN_SELCHANGE)
        {
            char text[1024];
            int selection =
                SendMessage(hControl, LB_GETCURSEL, 0, 0);
            SendMessage(hControl, LB_GETTEXT,
                        (WPARAM)selection, (LPARAM)text);
            MessageBox(hWnd, text, "", MB_OK);
        }
        break;
    case 6://Check button
    {
        if(controlEvent==BN_CLICKED)
        {

```

```

        int check = (int)SendMessage(hControl,
                                      BM_GETCHECK, 0, 0);
        if(check)
            MessageBox(hWnd,"Button Checked", "", MB_OK);
        else
            MessageBox(hWnd,"Button Unchecked", "", MB_OK);
    }
}
break;
case 7://Radio button
{
    if(controlEvent==BN_CLICKED)
        MessageBox(hWnd,"Radio 1 Clicked", "", MB_OK);
}
break;
case 8://Radio button
{
    if(controlEvent==BN_CLICKED)
        MessageBox(hWnd,"Radio 2 Clicked", "", MB_OK);
}
break;
}
break;
}

```

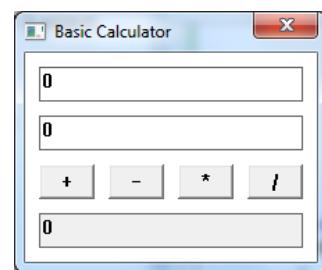


### 11.3.3. Basic Calculator

It is required to develop a simple calculator program that can add, subtract, multiply and divide two numbers. The program should have a GUI as shown in the figure.

User must fill the two edit boxes with the numbers then click one of the buttons.

When a button is clicked the related operation should be performed and the result should be displayed on the result box.



#### Main Code

LRESULT	WINAPI	WndProc	(HWND hWnd, UINT message, WPARAM wParam, LPARAM
---------	--------	---------	---

```

lParam)
{
    static char screenText[128] = "0";
    static char oldScreenText[128] = "0";
    static char oldSign = 0;
    static HWND hScreen = NULL;

    switch(message)
    {
        case WM_CREATE:
        {
            //GUI Creation and Program Initialization
            ...
        }
        break;
        case WM_COMMAND:
        {
            WORD controlEvent = HIWORD(wParam);
            WORD controlID = LOWORD(wParam);
            HWND hControl = (HWND)lParam;
            if(controlEvent==BN_CLICKED)
            {
                //Processing of Buttons Clicks
                ...
            }
        }
        break;
    }

    return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}

```

## GUI Creation and Program Initialization Code

```

int extraWidth = 2 * GetSystemMetrics(SM_CXFIXEDFRAME);
int extraHeight = 2 * GetSystemMetrics(SM_CYFIXEDFRAME) +
GetSystemMetrics(SM_CYCAPTION);
SetWindowLong(hWnd, GWL_STYLE,
    WS_OVERLAPPED|WS_BORDER|WS_CAPTION|WS_SYSMENU);
SetWindowPos(hWnd, HWND_TOP, 0, 0,
210 + extraWidth, 150 + extraHeight, SWP_SHOWWINDOW|SWP_NOMOVE);
SetWindowText(hWnd, "Basic Calculator");

hFirstEdit = CreateWindow("Edit", "0", WS_CHILD|WS_BORDER|WS_VISIBLE,
    10, 10, 190, 25, hWnd, (HMENU)(UINT_PTR)1, NULL, NULL);
hSecondEdit = CreateWindow("Edit", "0",
    WS_CHILD|WS_BORDER|WS_VISIBLE, 10, 45, 190, 25, hWnd,
    (HMENU)(UINT_PTR)2, NULL, NULL);
hResultEdit = CreateWindow("Edit", "0",
    WS_CHILD|WS_BORDER|WS_VISIBLE|ES_READONLY, 10, 115,
    190, 25, hWnd, (HMENU)(UINT_PTR)3, NULL, NULL);

HWND hButton;

hButton = CreateWindow("Button", "+", WS_CHILD|WS_VISIBLE, 10, 80,
    40, 25, hWnd, (HMENU)(UINT_PTR)'+', NULL, NULL);

```

```

hButton = CreateWindow("Button", "-", WS_CHILD|WS_VISIBLE, 60, 80,
                      40, 25, hWnd, (HMENU)(UINT_PTR)'-', NULL, NULL);
hButton = CreateWindow("Button", "*", WS_CHILD|WS_VISIBLE, 110, 80,
                      40, 25, hWnd, (HMENU)(UINT_PTR)'*', NULL, NULL);
hButton = CreateWindow("Button", "/", WS_CHILD|WS_VISIBLE, 160, 80,
                      40, 25, hWnd, (HMENU)(UINT_PTR)'/', NULL, NULL);

```

Processing of Buttons Clicks

```

double x, y, z = 0;
char text[128];

GetWindowText(hFirstEdit, text, 128);
x = atof(text);

GetWindowText(hSecondEdit, text, 128);
y = atof(text);

if(controlID=='/' && y == 0)
{
    strcpy(text, "Nan... ");
}
else
{
    switch(controlID)
    {
        case '+': z = x + y; break;
        case '-': z = x - y; break;
        case '*': z = x * y; break;
        case '/': z = x / y; break;
    }
    sprintf(text, "%f", z);
}
SetWindowText(hResultEdit, text);

```

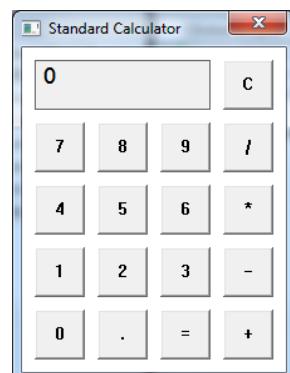
#### 11.3.4. Standard Calculator

In this example we will construct the standard calculator as shown in the following figure.

The calculator consists of 17 button and one edit box.

Standard calculator has following logic:

- Initially or after pressing the clear button “C” the calculator displays “0”.
- User can provide either integer or real numbers.
- User must not supply more than one “.” digit.
- User cannot provide other zeros if the displayed number is “0”.



- Whenever an operation button (+,-,\*,/) is clicked, you will have following situations:
  - This is the first time: The displayed value and the clicked operation must be stored internally.
  - Previous operation and value are available internally: Perform the stored operation over the stored value and the new value, and then, Store the result and the new operation internally.
- Whenever the “=” button clicked, you will have following situations:
  - Previous operation and value are available internally: Perform the stored operation over the stored value and the new value, and then displays the result.
  - There is no previous operation: displays the existing value as the result.
- If the user divides a number by zero, the program should displays “NaN...” and stop further operations until the user press the clear button “C”. (NaN: Not a Number)

Following code implements above logic:

Main Code

```
LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static char screenText[128] = "0";
    static char oldScreenText[128] = "0";
    static char oldSign = 0;
    static HWND hScreen = NULL;

    switch(message)
    {
        case WM_CREATE:
        {
            //GUI Creation and Program Initialization
            ...
        }
        break;
        case WM_COMMAND:
        {
            WORD controlEvent = HIWORD(wParam);
            WORD controlId = LOWORD(wParam);
            HWND hControl = (HWND)lParam;
            if(controlEvent==BN_CLICKED)
            {
                //Processing of Buttons Clicks
                ...
            }
        }
        break;
    }

    return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}
```

The **screenText** variable is used to prepare the current display contents. The (**oldScreenText**, **oldSign**) variables used to store old screen contents and the old sign.

GUI Creation and Program Initialization Code

```

int extraWidth = 2 * GetSystemMetrics(SM_CXFIXEDFRAME);
int extraHeight = 2 * GetSystemMetrics(SM_CYFIXEDFRAME) +
GetSystemMetrics(SM_CYCAPTION);
SetWindowLong(hWnd, GWL_STYLE,
WS_OVERLAPPED|WS_BORDER|WS_CAPTION|WS_SYSMENU);

SetWindowPos(hWnd, HWND_TOP, 0, 0, 210 + extraWidth,
260 + extraHeight, SWP_SHOWWINDOW|SWP_NOMOVE);
SetWindowText(hWnd, "Standard Calculator");

hScreen = CreateWindow("Edit", "0",
WS_CHILD|WS_BORDER|WS_VISIBLE|ES_READONLY,
10, 10, 140, 40, hWnd, (HMENU)(UINT_PTR)1, NULL, NULL);
HFONT hFont = WGCreateFont("Courier New", 14, FS_BOLD, 0);
SendMessage(hScreen, WM_SETFONT, (WPARAM)hFont, 0);

HWND hButton;

hButton = CreateWindow("Button", "C", WS_CHILD|WS_VISIBLE, 160, 10,
40, 40, hWnd, (HMENU)(UINT_PTR)' ', NULL, NULL);

hButton = CreateWindow("Button", "7", WS_CHILD|WS_VISIBLE, 10, 60,
40, 40, hWnd, (HMENU)(UINT_PTR)'7', NULL, NULL);
hButton = CreateWindow("Button", "8", WS_CHILD|WS_VISIBLE, 60, 60,
40, 40, hWnd, (HMENU)(UINT_PTR)'8', NULL, NULL);
hButton = CreateWindow("Button", "9", WS_CHILD|WS_VISIBLE, 110, 60,
40, 40, hWnd, (HMENU)(UINT_PTR)'9', NULL, NULL);
hButton = CreateWindow("Button", "/", WS_CHILD|WS_VISIBLE, 160, 60,
40, 40, hWnd, (HMENU)(UINT_PTR)'/', NULL, NULL);

hButton = CreateWindow("Button", "4", WS_CHILD|WS_VISIBLE, 10, 110,
40, 40, hWnd, (HMENU)(UINT_PTR)'4', NULL, NULL);
hButton = CreateWindow("Button", "5", WS_CHILD|WS_VISIBLE, 60, 110,
40, 40, hWnd, (HMENU)(UINT_PTR)'5', NULL, NULL);
hButton = CreateWindow("Button", "6", WS_CHILD|WS_VISIBLE, 110, 110,
40, 40, hWnd, (HMENU)(UINT_PTR)'6', NULL, NULL);
hButton = CreateWindow("Button", "*", WS_CHILD|WS_VISIBLE, 160, 110,
40, 40, hWnd, (HMENU)(UINT_PTR)'*', NULL, NULL);

hButton = CreateWindow("Button", "1", WS_CHILD|WS_VISIBLE, 10, 160,
40, 40, hWnd, (HMENU)(UINT_PTR)'1', NULL, NULL);
hButton = CreateWindow("Button", "2", WS_CHILD|WS_VISIBLE, 60, 160,
40, 40, hWnd, (HMENU)(UINT_PTR)'2', NULL, NULL);
hButton = CreateWindow("Button", "3", WS_CHILD|WS_VISIBLE, 110, 160,
40, 40, hWnd, (HMENU)(UINT_PTR)'3', NULL, NULL);
hButton = CreateWindow("Button", "-", WS_CHILD|WS_VISIBLE, 160, 160,
40, 40, hWnd, (HMENU)(UINT_PTR)'-', NULL, NULL);

hButton = CreateWindow("Button", "0", WS_CHILD|WS_VISIBLE, 10, 210,
40, 40, hWnd, (HMENU)(UINT_PTR)'0', NULL, NULL);
hButton = CreateWindow("Button", ".", WS_CHILD|WS_VISIBLE, 60, 210,
40, 40, hWnd, (HMENU)(UINT_PTR)'.', NULL, NULL);
hButton = CreateWindow("Button", "=", WS_CHILD|WS_VISIBLE, 110, 210,
40, 40, hWnd, (HMENU)(UINT_PTR)'=', NULL, NULL);
hButton = CreateWindow("Button", "+", WS_CHILD|WS_VISIBLE, 160, 210,
40, 40, hWnd, (HMENU)(UINT_PTR)'+', NULL, NULL);

```

## Processing of Buttons Clicks

```

switch(controlID)
{
case '0':case '1':case '2':case '3':case '4':
case '5':case '6':case '7':case '8':case '9':
case '.':
{
    if(oldSign=='?' || oldSign=='=')break;

    char buttonText[2] = "0";
    buttonText[0] = controlID;

    BOOL hasDot = (strstr(screenText, ".") !=0);

    if(hasDot && controlID=='.')
        break;

    if(atof(screenText)==0 && controlID!='.' && !hasDot)
        strcpy(screenText, "");

    strcat(screenText, buttonText);
    SetWindowText(hScreen, screenText);
}
break;

case '+':case '-':case '*':case '/':case '=':
{
    if(oldSign=='?')break;

    if(oldSign!=0)
    {
        char newScreenText[24];

        double x = atof(oldScreenText);
        double y = atof(screenText);
        double z = y;
        if(oldSign=='/' && y==0)
        {
            strcpy(screenText,"Nan...");
            SetWindowText(hScreen, screenText);
            strcpy(oldScreenText,"");
            oldSign = '?';
            break;
        }
        switch(oldSign)
        {
        case '+': z = x + y; break;
        case '-': z = x - y; break;
        case '*': z = x * y; break;
        case '/': z = x / y; break;
        }
        sprintf(oldScreenText, "%f", z);
        strcpy(screenText,"0");
        oldSign = (char)controlID;
    }
}

```

```

    else
    {
        strcpy(oldScreenText, screenText);
        strcpy(screenText, "0");
        oldSign = (char)controlID;
    }

    if(oldSign=='=')
    {
        strcpy(screenText, oldScreenText);
        SetWindowText(hScreen, oldScreenText);
        strcpy(oldScreenText, "0");
    }
}
break;
case ' ':
{
    strcpy(screenText, "0");
    strcpy(oldScreenText, "0");
    SetWindowText(hScreen, screenText);
    oldSign = 0;
}
break;
}

```

## 11.4 Advanced GUI/GDI Programming Topics

### 11.4.1. Drawing Polylines and Polygons

Windows GDI offers Polyline and Polygon functions to draw custom shapes like triangles, hexagonal or any custom shapes. Polyline function used to draw open shapes, however Polygon function used to draw closed and filled shapes. Following example illustrates how to use those functions.

```

case WM_PAINT:
{
    HDC hDC = GetDC(hWnd);

    HPEN hPen = WGCCreatePen(RGB(255, 0, 0), 5, PS_SOLID);
    HPEN hOldPen = (HPEN)SelectObject(hDC, hPen);

    HBRUSH hBrush = WGCCreateBrush(RGB(0, 0, 255), HS_SOLID);
    HBRUSH hOldBrush = (HBRUSH)SelectObject(hDC, hBrush);

    POINT apoints[] = {{10,10}, {200, 10}, {100, 150}};
    Polyline(hDC, apoints, sizeof(apoints)/sizeof(POINT));

    POINT bpoints[] = {{250,10}, {450, 10}, {350, 150}};
    Polygon(hDC, bpoints, sizeof(bpoints)/sizeof(POINT));

    SelectObject(hDC, hOldBrush);
    DeleteObject(hBrush);

    SelectObject(hDC, hOldPen);
}

```

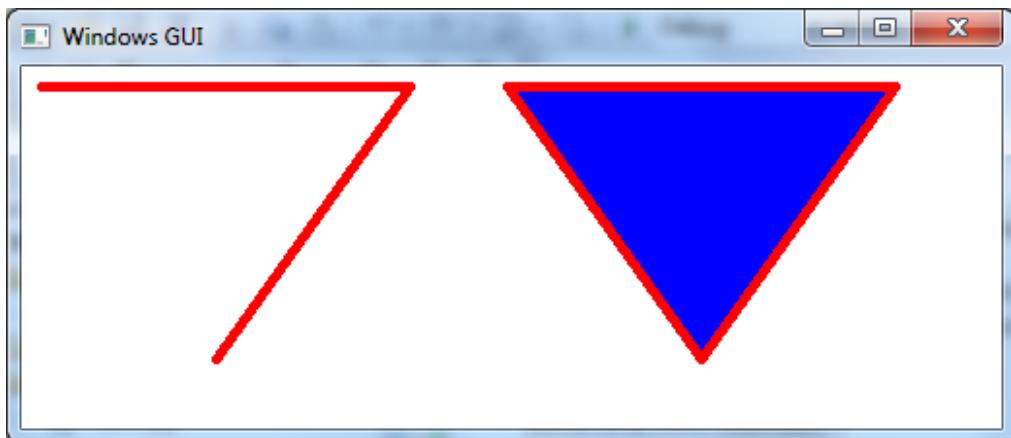
```

        DeleteObject(hPen) ;

        ReleaseDC(hWnd, hDC) ;
    }
break;
}

```

Above example produce the following graphics.



#### 11.4.2. Create Applications with Several Windows

Using windows APIs, you can create other windows in your application. Each window can have its own procedure and its own behavior. There are two main windows types:

1. Child Windows: created inside the parent window client area.
2. Popup Windows: created over the parent window area.

Following program uses the wrapper function **WGCreateWindow** to create either child or popup windows.

```

LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
    switch(message)
    {
        case WM_PAINT:
        {
            int width = WGGetClientWidth(hWnd) ;
            int height = WGGetClientHeight(hWnd) ;
            HDC hDC = GetDC(hWnd) ;
            Ellipse(hDC, 10, 10,
                    width-10, height-10) ;
            ReleaseDC(hWnd, hDC) ;
        }
        break;
    }
    return DefWindowProc(hWnd, message, wParam, lParam);
}

```

```

LRESULT WINAPI WndProc2(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
    switch(message)
    {
        case WM_PAINT:
        {
            int width = WGGetClientWidth(hWnd);
            int height = WGGetClientHeight(hWnd);
            HDC hDC = GetDC(hWnd);
            Rectangle(hDC, 20, 20,
                       width-20, height-20);
            ReleaseDC(hWnd, hDC);
        }
        break;
    }
    return DefWindowProc(hWnd, message, wParam, lParam);
}

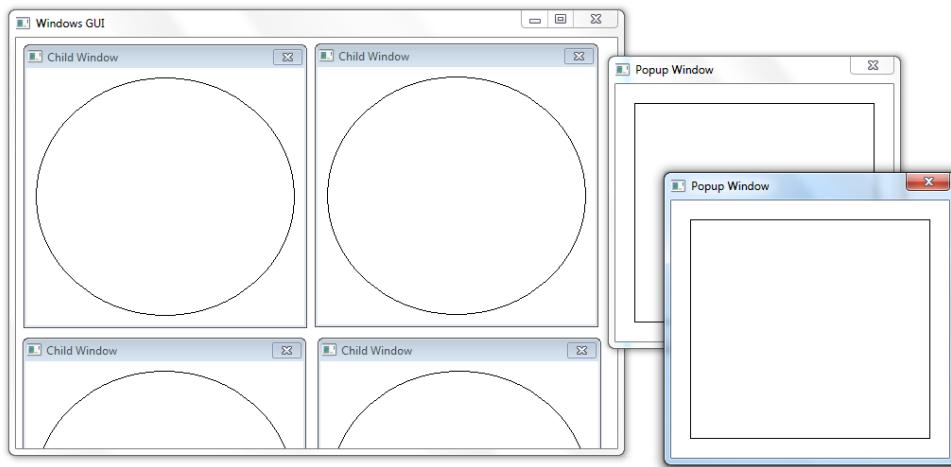
LRESULT WINAPI WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    switch(message)
    {
        case WM_LBUTTONDOWN:
        {
            int x = LOWORD(lParam);
            int y = HIWORD(lParam);
            HWND hNewWindow = WGCreateWindow("Child Window",
WS_CHILD|WS_VISIBLE|WS_CAPTION|WS_BORDER|WS_SYSMENU|WS_CLIPSIBLINGS,
x, y, 300, 300, hWnd, WndProc1);
        }
        break;
        case WM_RBUTTONDOWN:
        {
            POINT p;
            p.x = LOWORD(lParam);
            p.y = HIWORD(lParam);
            ClientToScreen(hWnd, &p);
            HWND hNewWindow = WGCreateWindow("Popup Window",
WS_POPUP|WS_VISIBLE|WS_CAPTION|WS_BORDER|WS_SYSMENU|WS_CLIPSIBLINGS,
p.x, p.y, 300, 300, hWnd, WndProc2);
        }
        break;
    }
    return WGDefaultWindowProc(hWnd, message, wParam, lParam);
}

```

The program should work as follows:

When the user clicks the left mouse button the program creates a **child** window and assigns to it the procedure `WndProc1` to handle its events. `WndProc1` paints an ellipse in its paint event.

When the user clicks the right mouse button the program creates a **popup** window and assigns to it the procedure WndProc2 to handle its events. WndProc2 paints a rectangle in its paint event.



The properties of the new window is controlled by the style parameters following values are used.

- **WS\_CHILD:** The window type is child.
- **WS\_POPUP:** The window type is popup.
- **WS\_VISIBLE:** The window is initially visible.
- **WS\_CAPTION:** The window has a caption.
- **WS\_BORDER:** The window has a border.
- **WS\_SYSMENU:** The window has system menu at the caption.
- **WS\_CLIPSIBLINGS:** Protects the drawing contents from being erased while other windows is passing over.

## 11.5 Exercises

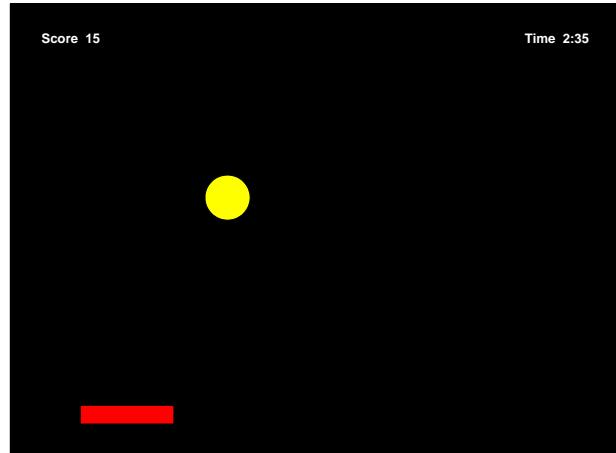
Q1.

Write a simple painting program, the user can change the brush shape and color using the keyboard.

Q2.

Write a simple computer game. Following figure illustrate the game screen, the screen consists of two items the ball and the plate. The ball is bouncing at the screen edges. The user can move the plate horizontally using left and right arrows at the keyboard. The plate cannot go outside the screen borders. When the ball touches the plate it bounces. The user must move the plate in order to reverse the ball direction before it reaches the bottom border.

If the ball reaches the bottom border the user lose, if the user succeeds to bounce the ball his or her score increased by one. The total score is displayed at the top-left corner of the screen. The spent time in minutes and seconds is displayed at the top-right corner. When the total game time reach 5 minutes, the game ends and the score is displayed at the middle of the screen.



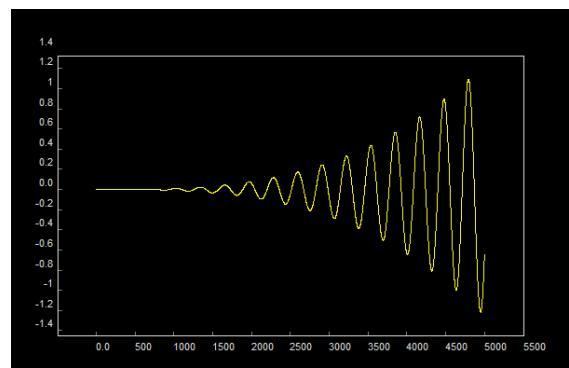
Q3.

Write a program that draws the following equation:

$$y = \sin\left(\frac{x}{5}\right) \times x^3 - 15x^2 + x - 5$$

$$0 \leq x \leq 1000$$

The program must fill the whole window. Also it must track the window resizing and squeeze or stretch the drawing as needed.



## Appendix A

“wingui.h”

```
#pragma once;

#undef UNICODE

#include "windows.h"
#include "stdio.h"

#define HS_SOLID 10 //soild brush
#define HS_NULL 11 //empty brush

HBRUSH WGCreateBrush(COLORREF color, int style)
{
    HBRUSH hBrush;

    switch(style)
    {
        case HS_SOLID: hBrush = CreateSolidBrush(color); break;
        case HS_NULL: hBrush = (HBRUSH)GetStockObject(NULL_BRUSH); break;
        default: hBrush = CreateHatchBrush(style, color); break;
    }

    return hBrush;
}

HPEN WGCreatePen(COLORREF color, int thickness, int style)
{
    return CreatePen(style, thickness, color);
}

#define FS_BOLD 0x00000001      //Bold font
#define FS_ITALIC 0x00000002    //Italic font
#define FS_UNDERLINE 0x00000004  //Underline font
#define FS_STRIKOUT 0x00000008  //Strick Out font

HFONT WGCreateFont(const char* name, int size, unsigned int options, int rotation)
{
    return CreateFont(-size, 0, rotation, rotation,
(options&FS_BOLD)?FW_BOLD:FW_NORMAL, (options&FS_ITALIC)?TRUE:FALSE,
(options&FS_UNDERLINE)?TRUE:FALSE, (options&FS_STRIKOUT)?TRUE:FALSE, DEFAULT_CHARSET,
OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH | FF_ROMAN,
name);
}

LRESULT WGDefaultWindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_DESTROY:
            {
                HWND hParentWnd = GetParent(hWnd);
                if(hParentWnd==NULL)
                    PostQuitMessage(0);
            }
            break;
    }

    return DefWindowProc(hWnd, message, wParam, lParam);
}

int WGCreateMainWindow(HINSTANCE hInstance, int nCmdShow, WNDPROC wndProc)
{
    // Initialize global strings
    WNDCLASSEX wndClassEx = {sizeof(WNDCLASSEX), CS_HREDRAW | CS_VREDRAW |
CS_DBCLKS, wndProc, 0, 0, hInstance, LoadIcon(NULL, IDI_APPLICATION), LoadCursor(NULL,
IDC_ARROW), (HBRUSH)(COLOR_WINDOW+1), NULL, "wingui", NULL};
    RegisterClassEx(&wndClassEx);

    // Perform application initialization:
    HWND hWnd = CreateWindow(wndClassEx.lpszClassName, "Windows GUI",

```

```

WS_OVERLAPPEDWINDOW|WS_CLIPCHILDREN, CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL,
hInstance, NULL);
    if (hWnd == NULL) return FALSE;

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    // Main message loop:
    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return (int) msg.wParam;
}

void WGPrintf(HDC hDC, int x, int y, const char* format, ...)
{
    int length;
    char buffer[1024];

    va_list arg;
    va_start(arg, format);
    length = vsprintf_s(buffer, sizeof(buffer), format, arg);
    va_end(arg);

    if(length>0)TextOut(hDC, x, y, buffer, length);
}

HWND WGCreateWindow(LPCSTR windowName, DWORD dwStyle, int x, int y, int nWidth, int
nHeight, HWND hWndParent, WNDPROC wndProc)
{
    char classname[128];
    sprintf_s(classname, 128, "wc%d", wndProc);

    // Initialize global strings
    WNDCLASSEX wndClassEx = {sizeof(WNDCLASSEX), CS_HREDRAW | CS_VREDRAW |
CS_DBLCLKS | CS_OWNDC, wndProc, 0, 0, NULL, LoadIcon(NULL, IDI_APPLICATION),
LoadCursor(NULL, IDC_ARROW), (HBRUSH)(COLOR_WINDOW+1), NULL, classname, NULL};
    ATOM RC = RegisterClassEx(&wndClassEx);

    // Perform application initialization:
    HWND hNewWindow = CreateWindow(wndClassEx.lpszClassName, windowName, dwStyle,
x, y, nWidth, nHeight, hWndParent, NULL, NULL, NULL);

    UpdateWindow(hNewWindow);

    return hNewWindow;
}
int WGGetClientWidth(HWND hWnd)
{
    RECT R;
    GetClientRect(hWnd, &R);

    return R.right-R.left;
}
int WGGetClientHeight(HWND hWnd)
{
    RECT R;
    GetClientRect(hWnd, &R);

    return R.bottom-R.top;
}

```