

Programming Fundamentals CT-175

Dynamic Memory Allocation

Objectives

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context. Students will be representing solutions to different problems as flowcharts and pseudo code

The purpose of this lab is to enable students to

understand the basics of
problem solving in
programmatic context.

Students will be representing
solutions to different
problems as

flowcharts and pseudo code

The purpose of this lab is to
enable students to

understand the basics of
problem solving in
programmatic context.

Students will be representing

solutions to different problems as flowcharts and pseudo code

The objective of this lab is to enable students understand and develop the problem solving skills by using programming concepts. By the end of this lab, students should be able to present solutions to different programming problems by using pseudo-codes and flowcharts.

Tools Required

MS Word or Paint

Course Coordinator –

Instructor –

Lab Instructor –

Department of Computer Science and Information Technology

NED University of Engineering and Technology

Algorithm

An algorithm is defined as a finite set of steps or instructions required to solve a given problem. Algorithms can be described by using two methods

1. Pseudo-codes
2. Flow charts

1. Pseudo-code

Pseudo-code is a compact and informal high-level description of a program. It is platform independent. Following are some keywords which are used while writing pseudo-codes.

BEGIN	indicates start of an algorithm
VARIABLES	indicates the variables used for storing the inputs or outputs
INPUT	indicates a user will be inputting something
OUTPUT	indicates that an output will appear on the screen
WHILE	a loop (iteration that has a condition at the beginning)
FOR	a counting loop (iteration)
REPEAT – UNTIL	a loop (iteration) that has a condition at the end
IF – THEN – ELSE	a decision (selection) in which a choice is made
END	indicates the end of a program
Any instructions that occur inside a selection or iteration are usually indented	

Example 1: Sequential Structure - Write pseudo-code for adding two numbers.

```

Begin
Variables X, Y, sum
Input X
Input Y
sum=X+Y
Output sum
End

```

Example 2: Iterative Structure - Write pseudo-code for finding sum of natural numbers from 1 to 100.

```

Begin
Variables counter, sum=0
For counter=1 to 100 step 1 do
    sum=sum+counter
Endfor
Output sum
End

```

Example 3: Conditional Structure - Write pseudo-code for checking if a number is positive or negative.

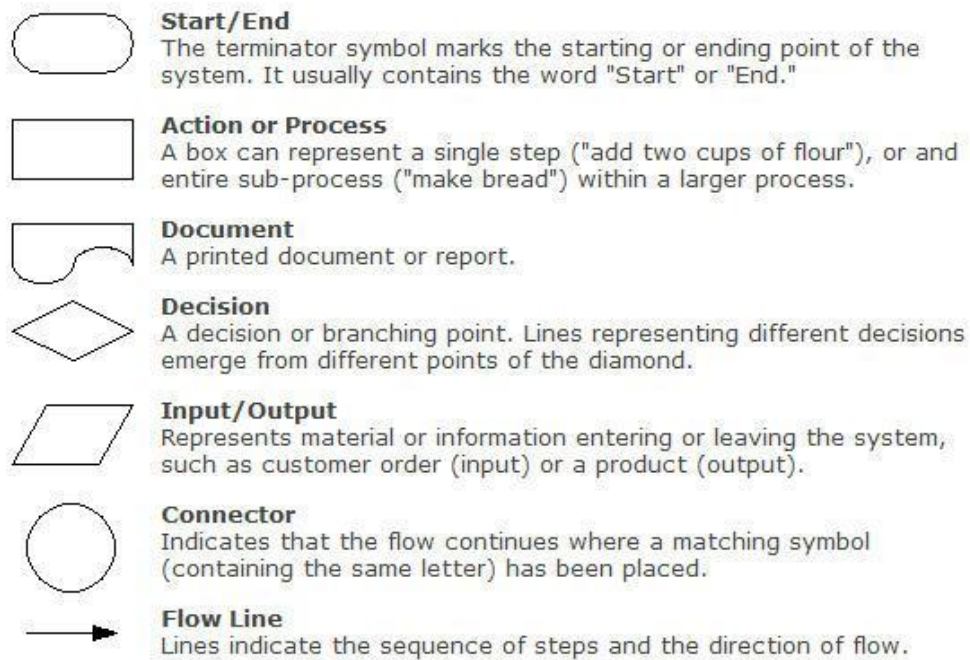
```

Begin
Variable num
Input num
If num>0 then
    output "entered number is positive"
Else if num <0 then
    output "entered number is negative"
Else
    output "entered number is zero"
Endif
End

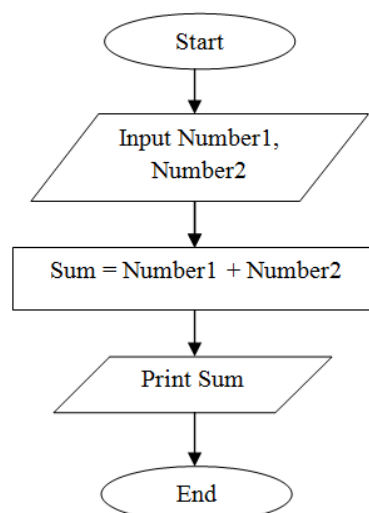
```

2. Flow charts

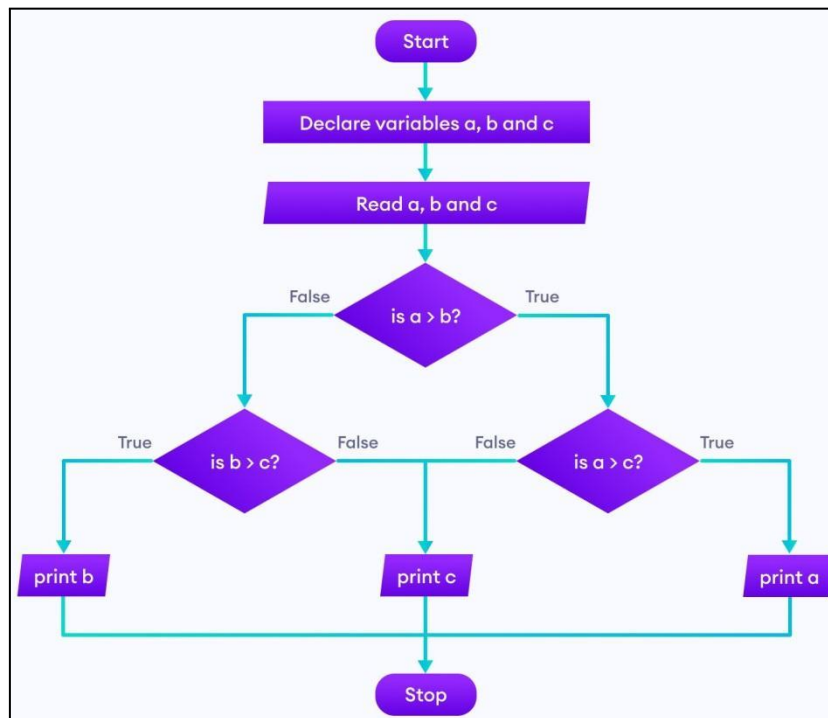
A flowchart is a pictorial representation of an algorithm in which the steps are drawn in the form of different shapes of boxes and the logical flow is indicated by interconnecting arrows. A flowchart is a graphical representation of the problem solving process.



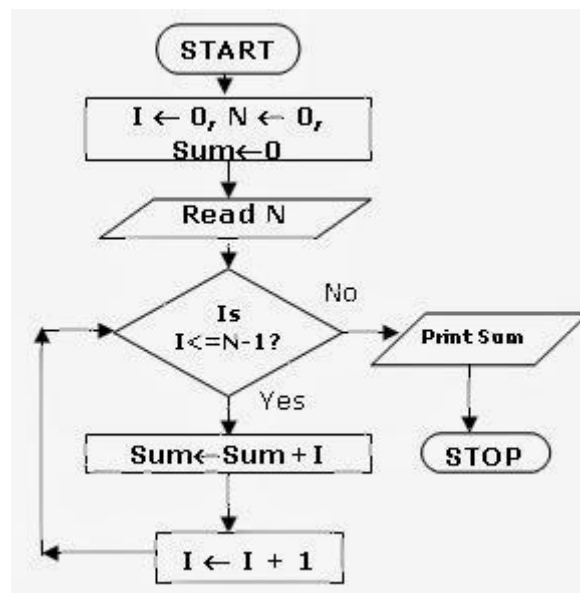
Example 4: Sequential Structure - Draw the flowchart of a program for adding two numbers.



Example 5: Conditional Structure - Draw the flowchart of a program for finding the maximum out of three numbers.



Example 6: Iterative Structure - Draw the flowchart of a program for finding sum of the initial "N" natural numbers.



Exercise

1. Write pseudo-code for Example 5 of Lab 01.
2. Write pseudo-code and draw flow chart. Ask a user to enter exam scores for five different courses and determine whether the student is passing or failing the course. Calculate the average score, the number of failed courses, and the number of passed courses. To confirm your solution, trace through the designed flowchart and pseudo-code by using the following test case: 88, 65, 45, 23, 77.
3. Ask a user to enter a number and then display the factorial of the entered number.
4. One of the jobs that Joe Roberts has been given at work is to order special paper for a report for a board meeting. The paper comes in reams of 500 sheets. He always makes five more copies than the number of people that will be there. Joe wants to know how many reams of paper he needs for a meeting. He can order only whole, not partial, reams. Assume the required number of pages will not equal an exact number of reams. Test your solution with the following data: The report is 140 pages long. There will be 25 people at the meeting.
5. Joe would like to build several bookcases that are different heights and widths. All will be 12 inches in depth. The bookcases will have three shelves, in addition to the bottom and the top. Write a solution to print the number of feet of 12-inch-wide boards that will Joe need to complete a bookcase, given the height and width.

Programming Fundamentals CT-175

Dynamic Memory Allocation

Objectives

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context. Students will be representing solutions to different problems as flowcharts and pseudo code

The purpose of this lab is to enable students to

understand the basics of
problem solving in
programmatic context.

Students will be representing
solutions to different
problems as

flowcharts and pseudo code

The purpose of this lab is to
enable students to

understand the basics of
problem solving in
programmatic context.

Students will be representing

solutions to different problems as flowcharts and pseudo code

The objective of this lab is to familiarize students with the features of Integrated Development Environment (IDE) for C language and basic structure of C program. By the end of this lab, students will be able to create, compile, and execute basic input-output programs written in C language.

Tools Required

DevC++ IDE

Course Coordinator –

Course Instructor –

Lab Instructor –

Department of Computer Science and Information Technology

NED University of Engineering and Technology

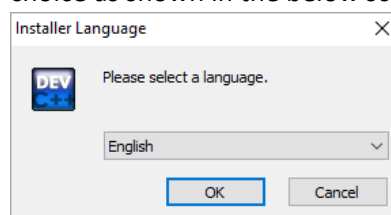
IDE – Integrated Development Environment

IDE stands for Integrated Development Environment. It is a software application that provides facilities for developing software. It consists of tools such as source code editor, automation tools, and debugger.

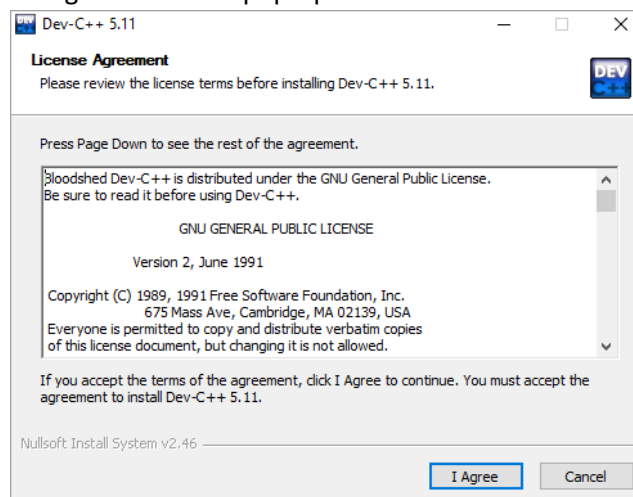
For C language programming we will be using DevC++. DevC++ is a full-featured IDE for the C/C++ programming language. DevC++ is free software and is distributed under the GNU General Public License. Thus we can distribute or modify the IDE freely. It offers to the programmer a simple and unified tool to edit, compile, link, and debug programs.

Installation and Configuration of DevC++ IDE

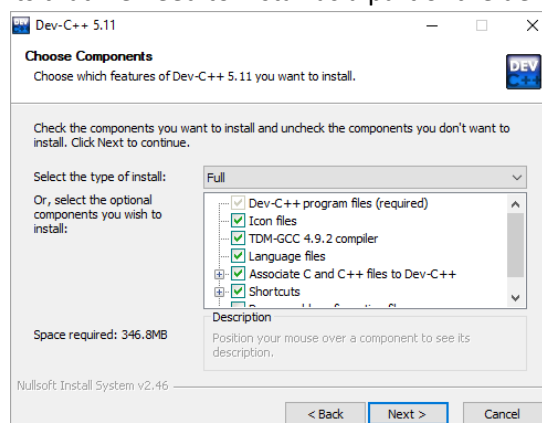
1. Download the required installer for dev-C++ IDE from <https://www.bloodshed.net/>
2. In order to start the installation process, double click on the downloaded file.
3. Select the language of our choice as shown in the below screenshot.



4. Agree to the license agreement that pop-ups next.

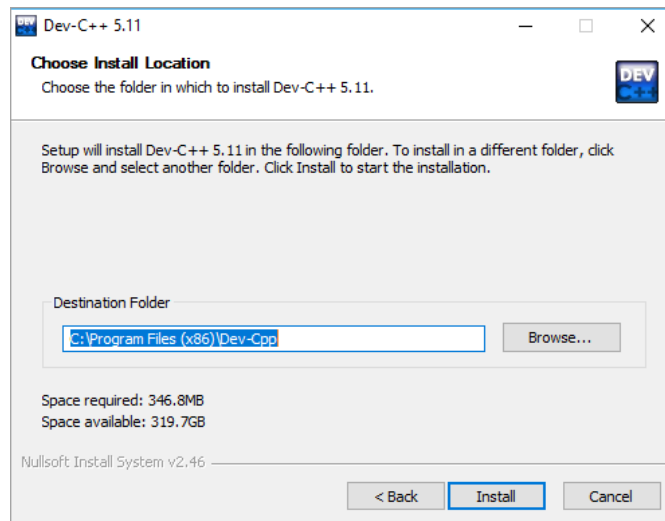


5. Select the components that we need to install as a part of the dev-C++ installation.



As shown in the above screenshot, we are provided with a list of components available for installation and a checkbox against each component. We can check/uncheck each box to indicate which components to install. Click next once the components are selected.

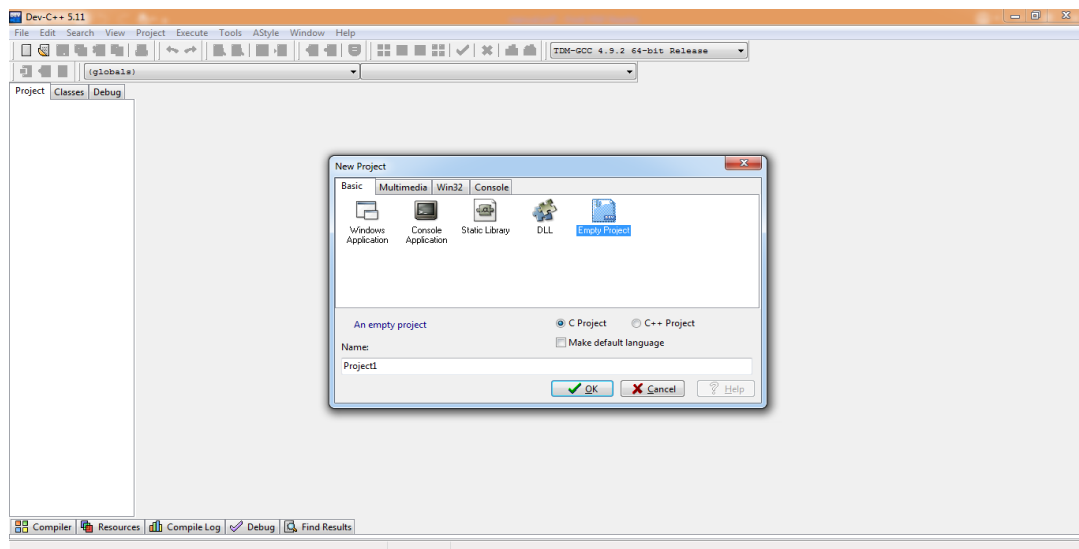
6. Now the installer prompts the user for the destination folder where the dev-C++ files/libraries etc. are to be copied. Click on install if you do not want to change the default path to destination folder. Otherwise, first define the path to destination folder and then click on install.



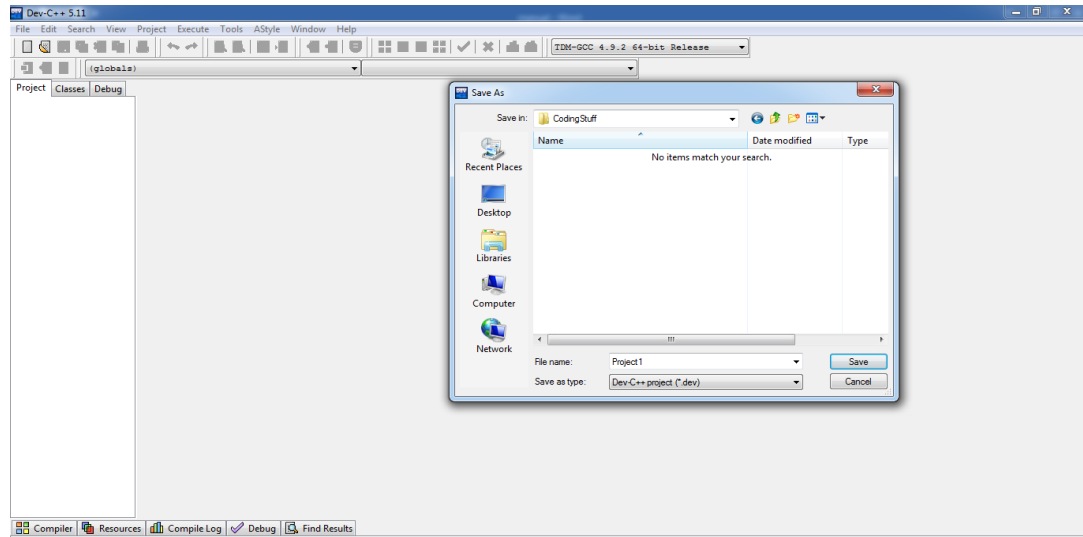
You are done with the installation of DevC++ IDE. Now let us use it to create the very first project. Double click on the DevC++ icon.

CREATE A NEW PROJECT

To create a new project or source file in dev-C++ go to menu File -> New->Project. Here, we can specify the project name. Make sure to select the “Empty Project” and also to check the “C Project” button.

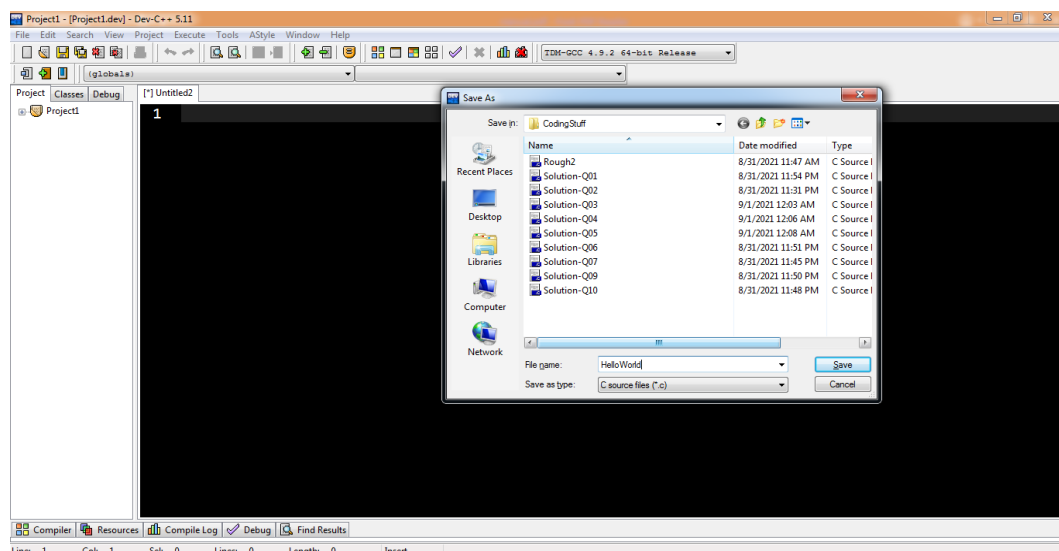


Once the entire information is provided, we can click ok and the IDE will ask for the path where the project is to be saved. When this is done, a workspace will open with the project explorer on the left-hand side that shows the project we just created.



ADD A SOURCE FILE TO A PROJECT

Add a new file by clicking **Project ->New File** or Right-click on **Project Name** in the project explorer and click **New File**.



Structure of a Program in C

Following is the traditional structure of a “Hello World” program in C language.

```

1 // Fig. 2.1: fig02_01.c
2 // A first program in C.
3 #include <stdio.h>
4
5 // function main begins program execution
6 int main( void )
7 {
8     printf( "Welcome to C!\n" );
9 } // end function main

```

Welcome to C!

Main() Function

All C programs are divided into units called functions. No matter how many functions there are in a C program, **main ()** is the one to which the control is passed from the operating system when the program is run; it's the first function executed.

Delimiters

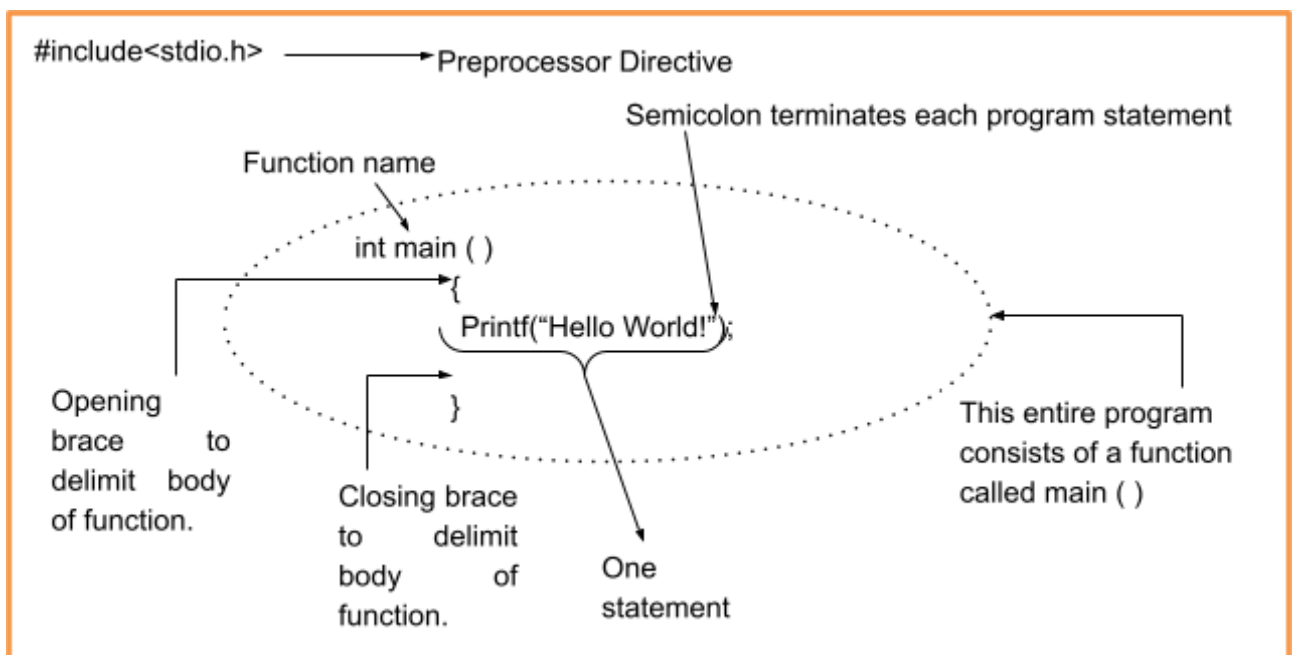
They signal the beginning and end of the body of the function. The opening brace ({) indicates a block of code that forms a distinct unit is about to begin. The closing brace (}) terminates the block of code. They are also used in loops and decision making statements.

Statement Terminator

A statement in a C program is terminated with a semicolon. Printf() is an example of a output statement. A semicolon does not separate statements; it terminates the line, not the carriage return you type afterwards. C does not pay attention to carriage return and white space characters.

Preprocessor library

A pound sign # indicates that the remainder of that line is an instruction (directive) to the preprocessor. #include <stdio.h> tells the preprocessor to add the code found in that file to the beginning of this program. This code handles all character stream inputs and outputs. Without this you cannot read from the keyboard or display anything to the monitor.



COMPILE/BUILD

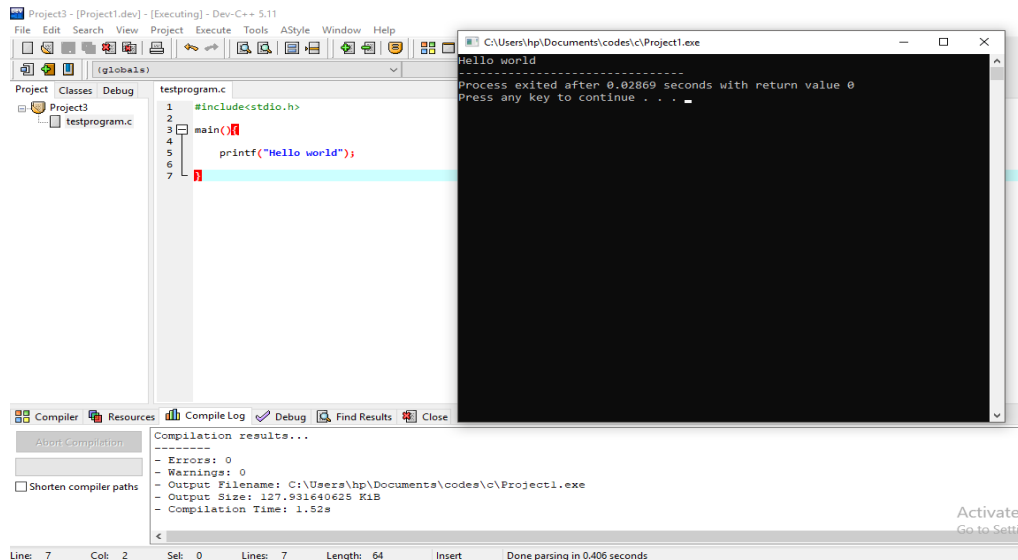
When we have all the code ready for the project, we will now compile and build the project.

- ✓ To compile the project, click **Execute -> Compile** (or click F9).
- ✓ We can see the compilation status in the **"Compile Log"** tab in the workspace.
- ✓ If there are any errors whether syntax or linker errors, then they will appear in the compiler tab.

- ✓ Once the project is compiled successfully, we need to run it.

EXECUTE PROJECT

- ✓ Click on **Execute ->Run.**(or click F10)
- ✓ The console window that gives us the output will be shown in the below screenshot.



Escape Sequences

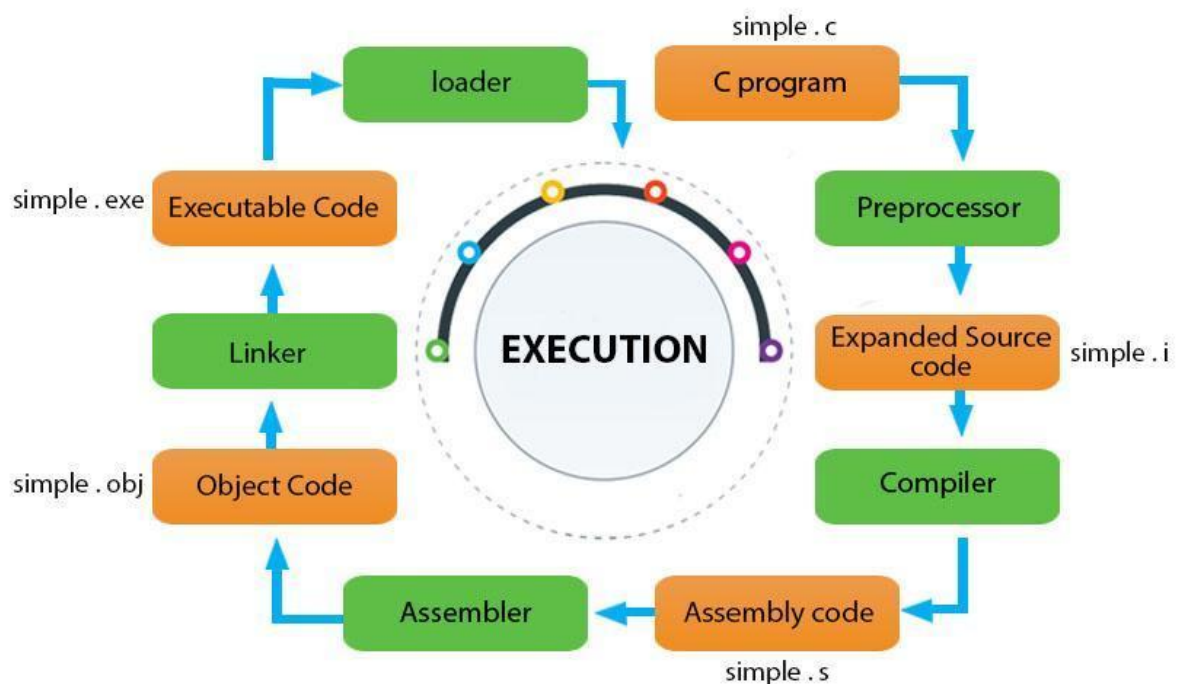
The backslash (\) is called an escape character. It indicates that printf is supposed to do something out of the ordinary. When encountering a backslash in a string, the compiler looks ahead at the next character and combines it with the backslash to form an escape sequence. Some common escape sequences are listed in the following Table.

Escape Sequence	Meaning
\n	New Line
\t	Horizontal Tab
\b	BackSpace
\r	Carriage Return
\a	Audible bell
\'	Printing single quotation
\"	printing double quotation
\?	Question Mark Sequence
\\	Back Slash
\f	Form Feed
\v	Vertical Tab
\0	Null Value
\nnn	Print octal value
\xhh	Print Hexadecimal value

<pre> 1 #include <stdio.h> 2 int main(){ 3 printf("Programming\\Fundamentals \\n"); 4 printf("new line \\n next line \\n"); 5 printf("welcome \\to\\' consolidated\\? \\v example \\n"); 6 printf("\\v"); 7 printf("\\learning is fun\\ " "); 8 printf("\\r"); 9 printf(" \\n\\'text surrounded with single quotation\\' " "); 10 printf(" \\n\\\"double quotes surrounded text\\\" " "); 11 printf(" \\n whats your name\\? "); 12 printf(" \\n E:\\test\\test1\\test2 "); 13 printf(" \\n A%B "); 14 return 0; 15 }</pre>	<pre> /tmp/q3K5ibfIYU.o ProgrammingFundamentals new line next line welcome 'to' consolidated? example "learning is fun" 'text surrounded with single quotation' "double quotes surrounded text" whats your name? E:\test\test1\test2 A%B</pre>
--	--

FLOW OF EXECUTION OF C PROGRAM

C is a high-level programming language developed in 1972 by Dennis Ritchie at AT&T Bell Laboratories. Programs in C typically go through six phases to be executed i.e., edit, preprocess, compile, link, load and execute.



1. Write a c program and save it with the extension “.c”.
2. The C program is sent to **preprocessor** first. The preprocessor is responsible to convert preprocessor directives (# sign denotes directives) into their respective values. The preprocessor generates an expanded source code.
3. Expanded source code is sent to **compiler** which compiles the code and converts it into assembly code. **Note** - Computer programs are written using high-level programming languages. The source code is converted into machine-understandable machine code. A compiler is used for this conversion. Compiler is a translator that converts the source code from high-level programming language to a lower level machine language in order to create an executable program.
4. The assembly code is sent to **assembler** which assembles the code and converts it into object code. Now a simple.obj file is generated.
5. The object code is sent to **linker** that links it to the library such as header files. Then it is converted into executable code. **Note** - Library functions are not a part of any C program but of the C software. Thus, the compiler doesn't know the operation of any function, whether it is

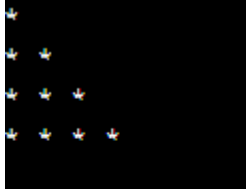
printf or scanf. The definitions of these functions are stored in their respective library which the compiler should be able to link. This is what the Linker does. So, when we write #include, it includes stdio.h library which gives access to Standard Input and Output. The linker links the object files to the library functions and the program becomes a .exe file. A simple.exe file is generated which is in an executable format.

6. The executable code is sent to **loader** which loads it into memory and then it is executed. After execution, output is sent to console. **Note** - Whenever we give the command to execute a particular program, the loader comes into work. The loader will load the .exe file in RAM and inform the CPU with the starting point of the address where this program is loaded.

Exercise

1. Write a C Program to play beep five times.
2. What does the following code print?

```
printf( "\n**\n***\n****\n*****\n" );
```
3. Write a C program to print the following shapes using escape sequences.



4. Write a program that prints the following shapes with asterisks.



Programming Fundamentals CT-175

Dynamic Memory Allocation

Objectives

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to understand the basics of

programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to

understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The objective of this lab is to enable students to understand the basic data types supported by C, rules for defining variables, and writing sequential programs by making

use of format specifiers. By the end of this lab students will be able to write simple sequential programs including variables, inputs, processes, and outputs.

Tools Required

DevC++ IDE

Course Coordinator –

Course Instructor –

Lab Instructor –

Department of Computer Science and Information Technology

NED University of Engineering and Technology

Variables in C

In programming, a variable is a container (storage area) to hold data. To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location. A variable can be declared and initialized using following syntax:

```
datatype variable_name = value ;
```

Example:

```
int num = 9;
```

Here, num is a variable of integer type. Here, the variable is assigned an integer value 9.

Rules for naming a variable

- ✓ A variable name can only have letters (both uppercase and lowercase letters), digits and underscore.
- ✓ The first letter of a variable should be either a letter or an underscore.
- ✓ There is no rule on how long a variable name (identifier) can be. However, you may run into problems in some compilers if the variable name is longer than 31 characters.
- ✓ C is a statically typed language. This means that the variable type cannot be changed once it is declared.

Basic Data Types Supported by C

In C programming, data types are declarations for variables. This determines the type and size of data associated with variables. Here is a table containing commonly used types in C programming for quick access.

C Basic Data Types	32-bit CPU		64-bit CPU	
	Size (bytes)	Range	Size (bytes)	Range
char	1	-128 to 127	1	-128 to 127
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
long long	8	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4	3.4E +/- 38	4	3.4E +/- 38
double	8	1.7E +/- 308	8	1.7E +/- 308

Format Specifiers

Format Specifiers are strings used in the formatted input and output. The format specifiers are used in C to determine the format of input and output. Using this concept the compiler can understand that what type of data is in a variable while taking input using the scanf()

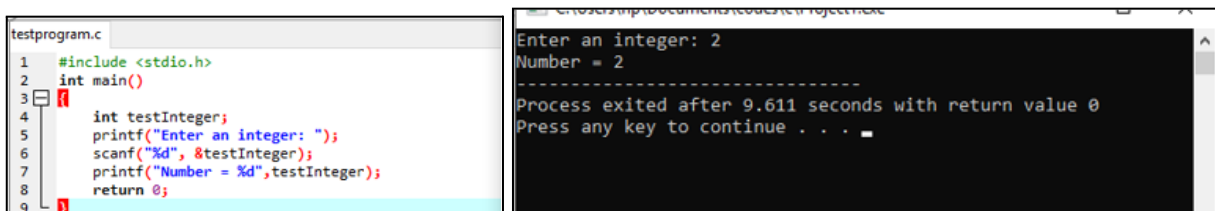
function and printing using `printf()` function. Some of the commonly used Format Specifiers are given in the following Table.

Format specifier	Description
<code>%d</code> or <code>%i</code>	It is used to print the signed integer value where signed integer means that the variable can hold both positive and negative values.
<code>%u</code>	It is used to print the unsigned integer value where the unsigned integer means that the variable can hold only positive value.
<code>%o</code>	It is used to print the octal unsigned integer where octal integer value always starts with a 0 value.
<code>%x</code>	It is used to print the hexadecimal unsigned integer where the hexadecimal integer value always starts with a 0x value. In this, alphabetical characters are printed in small letters such as a, b, c, etc.
<code>%X</code>	It is used to print the hexadecimal unsigned integer, but <code>%X</code> prints the alphabetical characters in uppercase such as A, B, C, etc.
<code>%f</code>	It is used for printing the decimal floating-point values. By default, it prints the 6 values after '.'.
<code>%e</code> or <code>%E</code>	It is used for scientific notation. It is also known as Mantissa or Exponent.
<code>%g</code>	It is used to print the decimal floating-point values, and it uses the fixed precision, i.e., the value after the decimal in input would be exactly the same as the value in the output.
<code>%p</code>	It is used to print the address in a hexadecimal form.
<code>%c</code>	It is used to print the unsigned character.
<code>%s</code>	It is used to print the strings.
<code>%ld</code>	It is used to print the long-signed integer value.

Input in C

In C programming, `scanf()` is one of the commonly used function to take input from the user. The `scanf()` function reads formatted input from the standard input such as keyboards. The syntax of `scanf()` function is: **`scanf("format string",&variable_name);`**

Example 01: Single input from user...



The image shows two side-by-side windows. The left window displays a C program named `testprogram.c` with the following code:

```

1 #include <stdio.h>
2 int main()
3 {
4     int testInteger;
5     printf("Enter an integer: ");
6     scanf("%d", &testInteger);
7     printf("Number = %d", testInteger);
8     return 0;
9 }

```

The right window shows the program's execution. It prompts "Enter an integer: 2", displays "Number = 2", and then shows the process exiting after 9.611 seconds with a return value of 0. It also prompts "Press any key to continue . . .".

Example 02: Multiple inputs from user ...

```
#include <stdio.h>
int main()
{
    int a;
    float b;

    printf("Enter integer and then a float: ");

    // Taking multiple inputs
    scanf("%d%f", &a, &b);

    printf("You entered %d and %f", a, b);
    return 0;
}
```

```
Enter integer and then a float: 4
5.5
You entered 4 and 5.500000
-----
Process exited after 8.889 seconds with return value 0
Press any key to continue . . .
```

Output in C

In C programming, `printf()` is one of the main output function. The function sends formatted output to the screen. The syntax of `printf()` function is: **`printf("format string", variable_name);`**

Example 03:

```
#include <stdio.h>

int main()
{
    printf("Hello World");
}
```

```
Hello World
-----
Process exited after 0.01564 seconds with return value 0
Press any key to continue . . .
```

Example 04:

```
#include <stdio.h>

int main(){
    int testInteger=5;
    printf("Number is %d",testInteger);
}
```

```
Number is 5
-----
Process exited after 0.03464 seconds with return value 0
Press any key to continue . . .
```

Exercise

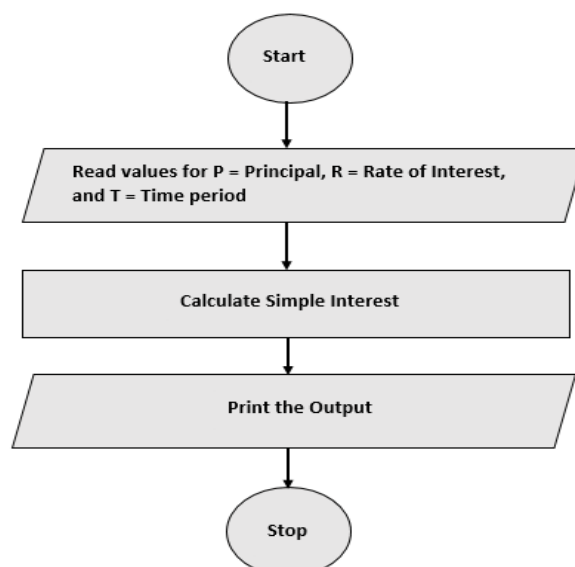
1. Write a C program that takes two integer values as input from the user. Then swap the values taken from the user and display the output of the variables.
2. A customer asks the IT firm to develop a program in C language, which can take tax rate and salary from the user on runtime and then calculate the tax, the user has to pay and the salary he/she will have after paying the tax. This information is then provided to the user.
3. A car traveled for some hours. The time car traveled is taken at run time of the program, and it must not be negative and must be between one to five hours. The car had not traveled same distance in each hour. The distance that the car covered must not be negative. Write a C Program that computes the Average Speed of the Car in miles per hour. Hint: the restrictions can be displayed in the form of message on the window.
4. Explain the output of this C program. Why the wrong value is being displayed in the output?

```
#include <stdio.h>

int main(){
    int testInteger=3000000000;
    printf("Number is %d",testInteger);
}
```

```
D:\PF\Lab 3 example codes\Printf.exe
Number is -1294967296
-----
Process exited after 0.03059 seconds with return value 0
Press any key to continue . . .
```


5. Construct a C program with the flowchart below. The input value of the Principle must be between 100 Rs. To 1,000,000 Rs. The Rate of interest must be between 5% to 10% and Time Period must be between 1 to 10 years. Hint: these restrictions can be displayed in the form of message on the window.



Programming Fundamentals CT-175

Dynamic Memory Allocation

Objectives

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to

understand the basics of
problem solving in
programmatic context.

Students will be representing
solutions to different
problems as

flowcharts and pseudo code

The purpose of this lab is to
enable students to

understand the basics of
problem solving in
programmatic context.

Students will be representing
solutions to different
problems as

flowcharts and pseudo code

The objective of this lab is to familiarize students with the conditional structure. By the end of this lab students will be able to write conditional programs by using different simple and nested conditional structure.

Tools Required

DevC++ IDE

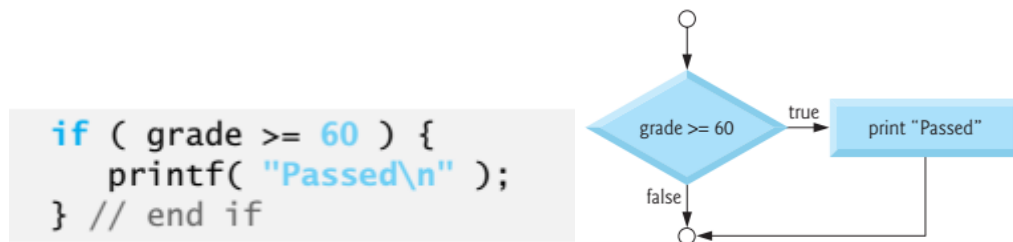
Course Coordinator –
Course Instructor –
Lab Instructor –
Department of Computer Science and Information Technology
NED University of Engineering and Technology

Conditional/Selection Structure in C

A selection structure is a programming feature that performs different processes based on whether a boolean condition is true or false. Selection structures use relational operators to test conditions. There are different types of selection structures that can be used to achieve different outcomes.

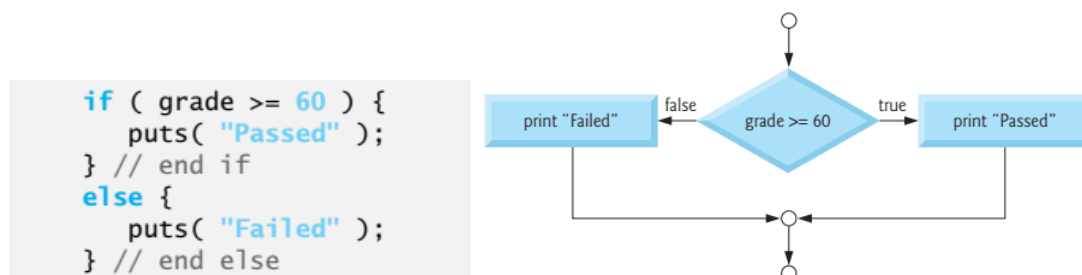
If – Single selection Statement

If you want your program to do something if a condition is true, but do nothing if that condition is false, then you should use an if-end structure.



If-else – Double selection Statement

If you want your program to do something if a condition is true and do something different if it is false, then you should use an if-else structure.



Example 01: Find out if a person is eligible for voting or not.

```

#include <stdio.h>
int main( ){
    int age;
    printf("Enter your age:");
    scanf("%d",&age);
    if(age >=18)
        printf("You are eligible for voting");
    else
        printf("You are not eligible for voting");
    return 0;
}
  
```

Note that we can omit the curly braces, enclosing the body, if there is only one statement inside body of if or else.

Ladder If-else-if – Multiple Selection Statement

If you want to test multiple conditions, then you can include an elseif structure within an if-end or if-else structure.

```
if ( grade >= 90 ) {
    puts( "A" );
} // end if
else if ( grade >= 80 ) {
    puts( "B" );
} // end else if
else if ( grade >= 70 ) {
    puts( "C" );
} // end else if
else if ( grade >= 60 ) {
    puts( "D" );
} // end else if
else {
    puts( "F" );
} // end else
```

Example 02: Find maximum of two numbers.

```
#include <stdio.h>
int main( ){
    int var1, var2;
    printf("Input the value of var1:");
    scanf("%d", &var1);
    printf("Input the value of var2:");
    scanf("%d",&var2);
    if (var1 !=var2)
        printf("var1 is not equal to var2\n");
    else if (var1 > var2)
        printf("var1 is greater than var2\n");
    else if (var2 > var1)
        printf("var2 is greater than var1\n");
    else
        printf("var1 is equal to var2\n");
    return 0;
}
```

Nested If... else Statements

Test for multiple cases by placing if...else statements inside if...else statements.

```

if ( grade >= 90 ) {
    puts( "A" );
} // end if
else {
    if ( grade >= 80 ) {
        puts("B");
    } // end if
    else {
        if ( grade >= 70 ) {
            puts("C");
        } // end if
        else {
            if ( grade >= 60 ) {
                puts( "D" );
            } // end if
            else {
                puts( "F" );
            } // end else
        } // end else
    } // end else
} // end else

```

Example 03: Find maximum of two numbers.

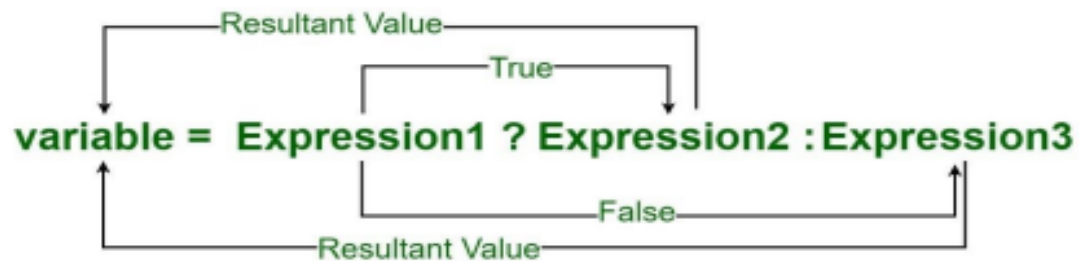
```

#include <stdio.h>
int main( ){
    int var1, var2;
    printf("Input the value of var1:");
    scanf("%d", &var1);
    printf("Input the value of var2:");
    scanf("%d",&var2);
    if (var1 != var2){
        printf("var1 is not equal to var2\n");
        //Nested if else
        if (var1 > var2)
            printf("var1 is greater than var2\n");
        else
            printf("var2 is greater than var1\n");
    }
    else
        printf("var1 is equal to var2\n");
    return 0;
}

```

Ternary Operator in C (?:)

The ternary operator takes three operands/expressions as input. If the result of expression1 is true then expression2 is returned otherwise expression3 is returned. It is equivalent to simple if-else statement. We can nest ternary operator within a ternary operator. For example expression 2/3 can be formed by using a ternary operator.



Example 04: Find out if a person is eligible for voting or not.

```
#include <stdio.h>
int main() {
    int age;
    // take input from users
    printf("Enter your age: ");
    scanf("%d", &age);
    // ternary operator to find if a person can vote or not
    (age >= 18) ? printf("You can vote") : printf("You cannot vote");
    return 0;
}
```

Example 05:

```
#include <stdio.h>
int main( ) {
    char operator = '+'; // create variables
    int num1 = 8, num2 = 7, result=0 ;
    result= (operator == '+') ? (num1 + num2) : (num1 - num2);
    printf("%d", result);
    return 0;
}
```

Exercise

1. Write a C program to input a character from user and check whether given character is small alphabet, capital alphabet, digit or special character, using if else.
2. Write a C program to receive an 8-bit number into a variable and then check if its 4th and 7th bits are on. If these bits are found to be on, then put them off.
3. An online shopping store is providing discounts on the items due to the Eid. If the cost of items is more than 1999 it will give a discount upto 50%. If the cost of shopping is 2000 to 4000, a 20% discount will be applied. If the cost of shopping is 4001 to 6000, a 30% discount will be applied. If it's more than 6000 then 50% discount will be applied to the cost of shopping. Print the actual amount, saved amount and the amount after discount.
4. Write a C program to find all roots of a quadratic equation by using the given formula; it is required to take user input for a, b and c values.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

5. Teacher asks the student to check whether the input number is divisible by 7 or not. For checking the divisibility, take the last digit and double it, take the rest of the digits and subtract the doubled last digit repeat until the result is 7, -7 or 0. For example:
 10976 -> 1097-12 = 1085 -> 108-10 = 98 -> 9-16 = -7
 49 -> 4 - 18 = 14 -> 1 - 8 = -7
6. Write a program that asks for the number of calories and fat grams in a food. The program should display the percentage of calories that come from fat. If the calories from fat are less than 30% of the total calories of the food, it should also display a message indicating that the food is low in fat. One gram of fat has 9 calories, so Calories from fat = fat grams * 9. The percentage of calories from fat can be calculated as: calories from fat/total calories
Input validation: Make sure the number of calories and fat grams are not less than 0. Also, the number of calories from fat cannot be greater than the total number of calories. If that happens, display an error message indicating that either the calories or fat grams were incorrectly entered.

Programming Fundamentals CT-175

Dynamic Memory Allocation

Objectives

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to understand the basics of problem solving in

programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to

understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The objective of this lab is to familiarize students with multiple selection statement. By the end of this lab students will be able to write conditional programs by using different simple and nested multiple selection statements.

Tools Required

DevC++ IDE

Course Coordinator –

Course Instructor –

Lab Instructor –

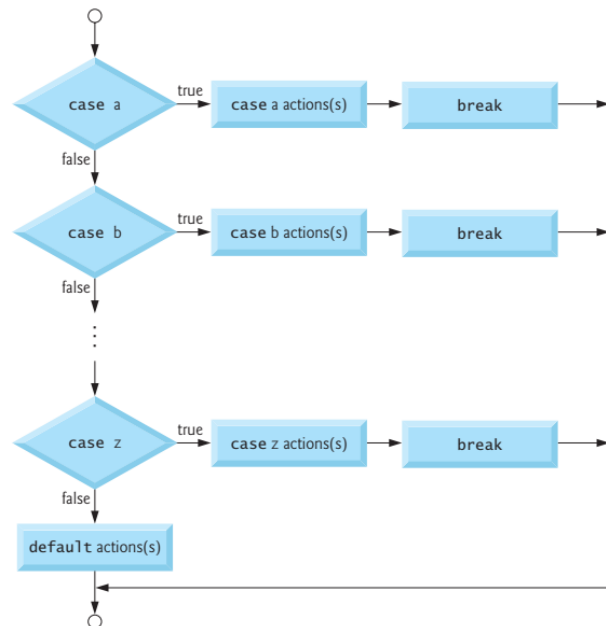
Department of Computer Science and Information Technology

NED University of Engineering and Technology

Multiple Selection Statement in C

The switch statement is defined as multiple selection construct and is intended to simplify series of if/else statements when comparing a single integer. The switch statement consists of a series of case labels, an optional default case and statements to execute for each case, as shown in the following diagrams. The switch statement is different from all other control statements in that braces are not required around multiple actions in a case of a switch. Although the case clauses and the default case clause in a switch statement can occur in any order, it's common to place the default clause last.

```
switch (selector)
{
  case L1: statements1; break;
  case L2: statements2; break;
  ...
  default: statements_n;
}
```



The break statement causes program control to continue with the first statement after the switch statement. The break statement is used because the cases in a switch statement would otherwise run together. If break is not used anywhere in a switch statement, then each time a match occurs in the statement, the statements for all the remaining cases will be executed. The flowchart makes it clear that each break statement at the end of a case causes control to immediately exit the switch statement.

When using the switch statement, remember that each individual case can test only a constant integral expression. A character constant can be represented as the specific character in single quotes, such as 'A'. Characters must be enclosed within single quotes to be recognized as character constants—characters in double quotes are recognized as strings.

```

#include <stdio.h>
int main() {
    int num = 8;
    switch (num) {
        case 7:
            printf("Value is 7");
            break;
        case 8:
            printf("Value is 8");
            break;
        case 9:
            printf("Value is 9");
            break;
        default:
            printf("Out of range");
            break;
    }
    return 0;
}

```

```

1 // find whether a character is vowel or consonant
2 #include <stdio.h>
3 int main(){
4     char ch;
5     printf("Enter any character: ");
6     scanf("%c", &ch);
7     switch(ch){ /* Switch ch value */
8         case 'a':
9         case 'e':
10        case 'i':
11        case 'o':
12        case 'u':
13        case 'A':
14        case 'E':
15        case 'I':
16        case 'O':
17        case 'U':
18            printf("Vowel");
19            break;
20        default: printf("Consonant"); }
21     return 0;
22 }

```

Example 01: Design a calculator using numbers to choose operators, if 1 is pressed then addition is performed, if 2 is pressed then subtraction if 3 is pressed then multiplication, if 4 is pressed then division otherwise print invalid choice.

```

#include <stdio.h>
int main() {
    int num1, num2, choice;
    printf("Enter two numbers\n");
    scanf("%d%d", &num1, &num2);

    printf("Press \n 1 for sum \n 2 for sub \n 3 for mul \n 4 for div\n");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Sum = %d\n", num1 + num2);
            break;
        case 2:
            printf("Subtraction = %d\n", num1 - num2);
            break;
        case 3:
            printf("Multiplication = %d\n", num1 * num2);
            break;
        case 4:
            printf("Division = %d\n", num1 / num2);
            break;
        default:
            printf("Enter valid choice\n");
    }
}

```

Nested Switch-Case Statements

Placing the simple switch case statements inside an existing case statement is called nested switch-case statement. Each block of nested switch case statement, logically performs the same as simple switch case statement. Following is the syntax of nested switch case statement.

```
#include <stdio.h>
int main() {
    1 int ID, password;
    printf("Plese Enter Your ID:\n ");
    scanf("%d", & ID);
    switch (ID) { 2
        case 500:
            printf("Enter your password:\n ");
            scanf("%d", & password);
            3 switch (password) {
                case 000:
                    printf("Welcome Dear Programmer\n");
                    break;
                default:
                    printf("incorrect password");
                    break;
            }
            break;
        4 default:
            printf("incorrect ID");
            break;
    }
}
```

Example 02: Searching for departments in a university with hierarchal structure, that is, we have schools and then under each school we have different departments.

```

1  #include<stdio.h>
2  int main()
3  {
4      int a,b;
5      printf("1.School of Computer Science\n");
6      printf("2.School of Business\n");
7      printf("3.School of Engineering\n");
8      printf("make your selection\n");
9      scanf("%d",&a);
10     switch (a)
11     {
12         case 1:
13             //code to be executed
14             //if school of computer science is chosen;
15             break;
16         case 2:
17             //code to be executed
18             //if school of business is chosen;
19             printf("Available Departments\n");
20             printf("1.Department of commerce\n");
21             printf("2.Department of purchasing\n");
22             printf("Make your selection.\n");
23             scanf("%d",&b);
24
25             //inner switch to display the departments
26             //under the school of commerce
27             switch(b)
28             {
29                 case 1:
30                     // code to be executed if b = 1;
31                     printf("You chose Department of commerce\n" );
32                     break;
33                 case 2:
34                     // code to be executed if b = 2;
35                     printf("You chose Department of purchasing" );
36                     break;
37             }
38             break;
39     }
40 }

```


Exercise

1. You must have seen the question before deleting anything like “Are you sure to delete [Y/y] / [N/n] ? Create a program that asks for this question if user enters Y or y it prints “Deleted successfully”. If the user enters N or n it prints “Delete cancelled” otherwise it prints choose the right option using switch statement.
2. Write a program to control a coffee machine. Allow the user to input the type of coffee as B for Black and W for White. Ask the user if the cup size is double and if the coffee is manual. The following table details the time chart for the machine for each coffee type. Display a statement for each step. If the coffee size is double, increase the baking time by 50 percent. Use functions to display instructions to the user and to compute the coffee time.

Operation	White Coffee	Black Coffee
Put Water	15 mins	20 mins
Sugar	15 mins	20 mins
Mix Well	20 mins	25 mins
Add Coffee	2 mins	15 mins
Add Milk	4 mins	-
Mix Well	20mins	25 mins

3. Write a program in which user enters his NTS and F.Sc marks and your program will help student in selection of university. Based on these marks Student will be allocated a seat at different department of different university.

Oxford

IT: Above 70% in Fsc. and 70 % in NTS

Electronics: Above 70% in Fsc. and 60 % in NTS

Telecommunication Above 70% in Fsc. and 50 % in NTS

MIT

IT: 70% - 60 % in Fsc. and 50 % in NTS

Chemical: 59% – 50 % in Fsc. and 50 % in NTS

Computer: Above 40% and below 50 % in Fsc. and 50 % in NTS

4. Using IF and Switch statement, write a program that displays the following menu for the food items available to take order from the customer:

B= Burger (Rs. 200)

F= French Fries (Rs. 50)

P= Pizza(Rs. 500)

S= Sandwiches (Rs. 150)

The costumer can order any combination of available food. The program first ask to enter the no of types of snacks i.e. 2, 3 or 4 then it ask to enter the choice i.e. B for Burger and then for quantity. The program should finally display the total charges for the order.

```
ABC Restaurant Online Order Placement
WELCOME!

Please select from the following Menu
B= Burger
F= French Fries
P= Pizza
S= Sandwiches
How many types of snacks you need to order: 2
Enter first Snack you want to order: B
Please provide quantity: 2
Enter second Snack you want to order: P
Please provide quantity: 3
-----
You have ordered!
2 Burger (s) value 400 PKR
3 pizza (s) value 1500 PKR
Total: 1900 PKR
Thank you for your order.. have a nice day.
```

Programming Fundamentals CT-175

Dynamic Memory Allocation

Objectives

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to understand the basics of

programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to

understand the basics of

problem solving in

programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The objective of this lab is to familiarize students with iterative structures supported by C. By the end of this lab students will be able to write iterative programs by using simple and nested iterative structures.

Tools Required

DevC++ IDE

Course Coordinator –

Course Instructor –

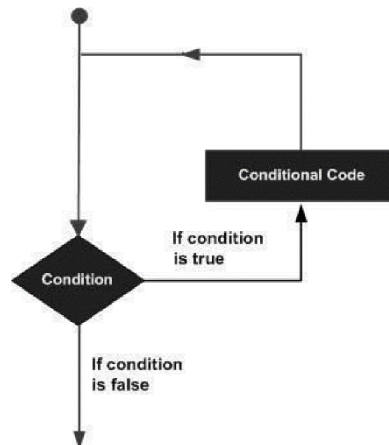
Lab Instructor –

Department of Computer Science and Information Technology

NED University of Engineering and Technology

Iterative Structure in C

By iteration or repetition, we mean doing the same task again and again. Usually, we want to repeat the task until some condition is met or for a predefined number of iterations. In computer science this is commonly referred as loop. Loops are used to repeat a statement, a block, a function or any combination of these. A typical looping workflow is shown in the figure.



Loops are very powerful and important tool in programming. Consider adding the salary of 1000 employees of a company. Or finding out how many people have been vaccinated from a list of 30 million people, that is just the population of Karachi what about whole Sindh or Pakistan? One way is to copy paste the code or repeating the lines of code. Well, the easier way is using a loop. C programming language offers three kinds of iterative statements, that is, For, While, and Do-While.

For Loop

The For loop is mostly used when we want to iterate for a specific number of times. Its syntax is as below

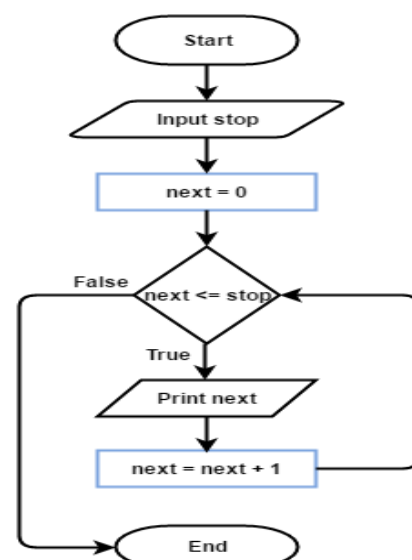
```

for ( initialization expression ; loop repetition condition ; update expression ){
    Body of the Loop;
}
  
```

- ✓ The initialization is an assignment statement that is used to initialize the loop control variable with a value. This is the first statement to be executed in the loop and only run once.
- ✓ The condition is a relational expression that determines when the loop exits. This runs after initialization, and verified before every iteration.
- ✓ The update expression either increment or decrement the loop control variable on each iteration.

```

#include <stdio.h>
int main ( ) {
    int stop, next;
    printf("Enter ending value:");
    scanf("%d",&stop);
    for(next = 0 ; next <= stop ; next=next+1){
        printf("%d\t", next);
    }
}
  
```



```
return 0;
}
```

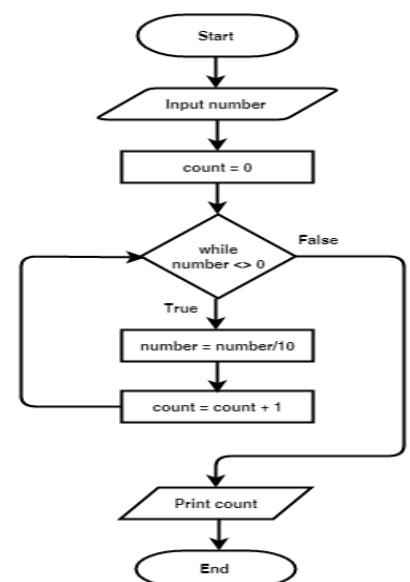
While Loop

While loop is used in situations when prior information about the number of iterations is not available. For example, as a game developer you don't have any knowledge how many times user will play the game so you want to loop over the choice if they want to play again or not. The syntax of while loop in C is given as follows.

```
Initialization expression;
while ( loop repetition condition ) {
    Body of the loop;
    Update expression;
}
```

- ✓ The loop repetition condition (a condition to control the loop process) is tested; if it is true, the statement (loop body) is executed, and the loop repetition condition is retested.
- ✓ The statement is repeated as long as (while) the loop repetition condition is true. When this condition is tested and found to be false, the while loop is exited and the next program statement after the while statement is executed. If the condition is true forever the loop will run forever, we call such loop an infinite loop.
- ✓ It may not be executed at all if condition is false right from start.

```
#include <stdio.h>
int main() {
    int number=0, count=0;
    scanf("%d",&number);
    while (number != 0) {
        number = number/10;
        count = count + 1;
    }
    printf("The number of digits are: %d", count);
    return 0;
}
```



Do-While Loop

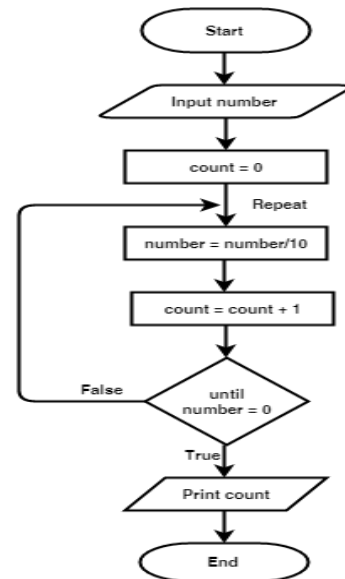
The do-while loop checks the loop repetition condition after running the body of the loop. This structure makes it most favorable in conditions where we are interested in running the body at least once. For example, as a web-developer you are writing a registration application. You would like to make sure if the username is already taken or not. Therefore, you will keep asking for a unique username until provided, in such situations you would want the user to provide the username first and then perform the check.

```
Initialization expression;
do{
    Body of the Loop;
```

```
Update expression;
} while (loop repetition condition);
```

- ✓ First, the body of the loop is executed.
- ✓ Then, the loop repetition condition is tested if it is true, the body is again and the condition is retested until it remains true the loop continues. When this condition is tested and found to be false, the loop is exited and the next statement after the do-while is executed.

```
#include <stdio.h>
int main(){
int number=0, count=0;
scanf("%d", &number);
do{
    number = number/10;
    count = count + 1;
} while (number != 0);
printf("The number of digits are: %d", count);
return 0;
}
```



Continue Statement

The continue statement is very powerful in situations where we want to execute some portion of the loop body and skip a statement or block of statements. The control skips the loop body as it reaches the continue statement and starts the next iteration. Usually there is a condition for which we want to skip certain statements. For example, as an AI developer you would work with a lot of datasets (e.g., thousands of images). Before training your AI model on these datasets you need to filter the corrupted or invalid images from the valid once. In such situations you can utilize continue on dependent situations instead of using multiple conditional statements.

```
for (initialization; condition; increment/decrement) {
    block of statements;
    if(condition){
        block of statements;
        continue;
    }
    block of statements;
}
```

```
#include<stdio.h>
int main( ){
int i;
for (int i=0; i<=10 ;i++){
    if((i==3)||(i==7)){
        printf("Continue statement executed\n");
        continue;
    }
}
```



```

    }
    printf("The sum is %d", sum);
    }
Return 0;
}

```

Break Statement

Break statement is used to exit the body of the loop without meeting the loop repetition condition. The statements after the break never get executed and usually break is used to stop the loop before the loop termination condition is met. The controls execute the next statement after the loop body once it reaches the break statement in the loop body. Usually there is a condition upon which we want to exit the loop. For example, you have written a fund-raising application for a cancer patient, after the required funds are collected; you would like to break out of the loop without knowing how many iterations have passed.

```

for (initialization; condition; increment/decrement) {
    block of statements;    // this block will execute always with each iteration of loop
    if(condition){
        block of statements;
        break;
    }
    block of statements;
}

```

```

#include<stdio.h>
int main( ){
int a, sum=0;
while(1){
    scanf("%d",&a);
    if(a== -999) {
        printf("break statement executed!\n");
        break;
    }
    sum=sum+a;
}
printf("The sum is %d", sum);
return 0;
}

```

Nested Loops

Nested loop means a loop statement inside another loop statement. That is why nested loops are also called as "loop inside loop".

```

do{
    while(condition) {
        for ( initialization; condition; increment ) {
            // statement of inside for loop
        }
        // statement of inside while loop
    }
    // statement of outer do-while loop
}

```

```
}while(condition);
```

```
// A program to print all prime factors of a number by using nested loops.
#include <math.h>
#include <stdio.h>
int main( ){
    int n = 315;
    while (n % 2 == 0) { // Print the number of 2s that divide n
        printf("%d ", 2);
        n = n / 2; }
    for (int i = 3; i <= sqrt(n); i = i + 2) { // n must be odd at this point, (Note i = i +2)
        while (n % i == 0) { // While i divides n, print i and divide n
            printf("%d ", i);
            n = n / i; }
        }
    if (n > 2) // if n is a prime number greater than 2
        printf("%d ", n);
    return 0; }
```

Exercise

- Write a program which will find the factorial of a given number. Exit the program if the input number is negative. **Test case:** Input number = 5, Factorial is=5*4*3*2*1
- Write a program which will generate the Fibonacci series up to 1000. Also find the sum of the generated Fibonacci numbers divisible by 3, 5 and 7 only.
Fibonacci series is: 1 1 2 3 5 8 13 25.....
Note: Do this task by using *for loop* and *while loop*. Also identify which one is more efficient?
- Write a program which will input a 5-digit number. If the sum of digits is even, find whether the input number is a prime or not. If the sum of digits is odd find, whether the number is palindrome or not?
Example of prime number: A number which is only divisible by itself and 1 i.e., 7, 11, and 13.
Example of a Palindrome: A number whose reverse order is the same as the original number i.e., 11211, 44344.
- Develop a user-registration system have the following options.
 - Ask the user for a user-name (5 alphabets).
 - Password should be 6 characters long with at least 1 numeric, 1 capital and 1 small letter.
 - Display a "Account Created Successfully".
 - Login the user with correct username and password.
 - Display "Welcome username, you are now logged in".**Note:** Develop your application using break and continue statement.
- (Calculating the Value of π) Calculate the value of π from the infinite series. Print a table that shows the value of π approximated by one term of this series, by two terms, by three terms, and so on. How many terms of this series do you have to use before you first get 3.14? 3.141? 3.1415? 3.14159?
- (Diamond-Printing Program) Write a program that prints the following diamond shape. You may use printf statements that print either a single asterisk (*) or a single blank. Maximize your use of repetition (with nested for statements) and minimize the number of printf statements.

```
    *  
  ***  
*****  
*****  
*****  
*****  
*****  
  ***  
    *  
    *
```

Programming Fundamentals CT-175

Dynamic Memory Allocation

Objectives

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to understand the basics of

programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to

understand the basics of

problem solving in

programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The objective of this lab is to familiarize students with Array data structure. By the end of this lab students will be able to write programs by using single and multidimensional arrays.

Tools Required

DevC++ IDE

Course Coordinator –

Course Instructor –

Lab Instructor –

Department of Computer Science and Information Technology

NED University of Engineering and Technology

Array

An array is a collection of data items of the same type. Programming problems can be solved efficiently by grouping the data items together in main memory than allocating an individual memory cell for each variable. For example, a program that processes exam scores for a class, would be easier to write if all the scores were stored in one area of memory and were able to be accessed as a group. C allows a programmer to group such related data items together into a single composite data structure called array.

One Dimensional (1D) Arrays

In one-dimensional array, the components are arranged in the form of a list. The syntax for declaring and initialization 1D arrays in C is as follows.

Data_type array_name [size]; /* uninitialized */

Data_type array_name [size] = { initialization list }; /* initialized */

Data_type array_name [] = { initialization list }; /* initialized */

- ✓ The general uninitialized array declaration allocates storage space for array array_name consisting of "size" number of items (memory cells).
- ✓ Each memory cell can store one data item whose data type is specified by data_type (i.e., double, int, or char).
- ✓ The individual array elements are referenced by the subscripted variables array_name [0], array_name [1], . . . , array_name [size -1].
- ✓ A constant expression of type int is used to specify an array's size. In the initialized array declaration shown, the size shown in brackets is optional since the array's size can also be indicated by the length of the initialization list.
- ✓ Element 0 of the array being initialized is set to the first entry in the initialization list, element 1 to the second, and so forth.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element. For example,

double x[5] = { 5.0, 2.0, 3.0, 1.0, -4.5}; in the memory is as follows,

x[0]	x[1]	x[2]	x[3]	x[4]
5.0	2.0	3.0	1.0	-4.5

```
/* 1D Array Example */
#include<stdio.h>
int main( ){
    int avg, sum = 0;
    int i;
    int marks[5]; // array declaration
    for(i = 0; i<=4; i++){
        printf("Enter student marks: ");
        scanf("%d", &marks[i]); /* stores the data in array*/
    }
    for(i = 0; i<=4; i++)
        sum = sum + marks[i]; /* reading data from array */
    avg = sum/5;
    printf("Average marks are:%d\n", avg);
    printf("total marks are :%d\n", sum);
    return 0;
```

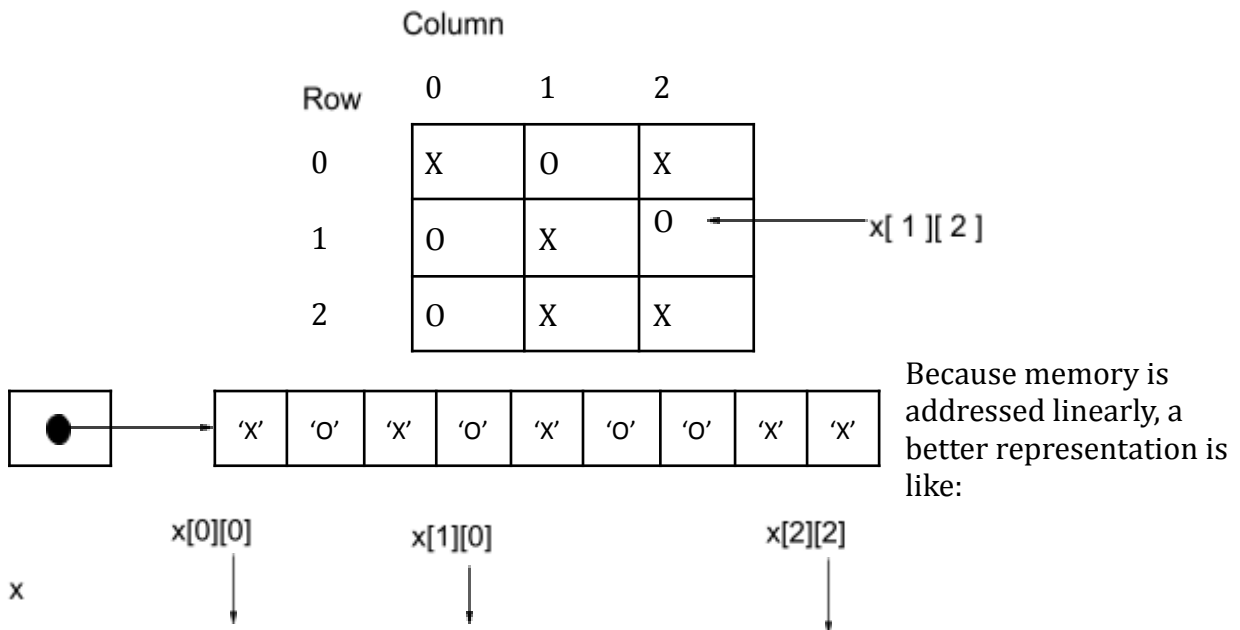
}

Two Dimensional (2D) Arrays

A two dimensional array is a collection of a fixed number of components arranged in rows and columns (that is, in two dimensions), wherein all components are of the same type. Two-dimensional arrays are used to represent tables of data, matrices, and other two-dimensional objects. The syntax of for declaring two dimensional arrays is as follows.
 Data_type array_name [size1] [size2]; /* uninitialized */

- ✓ Allocates storage for a two-dimensional array (array_name) with size1 number of rows and size2 number of columns.
- ✓ This array has size1*size2 number of elements, each of which must be referenced by specifying a row subscript (0 , 1 ,... size1-1) and a column subscript (0 , 1 ,...size2-1).
- ✓ Each array element contains an element of type data_type.

2D arrays are very useful for storing a table of data (not the only way). Any kind of matrix processing as a 2D array really is a matrix. The two array is represented in the memory as:
 char x[3][3] = {{'X', 'O', 'X'}, {'O', 'X', 'O'}, {'O', 'X', 'X'}};



```
//Example Program to display the transpose of given 2x2 matrix(2D array)
#include <stdio.h>
int main( ){
    int matrix[2][2], transpose[2][2], row, col;
    printf("\nEnter elements of matrix:\n");
    for(row=0; row<2; row++)          // Storing elements of the matrix
        for(col=0; col<2; col++){
            printf("Enter element a[%d][%d]: ",row,col);
            scanf("%d", &matrix[row][col]);
        }
    printf("\nEntered Matrix: \n"); // Displaying the matrix[][]
    for(row=0; row<2; row++)
```



```

    for(col=0; col<2; col++){
        printf("%d ", matrix[row][col]);
        if (col == 1)
            printf("\n\n");
    }
    for(row=0; row<2; row++) // Finding the transpose of matrix
        for(col=0; col<2; col++){
            transpose[col][row] = matrix[row][col];
        }
    printf("\nTranspose of Matrix:\n"); // Displaying the transpose of matrix
    for(row=0; row<2; row++)
        for(col=0; col<2; col++){
            printf("%d ",transpose[row][col]);
            if(col==1)
                printf("\n\n");
        }
    return 0;
}

```

String as Char Array

String is a sequence of characters that are treated as a single data item and terminated by a null character '\0'. Remember that the C language does not support strings as a data type. A string is actually a one-dimensional array of characters in C language. These are often used to create meaningful and readable programs. Strings can be declared or initialized in C by using char array as follows.

```

// valid
char name[13] = "StudyTonight";
char name[10] = {'c','o','d','e','\0'};

```

```

// Illegal
char ch[3] = "hello";
char str[4];
str = "hello";

```

You can use string data type for input and output in C as follows.

```

char str[20];
printf("Enter a string");
scanf("%[^\n]", &str);
printf("%s", str);

```

Gets() and Puts() string Functions

A string can be read using the %s placeholder in the scanf function. However, it has a limitation that the strings entered cannot contain spaces and tabs. To overcome the problem with scanf, C provides the **gets** function. It allows us to read a line of characters (including spaces and tabs) until the newline character is entered, i. e., the Enter key is pressed. A call to this function takes the following form:

gets(sentence);

where, sentence is an array of char, i.e., a character string. The function reads characters entered from the keyboard until newline is entered and stores them in the argument string sentence, the newline character is read and converted to a null character (\0) before it is

stored in s. C also provides another function named puts to print a string on the display. A typical call to this function takes the following form:

puts(sentence);

where, sentence is an array of characters, i.e., a character string. This string is printed on the display followed by a newline character.

Arrays of Strings

One string is an array of characters, an array of strings is a two-dimensional array of characters in which each row is one string. An array of strings can be initialized at declaration in the following manner:

```
char day[7][10] = {"Sunday" , "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"}
```

Exercise

1. Write a program that reads the numbers from user and store these numbers into an array of same size. Find and display the sum of all positive numbers and calculate the average.
2. Write a C program to read elements in a matrix and check whether matrix is Sparse matrix or not. **Logic:** To check whether a matrix is sparse matrix we only need to check the total number of elements that are equal to zero. The matrix is sparse matrix if $T \geq ((m * n) / 2)$ where T defines total number of zero elements where m and n are rows and columns respectively.
3. Write down a program which asks user to input his first name and last name in a separate array. After taking the input your program should be able to concatenate first name and last name and return it as full name. Count down number of characters in the full name as well.
For example: First name: Muhammad, Second name: Ahmed, Full name: Muhammad Ahmed
4. You taking a square matrix as input from keyboard and then you transpose the same matrix after meeting the requirements you are also interested to find out whether original Matrix A and transpose of Matrix A are equal are not. If the answer is yes, then you print the matrix along with message "matrix is symmetric" otherwise you print the "matrix is asymmetric".
5. Write a program which takes a matrix of any size as user input and returns the maximum element of matrix as output. Your code should also show the entered matrix on the screen.

Programming Fundamentals CT-175

Dynamic Memory Allocation

Objectives

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to understand the basics of

programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to

understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The objective of this lab is to implementing strings in C and to utilize them in an efficient and suitable manner.

Tools Required

DevC++ IDE

Course Coordinator –

Course Instructor –

Lab Instructor –

Department of Computer Science and Information Technology

NED University of Engineering and Technology

Strings in C

In C, a string is not a formal data type; instead, it is an array of type `char`. Most languages internally treat strings as character arrays, but somehow conceal this fact from the programmer. Character arrays or strings are used by programming languages to manipulate text as words and sentences.

A string is one dimensional array of characters terminated by a null character (`'\0'`) which has a numerical value of 0.

Declaration and initialization

A string can be declared as follow.

```
char string [8];
```

This will occupy eighty bytes in memory for the string. C concedes the fact that you would use strings very often and hence provides a shortcut for initializing strings. For Example,

```
char string [] = {'H', 'E', 'L', 'L', 'O', '\0'};
```

Each character in the array occupies one byte of memory and the last character is always `'\0'`. Ascii value of `'\0'` is 0 whereas value of `'O'` is 48.

The above string can also be initialized as

```
char [] = "HELLO";
```

Note that in this declaration `'\0'` is not necessary. C inserts the null character automatically.

Example:

```
int main (void)
{
    char str[80] = "anonymous";
    printf("%s", str);           //this will print HELLO
    printf("%c",str[2]);        //this will print the L
    printf ("%c",str[4]);       //this will print O
}
```

Traversing String

Traversing the string is one of the most important aspects in any of the programming languages. We may need to manipulate a very large text which can be done by traversing the text. Traversing string is somewhat different from the traversing an integer array. We need to know the length of the array to traverse an integer array, whereas we may use the null character in the case of string to identify the end the string and terminate the loop.

Hence, there are two ways to traverse a string.

- By using the length of string
- By using the null character.

Using the length of string

Let's see an example of counting the number of vowels in a string.

```
#include<stdio.h>
void main ()
{
    char s[11] = "hello world";
    int i = 0;
    int count = 0;
    while(i<11)
    {
        if(s[i]=='a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'u' || s[i] == 'o')
        {
            count ++;
        }
        i++;
    }
    printf("The number of vowels %d",count);
}
```

Let's see the same example of counting the number of vowels by using the null character.

```
#include<stdio.h>
void main ()
{
    char s[11] = "javatpoint";
    int i = 0;
    int count = 0;
    while(s[i] != NULL)
    {
        if(s[i]=='a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'u' || s[i] == 'o')
        {
            count ++;
        }
        i++;
    }
    printf("The number of vowels %d",count);
}
```

Input a String in C

Scanf can also be used in the case of strings but with a different scenario. Consider the below code which stores the string while space is encountered.

```
#include<stdio.h>
void main ()
{
    char s[20];
    printf("Enter the string?");
    scanf("%s",s);
    printf("You entered %s",s);
}
```

Above code will not work for space separated strings. To make this code working for the space separated strings, the minor changed required in the scanf function, i.e., instead of writing `scanf("%s",s)`, we must write: `scanf("%[^\\n]s",s)` which instructs the compiler to store the string `s` while the new line (`\\n`) is encountered. Let's consider the following example to store the space-separated strings.

```
#include<stdio.h>
void main ()
{
    char s[20];
    printf("Enter the string?");
    scanf("%[^\\n]s",s);
    printf("You entered %s",s);
}
```

We do not need to use address of (&) operator in scanf to store a string since string `s` is an array of characters and the name of the array, i.e., `s` indicates the base address of the string (character array) therefore we need not use & with it.

C gets() and puts() functions

The `gets()` and `puts()` are declared in the header file `stdio.h`. Both the functions are involved in the input/output operations of the strings.

C gets() function

The `gets()` function enables the user to enter some characters followed by the enter key. All the characters entered by the user get stored in a character array. The null character is added to the array to make it a string. The `gets()` allows the user to enter the space-separated strings. It returns the string entered by the user.

```
#include<stdio.h>
void main ()
{
    char s[30];
    printf("Enter the string? ");
    gets(s);
    printf("You entered %s",s);
}
```

The `gets()` function is risky to use since it doesn't perform any array bound checking and keep reading the characters until the new line (enter) is encountered. It suffers from buffer overflow, which can be avoided by using `fgets()`. The `fgets()` makes sure that not more than the maximum limit of characters are read. Consider the following example.

```
#include<stdio.h>
void main()
{
    char str[20];
    printf("Enter the string? ");
    fgets(str, 20, stdin);
    printf("%s", str);
}
```


C puts() function

The puts() function is very much similar to printf() function. The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function. The puts() function returns an integer value representing the number of characters being printed on the console. Since, it prints an additional newline character with the string, which moves the cursor to the new line on the console, the integer value returned by puts() will always be equal to the number of characters present in the string plus 1.

```
#include<stdio.h>
#include <string.h>
int main(){
    char name[50];
    printf("Enter your name: ");
    gets(name); //reads string from user
    printf("Your name is: ");
    puts(name); //displays string
    return 0;
}
```

String Library Function: (use header <string.h>)**1. strlen()**

Purpose: use length to find the length of a string, not counting the terminating null character.

Returns: returns the length of string.

strlen (string);

Example:

```
int main (void)
{
    char string [50];
    int len;

    printf("Enter the string:");
    gets(string);
    len = strlen(string);
    printf("%d",len);
}
```

2. strcpy()

Purpose: use strcpy() to copy one string to another.

Returns: The strcpy() function returns a pointer to the copied string (i.e., it returns string.)

char strcpy (char dest , const char src)

Example:

```
int main(void) {
    char string [ 10 ];
    char str2 [ ] = "Pakistan";
    printf ("%s \n", str2 );
    strcpy ( string , str2 );
```

Exercise

1. As a programmer, you are required to create a program that takes the first and last name from a user. The program then combines both the inputs taken and prints the string backwards.
2. Each student is required to find out the maximum frequency of characters occurring in their name and the courses offered in Fall 2021. To find it, the student enters their name, courses offered and the program finds the maximum occurrences of a character in the name and course. Course names should be used like Programming Fundamentals, Applied Physics, Pakistan Studies and so on.
3. Students are grouped in two to complete a lab task. Each student is required to enter a string of their own choice as an input to the program. The program will then display as a result whether both the strings are equal. If the strings are not equal, the program will display which of the string is greater.

Test cases:

Enter two strings that are same.

Enter two different strings.

4. Write down the output of the following program.

```
int main (void)
{
    char a[11] = "hello world";
    int i;
    for(i = 0; i <= 9; i++)
    {
        a[i] = a[i + 1];
        printf("%d \t %s \n", i, a);
    }
    Printf("\n %d", a);}
```

Programming Fundamentals CT-175

Dynamic Memory Allocation

Objectives

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to understand the basics of

programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to

understand the basics of

problem solving in

programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The objective of this lab is to enable students create user defined functions by passing and returning different types of arguments. By the end of this lab students will be able to write user defined functions in C.

Tools Required

DevC++ IDE

Course Coordinator –

Course Instructor –

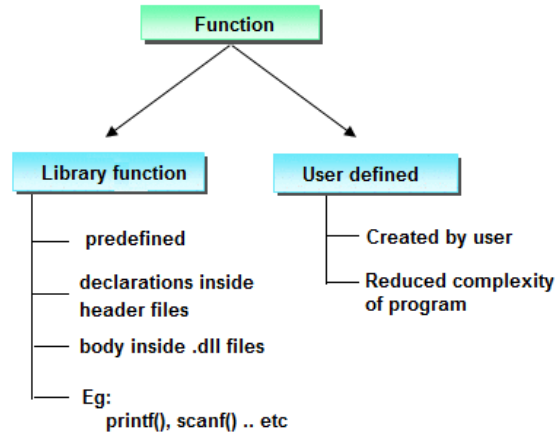
Lab Instructor –

Department of Computer Science and Information Technology

NED University of Engineering and Technology

Functions in C

A large program can be divided into basic building blocks, for performing specific tasks, known as functions. A function contains the set of programming statements enclosed by curly braces { }. The function is also known as procedure or subroutine in other programming languages. Functions are mainly classified into two categories, that is, library and user defined functions.



Following are a few benefits of dividing a large program in to small units called functions.

- ✓ It provides modularity to your program's structure.
- ✓ It makes your code reusable. You just have to call the function by its name to use it, wherever and whenever required.
- ✓ In case of large programs with thousands of code lines, debugging and editing becomes easier if you use functions.
- ✓ It makes the program more readable and easy to understand.

User Defined Functions

There are three steps for defining user defined functions in C, that is, function declaration, definition, and call.

Function Declaration or Function Prototype

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body. A function prototype gives information to the compiler that the function may later be used in the program. The function prototype is not needed if the user-defined function is defined before the main() function. Syntax for function declaration is as follows.

```
return_data_type functionName(data_type1 argument1, data_type2 argument2,...);
int add(int a, int b);
```

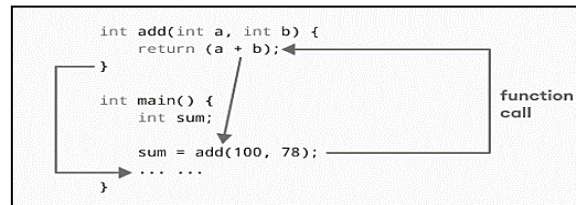
Function Definition

Function definition contains the block of code to perform a specific task and returning the result. Syntax for function definition is as follows.

```
return_data_type functionName(data_type1 argument1, data_type2 argument2,{
//body of the function
})
```

Function Call

When a function is called, the control of the program is transferred to the function definition. And, the compiler starts executing the codes inside the body of a function. In programming argument refers to the variable passed to the function.



In the above example, we have function definition and functions call which provides following information to the compiler:

- ✓ name of the function is `add()`
- ✓ return type of the function is `int`
- ✓ two arguments (100, 78) of type `int` are passed to the function. The parameters `a` and `b` accepts the arguments passed in the function definition. These arguments are called formal parameters of the function and store the copy of actual parameters.
- ✓ The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after return statement. In the above example, we have the data type `int` instead of `void`. This means that the function returns an `int` value.
- ✓ In the above example, function call is made using `add(100,78);` statement inside the `main()`.

User Defined Head Files

The purpose to understand and learn header file is that, it also contains specific function defines in it. You are required to call its header and can use its defined function in your program. Header file serves two purposes.

- ✓ You include them in your program to supply the definitions and declarations you need to invoke system calls and libraries.
- ✓ Your own header files contain declarations for interfaces between the source files of your program. Each time you have a group of related declarations and macro definitions all or most of which are needed in several different source files.

Follow the step to create your header file.

- ✓ Make a header file with `.h` extension and give it unique name e.g `sumfile-> sumfile.h`
- ✓ Define your program in header extension file.
 - `int add(int a,int b){`
`return(a+b);`
`}`
- ✓ Make source file of `c`, where your main program is set.

```

#include<stdio.h>
#include "sumfile.h" // don't use '<>', instead of it use"".
void main(){ int num1 = 10, num2 = 10, num3;
num3 = add(num1, num2);

```

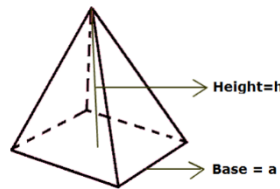
```
printf("Addition of Two numbers : %d", num3);
}
```

- ✓ Keep .h file path directory same as source file.
- ✓ In the above program the 'add' function is basically called from the heder file of sumfile.h which we have explicitly defined.

Exercise

- Write a function that prints all the unique values from an array and the number of times each value occurred. The main function takes a size of array as input and generates a random integer array with name **"array1"**. Random number limit must be between 0 and 10. The 'main' function calls a function with the name as "CountFrequency()" that will find the occurrence of each value in array.
- Salesflow is one of leading software house they are starting their recruitment process for three different following positions: Associate Developer, Assistant Developer, Trainee Engineer. There is a defined criterion for recruitment process: if candidate clears the test with 50 marks, he will be selected for the post of trainee engineer; experience is not the required for this post. If candidate secures 60 marks with at least one year of experience and 70 marks with at least 2 years of experience, then he/she will be selected as an assistant and associate developer, respectively. Write a function that takes the test marks from user and ask for experience (if the entered marks are $x \geq 60$). After that, function shows the assigned position.
- Write the program that calculate the volume by using following formula

$$V = a^2 * 1/3 h,$$



by creating two separate functions. One of the functions with prototype "getData(int h, int a)", takes two inputs from user. The other function with prototype "volumeCal()" calculates the volume, and this function must be called from the first function "getData ()". The first function must be called from the main function.

- Write a program that takes a positive number with a fractional part and rounds it to two decimal places. For example, 32.4851 would round to 32.49, and 32.4431 would round to 32.44.
- In shopping for a new house, you must consider several factors. In this problem the initial cost of the house, the estimated annual fuel costs, and the annual tax rate are available. Write a program that will determine the total cost of a house after a five-year period and run the program for each of the following sets of data.

Initial House Cost	Annual Fuel Cost	Tax Rate
67,000	2,300	0.025
62,000	2,500	0.025
75,000	1,850	0.020

To calculate the house cost, add the initial cost to the fuel cost for five years, then add the taxes for five years. Taxes for one year are computed by multiplying the tax rate by the initial cost. Write and call a function that displays instructions to the program user.

Programming Fundamentals CT-175

Dynamic Memory Allocation

Objectives

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to understand the basics of

programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to

understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The objective of this lab is to understand and implement structures in C and to utilize them in an efficient and suitable manner.

Tools Required

DevC++ IDE

Course Coordinator –

Course Instructor –

Lab Instructor –

Department of Computer Science and Information Technology

NED University of Engineering and Technology

Structures in C

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 structure. When a variable of that structure is made then memory is allocated. This is equivalent to saying that there is no memory for, but when we declare an integer i.e.

int var;

only then memory is allocated.

The struct keyword defines a structure type followed by an identifier (name of the structure). Then inside the curly braces, you can declare one or more members (declare variables inside curly braces) of that structure. For example:

```
struct Person {
    char name[50];
    int age;
    float salary;
};
```

Once a structure person is declared a structure variable is defined as:

Person bill;

A structure variable bill is of type structure Person. When structure variable is defined, only then the required memory is allocated by the compiler. Considering either 32-bit or 64-bit system, the memory of float is 4 bytes, memory of int is 4 bytes and memory of char is 1 byte. Hence, 58 bytes of memory is allocated for structure variable bill.

The members of structure variable are accessed using a **dot (.)** operator. Suppose, you want to access *age* of structure variable *bill* and assign it 50 to it. You can perform this task by using following code below:

Bill.age = 50;

Example

```
#include <stdio.h>
struct Person{
    char name[50];
    int age;
    float salary; };
int main(){
    Person p1;
    printf ( "Enter Full name: ");
    gets(p1.name);
    printf ( "Enter age: ");
    scanf("%d",&p1.age);
    printf ( "Enter salary: ");
    scanf ("%d",&p1.salary);
    printf ( "\nDisplaying Information. \n" );
    printf ( "Name: %s \n", p1.name);
    printf ("Age: %d \n", p1.age );
    printf ( "Salary: %f \n", p1.salary);
}
```

Exercise

1. Declare a structure named employee that stores the employee id, name, salary and department.
2. Take data input from user after declaring a variable of employee type and show the data in proper format on output screen.
3. A phone number, such as (212) 767-8900, can be thought of as having three parts: e.g., the area code (212), the exchange (767), and the number (8900). Write a program that uses a structure to store these three parts of a phone number separately. Call the structure phone. Create two structure variables of type phone. Initialize one, and have the user input a number for the other one. Then display both numbers.
 The interchange might look like this:
 Enter area code: 415
 Enter exchange: 555
 Enter number: 1212
 Then display like below:
 My number is (212) 767-8900
 Your number is (415) 555-1212
4. Define a structure to store the following student data: CGPA, courses (course name, GPA), address (consisting of street address, city, state, zip). Input 2 student records, compare and display which student have highest GPA in which course also Display which student has the highest CGPA . HINT: define another structure to hold the courses and address.
5. Write a C program that uses functions to perform the following operations:
 - i) Reading a complex number
 - ii) Writing a complex number
 - iii) Addition of two complex numbers
 - iv) Multiplication of two complex numbers
 (Note: represent complex numbers using a structure.)

Programming Fundamentals CT-175

Dynamic Memory Allocation

Objectives

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to

understand the basics of
problem solving in
programmatic context.

Students will be representing
solutions to different
problems as

flowcharts and pseudo code

The purpose of this lab is to
enable students to

understand the basics of
problem solving in
programmatic context.

Students will be representing
solutions to different
problems as

flowcharts and pseudo code

The objective of this lab is to familiarize students with the concept of pointers. By the end of this lab students will be able declare, initialize and assign values to pointers, use void and constant pointers, explore relationship between arrays and pointers and pass arguments to functions using pointers

Tools Required

DevC++ IDE

Course Coordinator –
Course Instructor –
Lab Instructor –
Department of Computer Science and Information Technology
NED University of Engineering and Technology

Pointers in C

A pointer is a variable that stores the memory address of another variable as its value. Pointers enable programs to simulate pass-by-reference, to pass functions between functions, and to create and manipulate dynamic data structures, i.e., data structures that can grow and shrink at execution time, such as linked lists, queues, stacks and trees.

Declaring a Pointer

Pointers, like all variables, must be defined before they can be used. The definition:

```
int *countPtr;
```

specifies that variable countPtr is of type int * (i.e., a pointer to an integer) and is read (right to left), “countPtr is a pointer to int” or “countPtr points to an object of type int.”

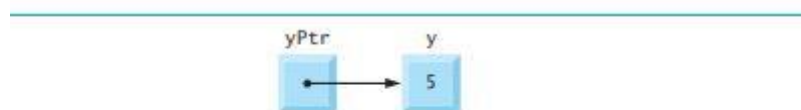
Initializing and Assigning Values to a Pointer

Pointers should be initialized when they’re defined, or they can be assigned a value. The **&**, or **address operator**, is a unary operator that returns the address of its operand.

Example 01: Declaring and assigning value to pointer.

```
#include <stdio.h>
int main( ){
    int y = 5;
    int *yPtr;
    yPtr = &y;
}
```

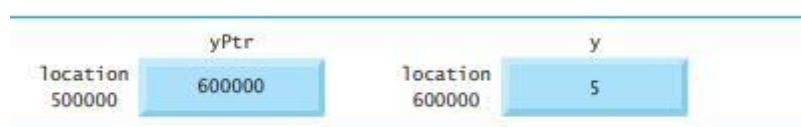
For example, assuming the definitions, the above example assigns the address of the variable y to pointer variable yPtr. Variable yPtr is then said to “point to” y.



Graphical representation of a pointer pointing to an integer variable in memory.

Pointer Representation in Memory

The below figure shows the representation of the pointer in memory, assuming that integer variable y is stored at location 600000, and pointer variable yPtr is stored at location 500000. The operand of the address operator must be a variable; the address operator cannot be applied to constants or expressions.



Representation of y and yPtr in memory.

The Indirection (*) Operator

The unary * operator, commonly referred to as the indirection operator or dereferencing operator, returns the value of the object to which its operand (i.e., a pointer) points. For example, the statement

```
printf( "%d", *yPtr );
```

prints the value of variable y, namely 5. Using * in this manner is called **dereferencing a pointer**.

Void Pointer

The void pointer is a pointer which is not associated with any data types. As the name suggests, the void pointer indicates that the pointer is basically empty- and thus, it is capable of holding any data type address in the program. It is also called general purpose pointer. The syntax:

```
void *ptr;
```

Constant Pointer

A **constant pointer** is a pointer that cannot change the address it is containing. In other words, we can say that once a constant pointer points to a variable then it cannot point to any other variable. These pointers can change the value of the variable they point to but cannot change the address they are holding. Compile and run the below mentioned example. What do you think will happen?

Example 02: Using constant pointer.

```
#include <stdio.h>
int main( ){
    int y = 5;
    int z = 6;
    int const *yPtr = &y;
    printf("%p\n", yPtr);
    *yPtr = &z;
    printf("%p\n", yPtr);
}
```

Pointers and Arrays

Arrays and pointers are intimately related in C and often may be used interchangeably. An array name can be thought of as a constant pointer. Pointers can be used to do any operation involving array subscripting. Assume that integer array b[5] and integer pointer variable bPtr have been defined. Because the array name (without a subscript) is a pointer to the first element of the array, we can set bPtr equal to the address of the first element in array b with the statement

```
bPtr = b;
```

This statement is equivalent to taking the address of the array's first element as follows:

```
bPtr = &b[ 0 ];
```

Array element b[3] can alternatively be referenced with the pointer expression

```
*( bPtr + 3 )
```

The 3 in the expression is the **offset** to the pointer. When the pointer points to the array's first element, the offset indicates which array element should be referenced, and the offset value is identical to the array subscript. This notation is referred to as **pointer/offset notation**. The parentheses are necessary because the precedence of * is higher than the precedence of +. Without the parentheses, the above expression would add 3 to the value of the expression *bPtr (i.e., 3 would be added to b[0], assuming bPtr points to the beginning of the array).

Example 03: Using pointers to access & manipulate array elements.

```
#include<stdio.h>
int main()
{
    int arr[3];

    // declare pointer variable
    int *ptr;
    // declare loop iterator variable
    int i;

    // ptr = &arr[0]
    ptr = arr;

    // use for loop to put values in all array elements using pointer notation
    for (i = 0; i < 3; ++i)
    {
        *(ptr+i) = i+1;
    }

    // use for loop to print values of all array elements using pointer notation
    printf("\nDisplaying value using pointers: ");
    for (i = 0; i < 3; i++)
    {
        printf("%d\n", *(ptr+i));
    }

    // use for loop to print addresses of all array elements using pointer notation
    printf("\nDisplaying address using pointers: ");
    for (i = 0; i < 3; i++)
    {
        printf("%p\n", ptr+i);
    }

    return 0;
}
```

Passing Arguments to Functions by Reference

There are two ways to pass arguments to a function—**pass-by-value** and **pass-by-reference**. All arguments in C are passed by value. As we know, **return** may be used to return one value from a called function to a caller (or to return control from a called function without passing back a value).

Many functions require the capability to modify variables in the caller or to pass a pointer to a large data object to avoid the overhead of passing the object by value (which incurs the time and memory overheads of making a copy of the object).

In C, you use **pointers** and the **indirection operator** to simulate **pass-by-reference**. When calling a function with arguments that should be modified, the addresses of the arguments are passed. This is normally accomplished by applying the address operator (&) to the variable (in the caller) whose value will be modified.

Example 04: Find out cube of a number using passing by reference.

```
#include <stdio.h>

void cubeByReference(int *nPtr) {
    *nPtr = *nPtr * *nPtr * *nPtr;
}

int main( void ) {
    int number = 5; // initialize number
    printf( "The original value of number is %d \n", number );
    //pass address of the number to cubeByReference
    cubeByReference( &number);
    printf( "The new value of number is %d", number );
}
```

Exercise

7. For each of the following, write a single statement that performs the indicated task. Assume that long integer variables value1 and value2 have been defined and that value1 has been initialized to 200000.
 - a) Define the variable IPtr to be a pointer to an object of type long.
 - b) Assign the address of variable value1 to pointer variable IPtr.
 - c) Print the value of the object pointed to by IPtr.
 - d) Assign the value of the object pointed to by IPtr to variable value2.
 - e) Print the value of value2.
 - f) Print the address of value1.
 - g) Print the address stored in IPtr. Is the value printed the same as the address of value1?
8. Write a program to implement the exchange or swap the values of 3 variables{a,b,c}. To implement this, you need to write a function called swaped().

```
void swaped(int *aPtr, int *bPtr, int *cPtr);
```

such that; b ----> temp

a ----> b

```
c ----> a
```

```
temp -> a
```

9. Write a program that calculates the sum of all the elements in array using pointers.
10. Write a program that finds the second highest number in a float type array of 20 elements using pointer.
11. Write a program that implements the SortFunction that takes argument pointer to an array, size of the array and the order in which it is going to be sort. Such as, 1 for Ascending order and 2 for Descending orde. Finally, print this array in Main() to check.

```
void SortFunction(int *arr, int *size, int order);
```

12. Declare an array of int at least 10 elements long. Fill in the array with the square of its index using array syntax, `a[i] = i * i ;`. Print out the array using pointer syntax `*(a + i)`.

13. Consider the following program:

```
#include <stdio.h>
int main( void ) {
    int *p1;
    char *p2;
    p2=p1;
}
```

Will the program compile successfully without warnings? Why and why not?

Programming Fundamentals CT-175

Dynamic Memory Allocation

Objectives

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to

understand the basics of
problem solving in
programmatic context.

Students will be representing
solutions to different
problems as

flowcharts and pseudo code

The purpose of this lab is to
enable students to

understand the basics of
problem solving in
programmatic context.

Students will be representing

solutions to different problems as flowcharts and pseudo code

The objective of this lab is to familiarize students with dynamic allocation of memory. By the end of this lab students will be able to allocate memory using `malloc()` and `calloc()` functions and free it using `free()` function.

Tools Required

DevC++ IDE

Course Coordinator –

Course Instructor –

Lab Instructor –

Department of Computer Science and Information Technology
NED University of Engineering and Technology

Introduction

Creating and maintaining dynamic data structures requires dynamic memory allocation—the ability for a program **to obtain more memory space at execution time to hold new nodes, and to release space no longer needed.**

sizeof Operator

There is a useful operator in C called the **sizeof** operator. You can use it to determine the size of a particular variable in memory. You can also use it to determine the size of a type.

We will use this operator to dynamically allocate memory space.

The number of elements in an array also can be determined with sizeof. For example, consider the following array definition:

```
double real[ 22 ];
```

Variables of type double normally are stored in 8 bytes of memory. Thus, array real contains a total of 176 bytes. To determine the number of elements in the array, the following expression can be used:

```
sizeof( real ) / sizeof( real[ 0 ] );
```

Example 01: sizeof returns size of each data type

```
#include <stdio.h>
int main( ){
    printf("%lu\n", sizeof(char));
    printf("%lu\n", sizeof(int));
    printf("%lu\n", sizeof(float));
    printf("%lu", sizeof(double));
    return 0;
}
```

malloc() Function

The name "malloc" stands for memory allocation. The malloc() function **reserves a block of memory of the specified number of bytes** and it returns a pointer of void which can be casted into pointers of any form. Function malloc is normally used with the sizeof operator. For example,

```
ptr = (float*) malloc(100 * sizeof(float));
```

The above statement allocates 400 bytes of memory. It's because the size of float is 4 bytes and the pointer ptr holds the address of the first byte in the allocated memory.

The expression results in a NULL pointer if the memory cannot be allocated.

Example 02: Taking number of elements of array as user input and initializing it using malloc().

```
#include <stdio.h>
#include <stdlib.h>
int main( ){
    // This pointer will hold the base address of the block created
    int *ptr;
    int n, i;

    // Get the number of elements for the array
```

```

printf("Enter number of elements:");
scanf("%d",&n);
printf("Entered number of elements: %d\n", n);

// Dynamically allocate memory using malloc()
ptr = (int*)malloc(n * sizeof(int));

// Check if the memory has been successfully allocated by malloc or not
if (ptr == NULL) {
    printf("Memory not allocated.\n");
    exit(0);
}
else {
    // Memory has been successfully allocated
    printf("Memory successfully allocated using malloc.\n");

    // Get the elements of the array
    for (i = 0; i < n; ++i) {
        ptr[i] = i + 1;
    }
    // Print the elements of the array
    printf("The elements of the array are: ");
    for (i = 0; i < n; ++i) {
        printf("%d, ", ptr[i]);
    }
}
return 0;
}

```

calloc() Function

“calloc” or “contiguous allocation” method in C is used to dynamically allocate the **specified number of blocks of memory** of the specified type. it is very much similar to malloc() but has two differences and these are:

- It initializes each block with a default value ‘0’.
- It has two parameters or arguments as compare to malloc().

Consider the following line of code:

```
ptr = (float*) calloc(25, sizeof(float));
```

This statement allocates contiguous space in memory for 25 elements each with the size of the data type float.

Example 03: Taking number of elements of array as user input and initializing it using calloc().

```

#include <stdio.h>
#include <stdlib.h>

int main(){
    // This pointer will hold the base address of the block created

```

```

int *ptr;
int n, i;

// Get the number of elements for the array
n = 5;
printf("Enter number of elements: %d\n", n);

// Dynamically allocate memory using calloc()
ptr = (int*)calloc(n, sizeof(int));

// Check if the memory has been successfully allocated by calloc or not
if (ptr == NULL) {
    printf("Memory not allocated.\n");
    exit(0);
}
else {
    // Memory has been successfully allocated
    printf("Memory successfully allocated using calloc.\n");

    // Get the elements of the array
    for (i = 0; i < n; ++i) {
        ptr[i] = i + 1;
    }
    // Print the elements of the array
    printf("The elements of the array are: ");
    for (i = 0; i < n; ++i) {
        printf("%d, ", ptr[i]);
    }
}
return 0;
}

```

free() Function

Dynamically allocated memory created with either `calloc()` or `malloc()` doesn't get freed on their own. You must explicitly use `free()` to release the space. Following is its syntax:

```
free(ptr);
```

Example 04: Allocating memory using `malloc()` and `calloc()` and then deallocating it using `free()`.

```

#include<stdio.h>
int main(){
    // These pointers will hold the base address of the blocks created
    int *ptr, *ptr1;

    // Dynamically allocate memory using malloc()
    ptr = (int*)malloc(5 * sizeof(int));

    // Dynamically allocate memory using calloc()
    ptr1 = (int*)calloc(5, sizeof(int));
}

```

```

// Check if the memory has been successfully
// allocated by malloc or not
if (ptr == NULL || ptr1 == NULL) {
    printf("Memory not allocated.\n");
    exit(0);
}
else {
    // Memory has been successfully allocated
    printf("Memory successfully allocated using malloc.\n");

    // Free the memory
    free(ptr);
    printf("Malloc Memory successfully freed.\n");

    // Memory has been successfully allocated
    printf("\nMemory successfully allocated using calloc.\n");

    // Free the memory
    free(ptr1);
    printf("Calloc Memory successfully freed.\n");
}
return 0;
}

```

Memory Leaks

A memory leak is an issue that occurs when all pointers to a block of dynamically-allocated memory are lost before the block of memory is freed. Memory allocated on the heap persists beyond function calls and the computer has no way of knowing when a block of memory is not needed any more. In Java, this is handled by garbage collection, but there are no such niceties in C. The programmer must explicitly release a block of memory back into the heap using the `free()` command when it is done being used.

Exercise

- Write a program that does the following:
 - Ask the user to type the size of the array.
 - Use `malloc` or `calloc` to create an integer array of that size.
 - Use the function `read` to read the numbers.
 - Display the sum and average of these numbers. Then display the array sorted.
 - ✓ Show 2 numbers after the floating point in the average.
 - Free the allocated memory.
- Write a program that ask the user to enter the total 'N' no of characters in user's name {First Name + Last Name} to create a dynamic array of characters. After create a dynamic array of that 'N' no of characters using `malloc` or `calloc` function. Finally copy your full name in it that has already been taken from the user before


```
Dynamic Array = "Muhib Ahmed";
```
- Using above question (2), resize that dynamic array of character and append the array with your studentId. That student id must be taken input from the user.


```
DynamicArray = "Muhib Ahmed";           // Before
DynamicArray = "K211234 Muhib Ahmed";    // After the text append
```


Programming Fundamentals CT-175

File Handling in C Language.

Objectives

The purpose of this lab is to enable students to understand the basics of problem solving in programmatic context.

Students will be representing solutions to different problems as

flowcharts and pseudo code

The purpose of this lab is to enable students to

understand the basics of
problem solving in
programmatic context.

Students will be representing
solutions to different
problems as

flowcharts and pseudo code

The purpose of this lab is to
enable students to

understand the basics of
problem solving in
programmatic context.

Students will be representing

solutions to different problems as flowcharts and pseudo code

The objective of this lab is to enable students handle files in C language.

Tools Required

DevC++

Course Coordinator –
Course Instructor –
Lab Instructor –
Department of Computer Science and Information Technology
NED University of Engineering and Technology

Filing

A file is a container in computer storage devices used for storing data. In this lab, we will be learning about file handling in C. We will learn to handle standard I/O in C using `fprintf()`, `fscanf()`, `fread()`, `fwrite()`, `fseek()` etc. with the help of examples.

Why files are needed?

When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates. If you have to enter a large number of data, it will take a lot of time to enter them all. However, if you have a file containing all the data, you can easily access the contents of the file using a few commands in C. You can easily move your data from one computer to another without any changes.

Types of Files

When dealing with files, there are two types of files you should know about:

1. Text files
2. Binary files

1. Text files

Text files are the normal .txt files. You can easily create text files using any simple text editors such as Notepad. When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents. They take minimum effort to maintain, are easily readable, and provide the least security and takes bigger storage space.

2. Binary files

Binary files are mostly the .bin files in your computer. Instead of storing data in plain text, they store it in the binary form (0's and 1's). They can hold a higher amount of data, are not readable easily, and provides better security than text files.

File Operations

In C, you can perform four major operations on files, either text or binary:

1. Creating a new file
2. Opening an existing file
3. Closing a file
4. Reading from and writing information to a file

Working with files

When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and the program.

```
FILE *fptr;
```

Opening a file - for creation and edit

Opening a file is performed using the `fopen()` function defined in the `stdio.h` header file. The syntax for opening a file in standard I/O is:

```
ptr = fopen("fileopen","mode");
```

For example,

```
fopen("E:\\cprogram\\newprogram.txt","w");  
fopen("E:\\cprogram\\oldprogram.bin","rb");
```

Let's suppose the file newprogram.txt doesn't exist in the location E:\cprogram. The first function creates a new file named newprogram.txt and opens it for writing as per the mode 'w'. The writing mode allows you to create and edit (overwrite) the contents of the file. Now let's suppose the second binary file oldprogram.bin exists in the location E:\cprogram. The second function opens the existing file for reading in binary mode 'rb'. The reading mode only allows you to read the file, you cannot write into the file.

Opening Modes in Standard I/O

Mode	Meaning of Mode	During Inexistence of file
R	Open for reading.	If the file does not exist, fopen() returns NULL.
Rb	Open for reading in binary mode.	If the file does not exist, fopen() returns NULL.
W	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
Wb	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
A	Open for append. Data is added to the end of the file.	If the file does not exist, it will be created.
Ab	Open for append in binary mode. Data is added to the end of the file.	If the file does not exist, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.

Opening Modes in Standard I/O		
Mode	Meaning of Mode	During Inexistence of file
rb+	Open for both reading and writing in binary mode.	If the file does not exist, fopen() returns NULL.
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
wb+	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exist, it will be created.
ab+	Open for both reading and appending in binary mode.	If the file does not exist, it will be created.

Closing a File

The file (both text and binary) should be closed after reading/writing. Closing a file is performed using the fclose() function.

```
fclose(fptr);
```

Here, fptr is a file pointer associated with the file to be closed.

Reading and writing to a text file

For reading and writing to a text file, we use the functions fprintf() and fscanf(). They are just the file versions of printf() and scanf(). The only difference is that fprintf() and fscanf() expects a pointer to the structure FILE.

Example 1: Write to a text file

```

#include <stdio.h>
#include <stdlib.h>

int main()
{
    int num;
    FILE *fptr;

    // use appropriate location if you are using MacOS or Linux
    fptr = fopen("C:\\program.txt", "w");

    if(fptr == NULL)
    {
        printf("Error!");
        exit(1);
    }

    printf("Enter num: ");
    scanf("%d", &num);

    fprintf(fptr, "%d", num);
    fclose(fptr);

    return 0;
}

```

This program takes a number from the user and stores in the file program.txt. After you compile and run this program, you can see a text file program.txt created in C drive of your computer. When you open the file, you can see the integer you entered.

Example 2: Read from a text file

```

#include <stdio.h>
#include <stdlib.h>

int main()
{
    int num;
    FILE *fptr;

    if ((fptr = fopen("C:\\program.txt", "r")) == NULL){
        printf("Error! opening file");

        // Program exits if the file pointer returns NULL.
        exit(1);
    }

    fscanf(fptr, "%d", &num);

    printf("Value of n=%d", num);
    fclose(fptr);

    return 0;
}

```

This program reads the integer present in the program.txt file and prints it onto the screen. If you successfully created the file from Example 1, running this program will get you the integer you entered. Other functions like fgetchar(), fputc() etc. can be used in a similar way.

Reading and writing to a binary file

Functions `fread()` and `fwrite()` are used for reading from and writing to a file on the disk respectively in case of binary files.

Writing to a binary file

To write into a binary file, you need to use the `fwrite()` function. The functions take four arguments:

1. address of data to be written in the disk
2. size of data to be written in the disk
3. number of such type of data
4. pointer to the file where you want to write.

```
fwrite(addressData, sizeData, numbersData, pointerToFile);
```

Example 3: Write to a binary file using `fwrite()`

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

struct threeNum
{
    int n1, n2, n3;
};

int main()
{
    int n;
    struct threeNum num;
    FILE *fptr;

    if ((fptr = fopen("C:\\program.bin", "wb")) == NULL){
        printf("Error! opening file");

        // Program exits if the file pointer returns NULL.
        exit(1);
    }

    for(n = 1; n < 5; ++n)
    {
        num.n1 = n;
        num.n2 = 5*n;
        num.n3 = 5*n + 1;
        fwrite(&num, sizeof(struct threeNum), 1, fptr);
    }
    fclose(fptr);

    return 0;
}
```

In this program, we create a new file `program.bin` in the C drive. We declare a structure `threeNum` with three numbers - `n1`, `n2` and `n3`, and define it in the main function as `num`.

Now, inside the for loop, we store the value into the file using `fwrite()`. The first parameter takes the address of `num` and the second parameter takes the size of the structure `threeNum`. Since we're only inserting one instance of `num`, the third parameter is 1. And, the last parameter `*fptr` points to the file we're storing the data. Finally, we close the file.

Reading from a binary file

Function `fread()` also take 4 arguments similar to the `fwrite()` function as above.

```
fread(addressData, sizeData, numbersData, pointerToFile);
```

Example 4: Read from a binary file using `fread()`

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

struct threeNum
{
    int n1, n2, n3;
};

int main()
{
    int n;
    struct threeNum num;
    FILE *fptr;

    if ((fptr = fopen("C:\\program.bin", "rb")) == NULL){
        printf("Error! opening file");

        // Program exits if the file pointer returns NULL.
        exit(1);
    }

    for(n = 1; n < 5; ++n)
    {
        fread(&num, sizeof(struct threeNum), 1, fptr);
        printf("n1: %d\\tn2: %d\\tn3: %d\\n", num.n1, num.n2, num.n3);
    }
    fclose(fptr);

    return 0;
}
```

In this program, you read the same file `program.bin` and loop through the records one by one. In simple terms, you read one `threeNum` record of `threeNum` size from the file pointed by `*fptr` into the structure `num`.

EXERCISE

1. Write a program to take input from user and save that text in a file as encrypted text (using Shift cipher along with $K=3$). Then retrieve the encrypted text from the same file and display the original text.

2. Write a C program to keep records and perform statistical analysis for a class of 20 students. The information of each student contains ID, Name, Sex, quizzes Scores (2 quizzes per semester), mid-term score, final score, and total score. All the records must be store in the file and you must read the scores <50, <80 and <100 until users selects the end file option.
3. You're the owner of a hardware store and need to keep an inventory that can tell you what tools you have, how many you have and the cost of each one. Write a program that initializes the file "hardware.txt" to 10 empty records, lets you input the data concerning each tool, enables you to list all your tools, lets you delete a record for a tool that you no longer have and lets you update any information in the file. The tool identification number should be the record number. Use the following information to start your file:

Record #	Tool name	Quantity	Cost
3	Electric sander	7	57.98
17	Hammer	76	11.99
24	Jig saw	21	11.00
39	Lawn mower	3	79.50
56	Power saw	18	99.99
68	Screwdriver	106	6.99
77	Sledge hammer	11	21.50
83	Wrench	34	7.50

4. Using C, create a file named budge.txt that contains three equal-length columns of numbers, like this:

```
-462.13 486.47 973.79
755.42 843.04 -963.67
442.58 -843.02 -462.86
-233.93 -821.67 399.59
-379.65 -556.37 837.46
55.18 -144.93 -93.15
533.73 804.64 -66.25
-922.12 914.68 -264.67
-600.27 -838.59 747.02
-962.97 49.96 -677.79
```

Now write a program named budget.c that reads this file and adds up the numbers in each column. The program's output should look like this:

Column sums are: -1774.16 -105.79 429.47

5. Write a C++ Program to Count Digits, Alphabets and Spaces using File Handling.