BIRZEIT UNIVERSITY

Comp.142/132/230

Programming with C

LABORATORY WORK BOOK

Compiled and Prepared by:

Mr. Nael

Mr. Wahbeh Musa

Mr. Ahmad Abu Snani

Miss. Deema Shita

Approved by:

Computer Science Department

2017

Acknowledgement

The material included in this manual has been entirely adopted from the following references:

1. Laboratory work book, Department of Electronic Engineering, N.E.D. University of Engineering & Technology, Karachi – Pakistan.
2. Problem Solving and program Design in C, by Jeri R. Hanly and Elliot B. Koffman, fourth Edition.

Introduction

This work book is especially designed to help the students to generate their own logic for the accomplishment of the assigned tasks. Every lab is provided with the syntax of the statements or commands which will be used to make the programs for exercises. In order to facilitate for the students some program segments are also provided explaining the use of commands. For a wide scope of usage of the commands several examples are also given so that the students can understand how to use the commands. The conditions, limitations and memory allocation are also mentioned where necessary.

This work book starts the programming of C Language from scratch and covers most of the programming structures of C-Language. For some commands or structures more than one lab is designed so that the students can thoroughly understand their use.

Contents

Table of Contents

Lab No.01: Numbering System

Objective

Numbering System

Theory

There are many ways to represent the same numeric value. Long ago, humans used sticks to count, and later learned how to draw pictures of sticks in the ground and eventually on paper. So, the number 5 was first represented as: **| | | | |** (for five sticks).

Later on, the Romans began using different symbols for multiple numbers of sticks: **| | |** still meant three sticks, but a **V** now meant five sticks, and an **X** was used to represent ten of them!

Using sticks to count was a great idea for its time. And using symbols instead of real sticks was much better. One of the best ways to represent a number today is by using the modern decimal system. Why? Because it includes the major breakthrough of using a symbol to represent the idea of counting **nothing**. About 1500 years ago in India, **zero** (**0**) was first used as a number! It was later used in the Middle East as the Arabic, sifr. And was finally introduced to the West as the Latin, zephiro. Soon you'll see just how valuable an idea this is for all modern number systems.

Decimal System

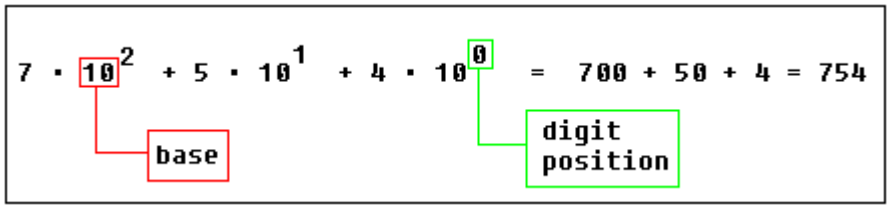
Most people today use decimal representation to count. In the decimal system there are 10 digits:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

These digits can represent any value, for example:

754.

The value is formed by the sum of each digit multiplied by the base (in this case it is 10 because there are 10 digits in decimal system) in power of digit position (**counting from zero**):



Position of each digit is very important! for example if you place "7" to the end: 547
it will be another value:

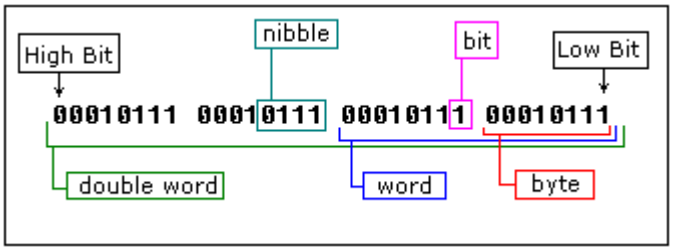
Binary System

Computers are not as smart as humans are (or not yet), it's easy to make an electronic machine with two states: on and off, or 1 and 0.

Computers use binary system, binary system uses 2 digits:

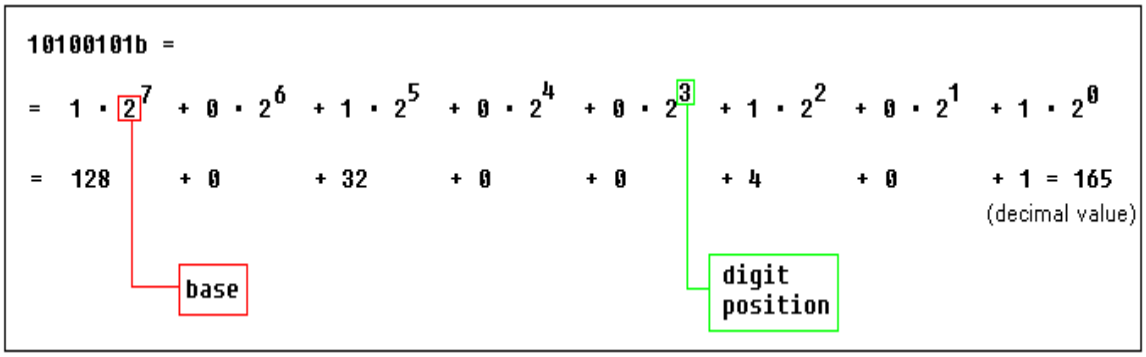
0, 1 And thus the base is 2.

Each digit in a binary number is called a **BIT**, 4 bits form a **NIBBLE,** 8 bits form a **BYTE**, two bytes form a **WORD**, and two words form a **DOUBLE WORD** (rarely used):



There is a convention to add "b" in the end of a binary number, this way we can determine that 101b is a binary number with decimal value of 5.

The binary number 10100101b equals to decimal value of 165:



Hexadecimal System

Hexadecimal System uses 16 digits:

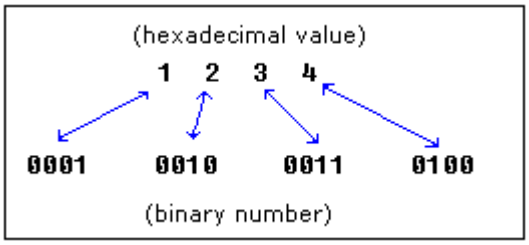
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

And thus the base is 16.

Hexadecimal numbers are compact and easy to read.

It is very easy to convert numbers from binary system to hexadecimal system and vice-versa, every **nibble** (4 bits) can be converted to a hexadecimal digit using this table:

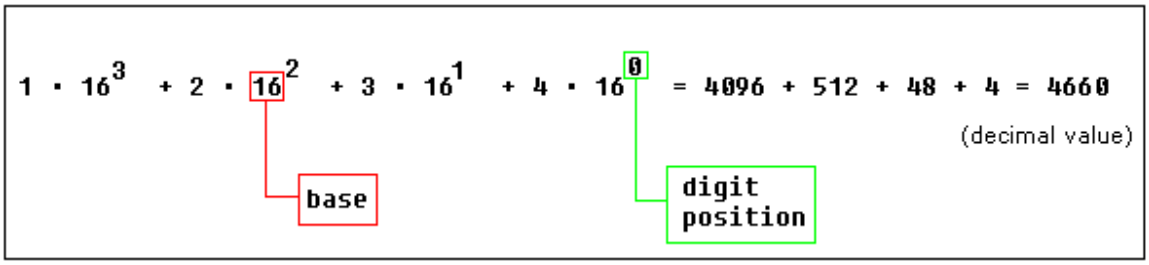
Decimal (base 10)	Binary (base 2)	Hexadecimal (base 16)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F



There is a convention to add "**h**" in the end of a hexadecimal number, this way we can determine that 5Fh is a hexadecimal number with decimal value of 95.

We also add "0" (zero) in the beginning of hexadecimal numbers that begin with a letter (A..F), for example 0E120h.

The hexadecimal number 1234h is equal to decimal value of 4660:



EXERCISE:

Binary to Decimal Conversion

1. Convert the following 8-bit numbers from binary to decimal.

    a. 11100001_2
    b. 10010010_2
    c. 01011010_2

2. Determine whether the following statements are true or false.

    a. $0010_2 < 8_{10}$
    b. $10_{10} > 110\ 2$
    c. $1100_2 = 1100_{10}$

3. Convert the following numbers to their binary equivalents.
    a. 77_{10}
    b. 178_{10}
    c. 295_{10}

Converting Fr*actions*

4. Convert the following numbers to their binary equivalents.
    a. 12.625_{10}
    b. 28.04_{10}
    c. 72.022_{10}

    2^5

5. Use the 32-bit floating representation to represent the binary numbers you have got from the previous question and show how it will be represented in the memory.

Adding Two Binary Numbers

6. Add the following 8-bit binary numbers.
    a. $100100100_2 + 010110010_2$
    b. $101101101_2 + 101001001_2$
    c. $111111111_2 + 111100000_2$

Two's Complement

7. Convert the following decimal numbers to binary using 8-bit 2's complement representation.
    a. -4_{10}
    b. -17_{10}
    c. -22_{10}

Subtraction with Two's Complement

8. Solve each of the following 8-bit subtraction problems using 2's complement representation.
    a. $11111000_2 - 17_{10}$
    b. $11001100_2 - 128_{10}$
    c. $01010101_2 - 111_{10}$

Hexadecimal and Octal Numbers

9. Convert the following numbers to decimal.

    a. 17_8
    b. 171_{16}
    c. $DE8_{16}$
    d. 101_8
    e. 37.7_8
    f. $B2.F_{16}$

ASCII Code Representation

10. Using the even parity bit to represent the following sentence in memory (Hexadecimal)?

    Welcome to BZU.

Lab No.02: Algorithms

Objective

Design computer algorithms

Theory

An algorithm (pronounced AL-go-rith-um) is a procedure or formula for solving a problem. The word derives from the name of the mathematician, Mohammed ibn-Musa al-Khwarizmi, who was part of the royal court in Baghdad and who lived from about 780 to 850. Al-Khwarizmi's work is the likely source for the word algebra as well.

To make a computer do anything, you have to write a computer program. To write a computer program, you have to tell the computer, step by step, exactly what you want it to do. The computer then "executes" the program, following each step mechanically, to accomplish the end goal.

When you are telling the computer what to do, you also get to choose how it's going to do it. That's where computer algorithms come in. The algorithm is the basic technique used to get the job done. Let's follow an example to help get an understanding of the algorithm concept.

Let's say that you have a friend arriving at the airport, and your friend needs to get from the airport to your house. Here are four different algorithms that you might give your friend for getting to your home:

The taxi algorithm:

1. Go to the taxi stand.
2. Get in a taxi.
3. Give the driver my address.

The call-me algorithm:

1. When your plane arrives, call my cell phone.
2. Meet me outside baggage claim.

The rent-a-car algorithm:

1. Take the shuttle to the rental car place.
2. Rent a car.
3. Follow the directions to get to my house.

The bus algorithm:

1. Outside baggage claim, catch bus number 70.
2. Transfer to bus 14 on Rukab Street.
3. Get off on Jerusalem street.
4. Walk two blocks north to my house

All four of these algorithms accomplish exactly the same goal, but each algorithm does it in completely different way. Each algorithm also has a different cost and a different travel time. Taking a taxi, for example, is probably the fastest way, but also the most expensive. Taking the bus is definitely less expensive, but a whole lot slower. You choose the algorithm based on the circumstances.

Pseudocode

Pseudocode is a kind of structured English for describing algorithms. It allows the designer to focus on the logic of the algorithm without being distracted by details of language syntax. At the same time, the pseudocode needs to be complete. It describes the entire logic of the algorithm so that implementation becomes a rote mechanical task of translating line by line into source code.

Example: Calculate the class average for 10 students:

Pseudocode:

```
Set total to zero

Set counter to one

While counter is less than or equal to ten

        Input the next grade

        Add the grade into the total

ENDWhile

Set the average to the total divided by ten

Print the class average
```

EXERCISES:

1. Write an algorithm to calculate the average of a set of students (we don't know their count). When -1 is entered this means that algorithm should stop receiving data, and print the output.

2. Write an algorithm to print the number of passes and the number of failures in a class of **n** students (n is **not** defined by the user), also let the program print the passes percentage. When -1 is entered this means that algorithm should stop receiving data, and print the output.

3. Write an algorithm to calculate and print the nth power of a number.
For example: if the user enters the number= 8 and n=3, the algorithm should calculate the value of 8^3= 8*8*8 and print the result which is 512.

4. Write an algorithm to print the sum of the following series: Sum=1/3 +3/7 +5/11+ 7/15+..+n/(2n+1).

5. Write an algorithm to check if the number is prime or not.

HW:

1. Write an Algorithm to do the function of a simple calculator which should be able to do +,-,*,/,% operations. For example : if the user insert 8*5, the program should print 40.

2. Write an Algorithm to find the maximum number of a set of **4** integers. Ask the user to enter numbers of the set.

Lab No.03: C Building Blocks

Objective

C Building Blocks

Theory

In any language there are certain building blocks:

- Constants
- Variables
- Operators
- Methods to get input from user (scanf(), getch() etc.)
- Methods to display output (Format Specifiers, Escape Sequences etc.) etc.

Format Specifiers

Format Specifiers tell the **printf** statement where to put the text and how to display the text.

The various format specifiers are:

%d => integer

%c => character

%f => float

…

Variables and Constants

If the value of an item is to be changed in the program then it is a variable. If it will not change then that item is a constant. The various variable types (also called data type) in C are: **int, float, char, long, double, …** they are also of the type **signed or unsigned**.

Escape Sequences

Escape Sequence causes the program to escape from the normal interpretation of a string, so that the next character is recognized as having a special meaning. The back slash "\" character is called the **Escape Character**". The escape sequence includes the following:

\n => new line

\t => tab

\b => back space

\r => carriage return

\" => double quotations

\\ => back slash etc.

Taking Input from the User

The input from the user can be taken by the following techniques: **scanf( ), getch( ),**

Operators

There are various types of operators that may be placed in three categories:

Basic: + ,-,*, /, %

Assignment: =, += ,-= ,*=, /=, %=

Relational: <, > ,<=, >=, == ,!=

Logical: && , || , !

EXERCISE:

1. Write a program which shows the function of each escape sequence character.

   e.g.

   printf("the new line is inserted like \n this");

   printf("the tab is inserted like \t this");

2. What will be the output of the following statements:

   double n=17.8295;

   printf("%4.2lf",n);

3. What will be the output of the mix mode use of integers and float:

   int x,y;

   float a,b;

   x=8/5;

   y=8.0/5;

   a=8/5;

   b=8.0/5;

   printf(" x=%d \n y=%d \n a=%f \n b= %f",x,y,a,b);

4. Write down C statements to perform the following operations:

   i. $x = a_1^2 + 2a_1b_1 + b_1^2$

   ii. $Z = \frac{4.2(x+y)/z - 0.25x/(y+z)}{(x+y)^2}$

5. Write a program that find the sum of three-digit number

   For example: if user insert 178 the program should calculate 1+7+8 and print 16.

6. Write a program that print a reverse of three-digit number

   For example: if user insert 178 the program should print 871.

HW:

Write a program that add two for each digit of three-digit number

Hint: if the digit is 8 it will be 0 and if it is 9 it will be 1 after addition.

For example: if user insert 178 the program should print 390.

Lab No.04: Functions

Objective

Functions in C-Language programming

Theory

Functions are used normally in those programs where some specific work is required to be done repeatedly and looping fails to do the same.

Three things are necessary while using a function.

i. Declaring a function or prototype:

The general structure of a function declaration is as follows:

```
return_type function_name(arguments);
```

Before defining a function, it is required to declare the function i.e. to specify the function prototype. A function declaration is followed by a semicolon ';'. Unlike the function definition only data type need to be mentioned for arguments in the function declaration.

ii. Calling a function:

The function call is made as follows:

```
variable = function_name(arguments);
```

ii. Defining a function:

All the statements or the operations to be performed by a function are given in the function definition which is normally given at the end of the program outside the main.

Function is defined as follows

```
return_type function_name (arguments)
{
    Statements;
}
```

There are four types of functions depending on the return type and arguments:

- Functions that take nothing as arguments and return nothing.
- Functions that take arguments but return nothing.
- Functions that do not take arguments but return something.
- Functions that take arguments and return something.

A function that returns nothing must have the return type "void**". If nothing is specified then the return type is considered as "int".**

EXERCISE:

1. Write a program which takes two integers as inputs and prints their sum. Use a function to find the sum.

2. Write a program which takes two points as inputs and prints the distance between them. Use a function to find the distance.
   $d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$

   Note: use the math library to find the square root -> sqrt(double).

3. Write a program to find:

   a. Surface area (**A= 2 r π**).

   b. volume(**v=r ^{2} π h**).

   of a cylinder using functions. One for the area and the other for the volume. The program should take the height and the radius as inputs then call functions to calculate the values and print the results in the main scope .

   Note: use math library to find **r ^{2}** -> pow(double, double).

4. Write a program that compute and display the area of the square in cm and inches. The relevant formulas are:

   Area = length * length

   Cm =2.54 * inch

   You need to build two functions in your program. Function **area** that takes a length of a square (in inches) as input data and compute and return the area of square measured in inches. Function **inch_cm** that takes area in inches as input then convert and return the area in cm.

HW:

Write a program to find the area of a rectangle. You need to build a function **rect_area** that takes 2 numbers as inputs and return the area of rectangle.

Lab No.05: Conditional Statements

Objective

Decision making if and if-else structure, the Switch case and conditional operator.

Theory

Normally, your program flows along line by line in the order in which it appears in your source code. But, it is sometimes required to execute a particular portion of code only if certain condition is true; or false i.e. you have to make decision in your program. There are three major decision making structures. **The 'if' statement, the if-else statement, and the switch statement.** Another less commonly used structure is the **conditional operator**.

The if statement

The if statement enables you to test for a condition (such as whether two variables are equal) and branch to different parts of your code, depending on the result or the conditions with relational and logical operators are also included..

The simplest form of an if statement is:

```
if (expression)
        statement;
```

The if-else statement

Often your program will want to take one branch if your condition is true, another if it is false. If only one statement is to be followed by the if or else condition then there is no need of parenthesis. The keyword else can be used to perform this functionality:

```
if (expression)
{
        statement/s;
```

```
}
else
{
        statement/s;
}
```

The switch Statement

Unlike if, which evaluates one value, switch statements allow you to branch on any of a number of different values. There must be break at the end of the statements of each case otherwise all the preceding cases will be executed including the default condition. The general

form of the switch statement is:

```
switch (identifier variable)
{
case  identifier One :     statement; . . .
                                    break;
case identifier Two   :    statement; . . .
                                    break;
....
case identifier N       :  statement; . . .
                                    break;
default              :     statement; . . .
}
```

Conditional Operator

The conditional operator **(? :)** is C's only ternary operator; that is, it is the only operator to take three terms.

The conditional operator takes three expressions and returns a value:

(expression1) ? (expression2) : (expression3)

It replaces the following statements of if else structure

```
If(a>b)
        c=a;
else
        c=b;
```

can be replaced by

```
c=(a>b)? a : b
```

This line is read as "If expression 1 is true, return the value of expression 2; otherwise, return the value of expression 3." Typically, this value would be assigned to a variable.

EXERCISE:

1. Write a program that inputs a grade of a student and determines if it is a pass (60 or more) or a fail.

2. Write a program that inputs an integer and determines if it is divisible by 8 or not.

3. Write a program which takes a character as input and checks whether it is a consonant or a vowel.

4. Write a program which takes three sides a, b and c of a triangle as input and calculates its perimeter if these conditions are satisfied a+b > c, b+c > a and a+c > b.

(Help: Perimeter = a+b+c), use a function to calculate the perimeter.

5. Write a program to make a simple calculator which should be able to do +,-,*,/,% operations. Use switch statement. Make sure division at **0** isn't allow.

6. Write a program which takes **3** integers as input and prints the smallest one.

HW:

Write a program to count the odd numbers. You need to build a function **count_odd** that takes **4** numbers as inputs and return how many number of them are odd.

Lab No.06: Looping

Objective

Looping constructs in C-Language, and Nested looping

Theory

Types of Loops

There are three types of Loops:

1) for Loop

    i. simple for loop

    ii. nested for loop

2) while Loop

    i. simple while loop

    ii. nested while loop

3) do - while Loop

    i. simple do while loop

    ii. nested do while loop

Nesting may extend these loops.

The for Loop

for (initialize(optional) ; condition(compulsory) ; increment(optional))

{

    Body of the Loop;

}

This loop runs as long as the condition in the center is true. Note that there is no semicolon after the "for" statement. If there is only one statement in the "for" loop then the braces may be removed. If we put a semicolon after the for loop instruction then that loop will not work for any statements.

The Nested for Loop

```
for (initialize ; condition ; increment)
{
        Body of the loop;
        for (initialize ; condition ; increment)
        {
                Body of the loop;
        }
        Body of the loop;
}
```

The inner loop runs as many times as there is the limit of the condition of the external loop. This loop runs as long as the condition in the parenthesis is true. We can nest many loops inside as needed.

The while Loop

```
while ( condition is true )
{
        Body of the Loop;
}
```

This loop runs as long as the condition in the parenthesis is true. Note that there is no semicolon after the "while" statement. If there is only one statement in the "while" loop then the braces may be removed.

The nested while Loop

```
while (condition is true)
{
        while (condition is true)
        {
```

```
            Body of the loop;
        }
        Body of the loop;
}
```

The inner loop runs as many times as there is the limit of the condition of the external loop. This loop runs as long as the condition in the parenthesis is true. We can nest many loops inside as needed.

The do-while Loop

```
do
{
        Body of the Loop;
} while ( condition is true);
```

This loop runs as long as the condition in the parenthesis is true. Note that there is a semicolon after the "while" statement. The difference between the "while" and the "do-while" statements is that in the "while" loop the test condition is evaluated before the loop is executed, while in the "do" loop the test condition is evaluated after the loop is executed. This implies that statements in a "do" loop are executed at least once. However, the statements in the "while" loop are not executed if the condition is not satisfied.

The Nested do-while Loop

```
do
{
        do
        {
                body of the loop;
        } while (condition is true);
        body of the loop;
} while (condition is true);
```

This loop runs as long as the condition in the parenthesis is true. Note that there is a semicolon after the “while” statement. The difference between the “while” and the “do-while” statements is that in the “while” loop the test condition is evaluated before the loop is executed, while in the “do” loop the test condition is evaluated after the loop is executed. This implies that statements in a “do” loop are executed at least once. The inner loop runs as many times as there is the limit of the condition of the external loop. This loop runs as long as the condition in the parenthesis is true. We can nest many loops inside as needed.

EXERCISE:

1. Write down the output of the following program statement

```
for (i=1; i<=10;i++)
        printf("%d \n",i);
```

2. Write a program that prints the first **10** positive numbers that are divisible by 5.

3. Write a program which will read pairs of integers from the user in a loop. Each time it reads a new pair, it checks if the first integer divides the second (second integer % first integer = 0). When the program reads such a pair it will exit the loop.

4. Write down the output of the following program statements

```
for (int a=1,j=1; j<=3;j++)
{
        for (i=1; i<=3;i++)
        {
                printf("%d\nj= ",j);
                printf("%d\ni= ",i);
        }
        printf("%d\n++a = ",++a);
        printf("%d\na++ = ",a++);
}
```

5. Write a program which takes an integer as input and checks whether it's prime or not.

6. Write a program to print a series of the first **10** positive prime numbers.

7. Write a program which prints the following pattern up to 10 lines

0

101

21012

3210123

n.....3210123.....n

8. Write a program to print the following pattern up to z only

a b c d e

f g h i j

k l m n o

p q r s t

u v w x y

z

HW:

Write a program that takes **n** of integer numbers as input and the count of even numbers .

Lab No.07: Pointers

Objective

Modular programming and pointers.

Theory

In lab No.04 you learned how to write the separate functions of a program. The functions correspond to the individual steps in a problem solution. You also learned how to provide inputs to a function and how to return a single output. In this lab you will complete your study of functions, learning how to connect the functions to create a program system- an arrangement of parts that makes your program operate like a stereo system as it passes information from one function to another.

1. Parameters enable a programmer to pass data to functions and to return multiple results from functions. The parameter list provides a highly visible communication path between a function and its calling program. Using parameters enables a function to process different data each time it executes thereby making it easier to reuse the function in other programs.

2. Parameters may be used for input to a function, for output or sending back results, and for both input and output. An input parameter is used only for passing data into a function. The parameter's declared type is the same as the type of the data. Output and input/output parameters must be able to access variables in the calling function's data area so they are declared as pointers to the result data TYPES. The actual argument corresponding to an input parameter may be an expression or a constant; the actual argument corresponding to an output or input/output parameter must be the address of a variable.

3. The scope of an identifier dictates where it can be referenced. A parameter or local variable can be referenced anywhere in the function that declares it. A function name is visible from its declaration (the function prototype) to the end of the source file except within functions that have local variables of the same name.

EXERCISE

1. Determine the output of the following program:

   Note : %p used to print the address in hexadecimal.

```
int  main()
{
        int val=8;
         int *p;
        p=&val;
        *p=295;
         printf("%d\n",val);
         printf("%p\n",p);
         printf ("%d\n", *p);
        return 0;
}
```

Variable	Value
val	
p	
*p	

2. Determine the output of the following program:

```
int main()
{
    int x=5,y=7,z=8;
    int *p1,*p2,*p3;
    p1=&x;
    p2=&y;
    p3=&z;
    *p1=(*p2)*(*p3);
```

Variable	Value
x	
y	
z	
*P1	
*P2	
*P3	

```
    ++(*p1);
     (*p2)--;
    *p1=(*p2)+(*p3);
    *p2=(*p2)*(*p1);
    x=y+z;
    printf("%d\n",x);
    printf("%d\n",y);
    printf("%d\n",z);
    printf("%d\n",*p1);
    printf("%d\n",*p2);
    printf("%d\n",*p3);
    return 0;
}
```

3. Write a program to dispense change. The user enters the amount paid and the amount due. The program determines number of bills and coins you need to return from the following categories: 200, 100, 50 and 20 bills and 10, 5, 2, 1 and 0.5 coins. Write a function with nine output parameters that determines the quantity of each kind of coins and bills.

Bills	Quantity
200	2
100	1
50	1
20	0

Coins	Quantity
10	0
5	1
2	0
1	1
0.5	1

e.g. If user enters **556.5** NIS the output will be:

556.5 = 200*2 + 100*1 + 50*1 + 20*0 + 10*0 + 5*1 + 2*0 + 1*1 + 0.5*1

556.5= 400 + 100 + 50 + 0 + 0 + 5 + 0 + 1 + 0.5

HW:

Write a program which takes two integers and print their sum, subtraction, multiplication, and division. Use a function with four output parameters.

Lab No.08: One dimensional arrays

Objective

Arrays in C (one-dimensional)

Theory

An array is a collection of data storage locations, each of which holds the same type of data. Each storage location is called an element of the array. You declare an array by writing the type, followed by the array name and the subscript. The subscript is the number of elements in the array, surrounded by square brackets. For example:

long LongArray [25];

Declares an array of 25 long integers, named Long Array. When the compiler sees this declaration, it sets aside enough memory to hold all 25 elements. Because each long integer requires 4 bytes, this declaration sets aside 100 contiguous bytes of memory.

EXERCISE:

1. Write a program that declares two arrays of integers. Fill the first one from the user and the second is static, and exchanges their values.

2. Write a program which takes a string as input and then counts the total number of vowels in that string.

3. Write a program that takes 5 integers as input and prints the smallest integer and its location in the array.

4. Write a function which take two arrays and return an array contains the sum of them.

5. Write a program to pass an integer array of 10 elements to a function, which returns the same array after sorting it in descending order. Print the array.

HW:

The electric company charges according to the following rate schedule:

0.5 NIS per kilowatt-hour (kWh) for the homes.

0.8 NIS per kWh for commercial uses.

Write a function to compute the total charge for each customer. Write a main function to call the charge calculation function using the following data:

Customer Number	Kilowatt-hours Used
123/home	725
205/home	115
464/ commercial uses	600
596/ commercial uses	327

The program should print a three-column chart listing the customer number, the kilowatt hours used, and the charge for each customer. The program should also compute and print the number of customers, the total kilowatt hours used, and the total charges.

Lab No.09: Multidimensional Array

Objective

Arrays in C (Multidimensional) and string functions.

Theory

A Multidimensional array is a collection of data storage locations, each of which holds the same type of data. Each storage location is called an element of the array. You declare an array by writing the type, followed by the array name and the subscript. The subscript is the number of elements in the array, surrounded by square brackets. For example: int a [5] [10].

Declares an array of 50 integers, named a. Its declaration shows that array a comprises of 5 one dimensional arrays and each one dimensional array contains 10 elements. When the compiler sees this declaration, it sets aside enough memory to hold all 50 elements. Because each integer requires 2 bytes, this declaration sets aside 100 contiguous bytes of memory.

EXERCISE:

1. Write a program that adds up two **2x2** arrays and stores the sum in third array.

2. Write a program which takes names of five countries as input and prints them in alphabetical order.

3. Write and test a function hydroxide that returns a 1 for true if its string argument ends in the substring CL.

   Try the function hydroxide on the following data:

   HOCl MgOH2 NaCL NaOH C9H8O4 HCl

4. Write a program that takes data one line at a time and reverses the words of the line. For example,

   Input: worry less smile more

   Output: more smile less worry

HW:

Write a program that takes array of 10 students, fined how many students more than the average, and print it.

Lab No.10: Structures

Objective

Structures

Theory

If we want a group of same data type, we use an array. If we want a group of elements of different data types, we use structures. For Example: To store the names, prices and number of pages of a book you can declare three variables. To store this information for more than one book three separate arrays may be declared. Another option is to make a structure. No memory is allocated when a structure is declared. It simply defines the “form” of the structure. When a variable is made then memory is allocated. This is equivalent to saying that there is no memory for “int”, but when we declare an integer i.e. int var; only then memory is allocated.

The structure for the above mentioned case will look like

```
Struct books
{
        char bookname[20];
        float price;
        int pages;
}
struct book [50];
```

the above structure can hold information of 50 books.

EXERCISE

1. Write a program to maintain the library record for **50** books with book name, author's name, and edition, year of publishing and price of the book.

2. Write a program to make a tabulation sheet for a class of **20** students with their names, seat no's, marks, percentages and grades.

3. Consider the following structure definition:

```
struct Student
{
        char name [5];
        int section;
        float grade;
};
```

Write a program that declare a list (struct Student) with size > 0. Your Program will do the following:

(a) Raise all the grades in the class 5% to a maximum of 100.

(b) Give everyone in the class whose name starts with "Al" a grade of 100.

(c) Sort the list by grades, highest to lowest.

HW:

Write a program to add two distances (in inch-feet) system using structures.

Lab No.11: Files

Objective

Filing in C-Language

Theory

Data files

Many applications require that information be “written & read” from an auxiliary storage device.

This information is written in the form of **Data Files**.

Data files allow us to store information permanently and to access and alter that information whenever necessary.

Types of Data files

Standard data files. (stream I/O)

System data files. (low level I/O)

Standard I/O

Easy to work with, & have different ways to handle data.

Four ways of reading & writing data:

1. Character I/O.
2. String I/O.
3. Formatted I/O.
4. Record I/O.

File Protocol

1. fopen
2. fclose

fopen:

It is used to open a file.

Syntax:

fopen (file name, access-mode).

“r” open a file for reading only.

“w” open a file for writing.

“a” open a file for appending.

“r+” open an existing file for reading & writing.

“w+” open a new file for reading &writing.

“a+” open a file for reading & appending & create a new file if it does exist.

Example:

```
#include <stdio.h>
main
{
        FILE *fpt;
        fpt = fopen (“first.txt”,”w”);
        fclose(fpt);
}
```

Establish buffer area, where the information is temporarily stored before being transferred b/w the computer memory & the data file.

File is a special structure type that establishes the buffer area.

fpt is a pointer variable that indicates the beginning of buffer area.

fpt is called stream pointer.

fopen stands for File Open.

A data file must be opened before it can be created or processed.

Standard I/O:

Four ways of reading and writing data:

Character I/O.

String I/O.

Formatted I/O.

Record I/O.

Character I/O:

In normal C program we used to use getch, getchar and getche etc. In filling we use putc and getc.

putc ();

It is used to write each character to the data file.

Putc requires specification of the stream pointer *fpt as an argument.

Syntax:

putc (c, fp);

c = character to be written in the file.

fp = file pointer.

putch or putchar writes the I/P character to the consol while putc writes to the file.

Example (1):

Reading the Data File:

```
#include < stdio.h >
main ()
{
        FILE*fpt;
        char c;
        fpt = fopen (“star.dat”,” r”);
        if (fpt = = NULL)
                printf (“Error-can’t open”);
        else
        do
        {
                putchar (c= getc(fpt));
        } while (c! = ‘\n’);
        fclose (fpt);
}
```

Exercise:

1. What will be the output of the given program?

```
#include < stdio.h >

main ()

{

        FILE*fpt;

        char c;

        fpt = fopen ("star.dat"," w");          // a new file is made

        do

        {

                putc ((c= getchar ()), fpt);

        } while (c! = '\n');                    // or '\r'

        fclose (fpt);

}
```

2. Write a program which takes 5 integers as input from file and print the largest number on consol.

3. Write a program which takes 5 words as input and print them on file as a one statement.

HW:

Write a program which takes two points from file as inputs and prints the distance between them in another file. Use a function to find the distance.

$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$

Note: use the math library to find the square root -> sqrt(double).