Birla Institute of Technology &Science, Pilani
Computer Programming (CS F111)
Lab-3 (Introduction to C Programming Language)

Topics to be covered:
1. C Program structure and execution
2. Data type
3. Reading input and printing output
4. Expression
5. Operator precedency
6. Typecasting

In UNIX the keyboard is defined as a standard input device and the computer monitor is defined as a standard output device. If a command is defined to take input from the standard input, it means it takes its input from the keyboard. Similarly, if a command gives its output to the standard output, it means it displays the output to the monitor. UNIX allows us to temporarily change the standard input and standard output by means of what is called as Indirection & Piping.

1. INDIRECTION OF OUTPUT

- Create a file named as ***file1*** using **vi editor**, write the following contents in the ***file1***:

  A programmer for whom computing is its own reward; may enjoy the challenge of breaking into other computers but does no harm; true hacker subscribe to a code of ethics and look down upon crackers.

- Create another file with a name ***file2*** using **vi editor**, write the following contents in the ***file2***:

  I am a white hat hacker. I have my own code of ethics

- Execute the following two commands:
  ```
  [f2012999@ prithvi ~] $ cat  file1
  [f2012999@ prithvi ~] $ cat  file1 > file3
  ```

  Observe the difference. The first command displays the contents of ***file1*** onto the standard output, i.e. the monitor. Instead, the second command redirected that output to another file, named ***file3***. Observe that this file did not exist but execution of the second command created it. You can look at the contents of ***file3*** using vi editor or cat command.
  Now execute the following command:
  ```
  [f2012999@ prithvi ~] $ cat  file1 > file2
  ```

  Look at the contents of ***file2*** and observe that by executing the above command you have declared ***file2*** as the temporary standard output causing the contents of ***file1*** to be redirected to ***file2***. Since the file ***file2*** is not empty, it will be overwritten. To avoid overwriting the contents of ***file2***, instead of using **>** use **>>**. This will append the contents of ***file1*** to the contents of ***file2.*** Check it out yourself.

- Execute the following commands and observe the result
  ```
  [f2007999@ prithvi ~] $ cal  1  2010  >  cal1
  ```
  Look at the contents of ***cal1***
  ```
  [f2007999@ prithvi ~] $ cal  2  2010  >  cal2
  ```
  Look at the contents of ***cal2***
  ```
  [f2007999@ prithvi ~] $ cat  cal2  >>  cal1
  ```
  Again look at the contents of ***cal1***. Make a note of your observations

d2. INDIRECTION OF INPUT

- Execute the following two command and observe the result:
  [f2012999@ prithvi ~] $ cat
  [f2012999@ prithvi ~] $ cat file2
  In the first case, the cat command takes its input from standard input device, i.e. keyboard. It displays whatever you type on the keyboard. Press ctrl+c to terminate its execution.
  In the second case, the **cat** command takes its input from the file named ***file2*** and sends the result to the standard output (i.e. your monitor). You should note that this command does not take its input from standard input (keyboard); rather it takes the input from a file.

- Execute the following command and observe the result:
  [f2012999@ prithvi ~] $ cat < file2
  Do you find any difference between this command and the previous command? No difference, you are reading the contents of the file ***file2*** and outputting on the standard output (monitor), but the file ***file2*** doesn't become

Exercise -1 (Execute the following commands and record the observations)

Important – In place of whole prompt ***[f2012999@ prithvi ~] $*** only ***$*** symbol is used

standard input for always, after the execution of the command UNIX will automatically make the standard input as keyboard.

- Execute the following command
  [f2012999@ prithvi ~] $ cat < cal1 > testCal1
  List the contents of your directory (use ***ls***), you will find a new file ***testCal1.*** Look at the contents of file ***testCal1***.
  Now execute the following command
  [f2012999@prithvi ~] $ cat cal2 > testCal2
  List the contents of your directory (use ***ls***), again you will find a new file ***testCal2.*** Look at the contents of file ***testCal2***. So what difference do you find in previous two commands?

- The **indirection operators** and usage

> filename	- make file with a name ***filename*** as the standard output
< file	- make file with a name ***filename*** as the standard input
>> file	- make file with a name ***filename*** as the standard output, append to it if it exists

Now try to make out what the following commands achieve

Commands	What does the command do?
$ls > filelist	????
$ ls –l > longFileList	????
$date; cal 2 2010	????
$date; cal 2 2010; cal 3 2010	????
$(date ; ls) > complex	????

1. C PROGRAM STRUCTURE AND EXECUTION FLOW

let's write a simple C program.

Step-1: Typing the program. Using the vi editor create a file, **firstprog.c**

$ vi firstprog.c

Step-2: Type the following program in the file, **firstprog.c**

```
#include<stdio.h>       /* Include Header Files */
int main()              /* main() is the entry point of the program */
{
      printf ("Welcome to C Programming");
      return 0;
}
```

Note: In the above program, the statements that appear enclosed in "/* */" are called comments. These comments are not executable statements. The comments are added to improve the understanding of the program to the reader. These are ignored by the compiler and have no effect on the outcome of the program.

Header files contain definitions of functions and variables which can be incorporated into any C program by using the pre-processor ***#include*** statement. Standard header files are provided with each compiler, and cover a range of areas, string handling, mathematical, data conversion, printing and reading of variables.

To use any of the standard functions, the appropriate header file should be included. This is done at the beginning of the C source file.

For example, to use the function ***printf()*** (will be explained in detail in next program) in a program, the line ***#include <stdio.h>*** should be at the beginning of the source file, because the definition for ***printf()*** is found in the file ***stdio.h***

All header files have the extension .h and generally reside in the /include subdirectory. The use of angle brackets <> informs the compiler to search the compilers include directory for the specified file. ***#include <stdio.h>***

Each instruction in a C program is written as a separate statement, ending with a semicolon. These statements must appear in the same order in which we wish them to be executed.

Step-3: Save and quit from the file, **firstprog.c** (press **ESC** button and type **:wq**)

Step-4: Now, let's run gcc compiler over this source code to create the executable.

```
$gcc firstprog.c -o exeFirstProg
```

In the above command:

- gcc – Invokes the GNU C compiler
- firstprog.c – Your C program file name (this contains your source code)
- -o firstprog – Instructs C compiler to create the executable with a name exeFirstProg. If you don't specify "-o" and the name of the output executable file, by default C compiler will create an output executable with a name a.out

If the program has any errors/mistakes, those will get listed. Most of the times, the error messages also contain the exact line number where we have made the mistake. Errors at this stage would be primarily because of incorrect usage of the syntax (bad typing of the code); these errors are known as syntax errors. Look at the given program carefully and correct errors, if any.

Step-6: Finally, execute **exeFirstProg**, if no errors are displayed in the above step. This step will execute **firstprog.c** program to display the final output.

$./exeFirstProg
Or
$./a.out **if during compilation the name of the output file was not given, i.e. `$gcc firstprog.c`**

Now you have a basic idea about how gcc is used to convert a source code into executable.

2. DATA TYPES

C language supports various data types. Primitive data type stores a single value. Four primitive data types are integer, float, double and char. Integer data type stores whole value. Float and double data type stores integral and fraction value. Double data type has more precision and stores into 64 bits (of course this depends upon the underlying Instruction Set Architecture and compiler). Char data type stores a single character. Although range of primitive type varies from machine to machine. Below table describes size (in bytes) and range of primitive data types based on 16 bits machine.

Data type	Range	Bytes
char	-128 to 127	1
int	-32768 to 32767	2
Float	3.4 e-38 to 3.4 e+38	4
double	1.7e-308 to 1.7 e+308	8

Variable is named location of memory that is used to store different type of values, during the program execution. The values of variable can change during the execution of the program. Variable consists sequence of letters and digits, with a letter as a first character.
For example two variables are declared as follows
`int a=5,b=10; // layout of the memory and variable is shown below.`

1001	5
1002	
1003	10
1004	

A

B

Memory
```
float f=6.23; // f is float variable and it is also initialized
char c ='x'; // c is character variable and char variable is initialized in single quotes
int w,x; // we can declare more than one variable at a time;
```

Integer data type can be prefix with short, long, unsigned and signed keyword in order to increase the range of value. Floating point can be prefix with long keyword. Char can be prefix with signed and unsigned keyword. For example

```
unsigned int = 40000;
```

Although integer can store upto 32767 (refer to above table), due to unsigned its range has increased.

3. READING DATA FROM KEYBOARD

scanf function is used to accept the input from the user through keyboard. This function is defined in **stdio.h** file, that's why we have to include **stdio.h** file. scanf function has two parts; control string and variable/s. The control string specifies which data type (int, char, or float) it should accept. Variable specify the address of the locations where the data is stored. Syntax of scanf() is as follows

```
scanf("control string", &variable1, &variable2 … variableN);
```

Variable name is prefixed with & **(ampersand)** symbol. It is known as "**Address Of**" operator and specifies a memory location to the **scanf** function where the value associated with variable will be stored in your computer's memory. The ampersand "**&**" symbol in **scanf** function is mandatory. Missing the ampersand in **scanf** function is the most common C Programming error because in this case compiler will not produce any compile time error.

Control String:

It contains field specifications which direct the interpretation of input data. It consists of conversion character % and type specifier. It is written in double quotes. Various type specifiers are listed below

Control string	**Meaning**
%d	Integer
%f	Float
%c	Char
%h	Short
%l	Long
%u	Unsigned
%lf	Double
%e	Scientific notation

For example:

```
int a;       // variable is declared
scanf("%d",&a);  // taking input into variable from user
```

Writing on screen:

printf function is used to print output on console or monitor screen. It is similar to **scanf** function, except it does not prefix with ampersand before the variable name. Syntax of printf as follows

```
printf("control string", variable , variable);
printf("any message"); // it will print whatever be written in double quotes
```

Type the following program into userinput.c

```
#include<stdio.h>
int main()
{
 int a;
 int b;
 printf("Enter value of a");
 scanf("%d",&a);                   // Reading user input for the variable 'a'
 printf("Enter the value of b");
 scanf("%d",&b);                  // Reading user input for the variable 'b'
 printf("a= %d",a);
 printf("b= %d",b);
 return 0;
}
```

Compile the program using the command

$gcc userinput.c → **this will create an executable output file with a name "a.out"**

Execute the program using:

$./a.out

3. Expression
It is sequence of operands and operators that reduces into single value. Operands can be variable, constants and an expression. Expression is written as follows

```
variable = expression;
```

When the statement is encountered, the expression is evaluated first, and then the result is stored into the variable. Example of evaluation statements are

```
int a,x,y; // declaration
a=5; // initialization
x= a * 10; // x value will be 50 after evaluating expression
y=10+20; // y value will be 30
a= a*a; // a value will be 25
```

Type and run the following program into sum.c. .

```
#include<stdio.h>
int main()
{
 int a,b,c,sum;
 printf("Enter value of a ,b, c");
 scanf("%d%d%d",&a,&b,&c);              // Reading user input
 sum = a+b+c;
 printf("sum= %d",sum);
  return 0;
}
```

Exercise 1: Write a program to compute the simple interest, input principle, rate and year.
Exercise 2: Write a C program to calculate the area of a circle. Area of a circle of radius R is given by (Area=PI * R*R).

4. Operator
C language has rich set of operators. For example relation operators, logical operator and arithmetic operator. Table-2 lists c operators.

Arithmetic operator: arithmetic operator are used to perform arithmetic operation. It can be binary or unary. Binary operator requires two operands and unary operator requires one operand. For example

int a =10,A;

```
a=a+10 // is a binary operator it requires two operand that can be variable or constant, result
              of this expression is 20;
a=20%2 // is mod operator that works only on integers operands and returns the remainder, result
              of this expression is 0
a++ // is unary operator equivalent to a=a+1, result of this expression is 11
a--; // is unary operator equivalent to a=a-1, result of this expression is 9
++a; // is pre increment unary operator, if it is used in expression, first value of a is
       increment than used in expression.
```

For example

```
A = ++a-10; // A will have 1 value because first a value is incremented
A= 6.07/2.2; // division on real variable.
```

Relation operator: are used to compare two quantities, depending on their relation, it returns the result. Result of relation operator is either true of false. True and false is represent as 1 and 0 respectively. For example

```
 10<20 // its result is 0 that is false.
 10==10 // its comparison of two values, it return 1 that true
```

Logical operator: are used to test more than one condition. Logical operator requires at least relational operator. C has the following three logical operator
&& meaning logical AND (it requires two relation operators, result is true if both conditions are true else false)
|| meaning logical OR (it requires two relation operators, result is true if at least one condition is true)
! meaning logical NOT (its result is true if condition is false)
For example

```
int a=10,b=12;
(a>b) && (a==10) // return false because one condition is false
```

```
(a>b) || (a==10) // returns true because of logical or and condition is true
```

Type the following program into logical.c and see the results

```
#include<stdio.h>
int main()
{
int i=7,result;
float f= 5.5;
char c ='w';
result= (i>= 6) && (c == 'w') // result is 1
printf("result= %d", result);
result = (i>=6) || (c== 119) // result is 1
printf("result= %d", result);
result = (f <11) && (i>100) // result is 0
printf("result= %d", result);
result =( c!='P') ||((i+f)) <=10) // result is 1
printf("result= %d", result);
return 0;
}
```

Conditional operator: A conditional expression is a compound expression that contains three expressions and segregated with (? :)**.** Syntax of conditional operator is

```
 Variable = expression1? expression2 : expression3
```

First, expression1 is evaluated, if the result of first expression is true, then expression2 is evaluated and its value is final result. Otherwise, expression3 is evaluated and its value is final result. For example

```
int m=1,n=2, min;
min = (m < n ? m : n);
```

Here m < n expression evaluates to be true therefore 1 is assigned to min. Type and run the following program into three.c

```
#include<stdio.h>       /* Include Header Files */
int main()              /* main() is the entry point of the program */
{
int a,b,c,min;
printf("enter three numbers");
scanf("%d%d%d",&a,&b,&c);
min= (a< b ? (a < c ? a :c) : (b < c ? b : c));
printf("min =%d",min);
return 0;
}
```

5. OPERATOR PRECEDENCY AND ASSOCIATIVITY

If more than one operators are involved in an expression then, C language has predefined rule of priority of operators. This rule of priority of operators is called operator precedence. You have already done this in your class.
In C, precedence of arithmetic operators(*,%,/,+,-) is higher than relational operators(==,!=,>,<,>=,<=) and precedence of relational operator is higher than logical operators(&&, || and !). Suppose an expression:

```
a + b < c
```

The + (addition) operation is performed before < (less than) operation because of precedence.
Associativity indicates in which order two operators of same precedence (priority) executes. Let us suppose an expression:

```
a+b*c/d
```

the * and / operations are performed before + because of precedence(refer to table 2), b is multiplied by c before it is divided by d because of associativity.
Note: Parentheses are also used to group sub-expressions to force a different precedence; such parenthetical expressions can be nested and are evaluated from inner to outer.

The table 2 shows all the operators in C with precedence and associativity.

Operator Symbol	Name	Category	Associative
! ~ + - ++ -- & * sizeof (type)	Logical negation Bitwise(1 's) complement Unary plus Unary minus Increment Decrement Dereference Operator(Address) Pointer reference Returns the size of an object Type cast(conversion)	Unary operator	Right to left
* / %	Multiply Divide Remainder	Binary operator	Left to right
+ -	Binary plus(Addition) Binary minus(subtraction)	Binary operator	Left to right
< <= > >=	Less than Less than or equal Greater than Greater than or equal	Relation operator	Left to right
== !=	Equal to Not equal to	Relation operator	Left to right
&&	Logical AND	Logical operator	Left to right
||	Logical OR	Logical operator	Left to right
?:	Conditional Operator	Logical operator	Left to right
= *= /= %= -= &= ^= |= <<= >>=	Simple assignment Assign product Assign quotient Assign remainder Assign sum Assign difference Assign bitwise AND Assign bitwise XOR Assign bitwise OR Assign left shift Assign right shift	Assignment operator	Right to left

Table 2 Precedency and associative

Type the following program to evaluate the following expression

x1 = (-b+√ (b^2-4ac))/2a, where ^ denotes x raise to the power y, e.g. b^2
x2 = (-b-√ (b^2-4ac))/2a

Input variables in given formula: a, b, c
Output variables in given formula: x1 and x2

Equivalent C programming expression:

x1 = (-b + sqrt((b*b) – (4*a*c)))/(2*a);
x2 = (-b - sqrt((b*b) – (4*a*c)))/(2*a);

Type the following program in the file, **myroots.c**

```
#include<stdio.h>
#include<math.h>
int main()
{
  double  a,b,c,x1,x2;

  printf("Enter values of a,b and c");
  scanf("%lf %lf %lf",&a,&b,&c);

  x1 = (-b + sqrt((b*b) - (4*a*c)))/(2*a);
  x2 = (-b - sqrt((b*b) - (4*a*c)))/(2*a);

  printf("The first root of the quadratic eqn is %lf\n", x1);
  printf("The second root of the quadratic eqn is %lf\n", x2);
  return 0;
}
```

Why is "math.h" included?
Since we are using sqrt() in our program which is a mathematical function to find the square root of a number. This function is present in "math.h" file.

Now try to compile the file **myroots.c** with the following command:
$gcc myroots.c

Do you get an error message which says, **"Undefined reference to sqrt"**? Since we have a math feature in our program, we need to compile the program slightly differently.
$ gcc myroots.c –lm

The option "**lm**" helps in linking the math library **math.h** to our program, **myroots.c**. After compilation, do a listing (ls command) to find that the "**a.out**" file has been created.

To execute:**$./a.out**

6. TYPE CASTING

Typecasting is a way to convert a variable from one data type to another data type. It can be explicit or implicit. Explicit type casting is performed by the programmer. Implicit type casting performed by the compiler whenever an expression is combination of different data type. Syntax of explicit type casting is
```
Variable=(data type) expression
```
For example if you want to compute the average of N students computer programming marks. Write the program in average.c
```
#include <stdio.h>
int main()
{
int total=65,n=7;
float average;
average = total/n;
printf(“average=%f”,average); // result would be 7.000000 because expression
                                            type is integer
average= (float)total/n; // this will convert marks into float than expression is evaluated
printf(“average=%f”,average); // result would be7.03……
return 0;
}
```

It should be noted here that the cast operator has precedence over division, so the value of average is first converted to type float and finally it gets divided by n yielding a float value.

Implicit conversion: This conversion is performed by the compiler, automatically whenever two different types of operand encounter in an expression. The lower type is automatically converted into higher type before the operation proceeds. The result is higher type. For example if the above program is written as follows

```
#include <stdio.h>
int main()
{
float total=65,average;
int n=7;
average = total/n;
printf(“average=%f”,average);
/* result would be 9.285714 because n is automatically converted into float
for expression evaluation purpose */
return 0;
}
```

This conversion first performs integer promotion to data type, if operands still have different types then they are converted to the type that appears highest in the following hierarchy:

int->unsigned int -> long-> unsigned long-> float->double->long double

The usual arithmetic conversions are not performed for the assignment operators, nor for the logical operators && and ||.
Type the following program in conversion.c and predicate the output.

```
int main()
{
   int  i = 17;
   char c = 'c'; /* ascii value is 99 */
   float sum;
   sum = i + c;
   printf("Value of sum : %f\n", sum );
  return 0;
}
```

Note: When the above code is compiled and executed, it produces the following result:

Value of sum : 116.000000

Because first c gets converted into integer, due to integer promotion and result is store into sum that is float type. Type the following program into conversion1.c and predicate the output.

```
#include<stdio.h>
int main()
{ char ch;
int i;
i=321;
ch=i;
printf("\n ch  value  is %d \n",ch); // what will be output ??
//return 0;
}
```

Note: The output is 65 because assignment will drop the upper bits of integer 321. The lower eight bits of the number 321 represented the number 65.

Type the following program in Example2.c

```
#include<stdio.h>
int main()
{
int a=10;
unsigned long b=40000;
float f=1.23, g;
double d;
g = a + f; // a transforms to float
d = a + b;
/* a and b transform to unsigned long, adding is produced in unsigned long
domain and then the result type unsigned long is transformed to double */
printf(" g value is %f " g);
printf(" d value is  %lf", d);
return 0;
}
```

Exercise-3

Write a C Program to swap contents of two characters.

Example:

Initial Value	**Final Value**
char1= 'A'	char1= 'B'
char2= 'B'	char2= 'A'

Type the following program in the file, **myswap.c**

```
#include<stdio.h>
int main()
{
char char1,char2,temp;
printf("Enter values of char1 and char2"); /* char1='A' and char2='B' */
scanf("%c %c",&char1,&char2);
/* Swap the characters using Temp variable */
temp =  char1;
char1 = char2;
char2 = temp;
printf("%c%c",char1,char2)
return 0;
}
```

Compile and Execute:

```
$ gcc myswap.c
$./a.out
```

All the output that we have seen till now shows the output on the screen. How can we see the result of a C program on a file and not on the screen?

Hint: Use the output redirection '>' operator.

Exercise – 4

Write a C program to calculate the total distance travelled by a vehicle in 't' seconds, given by:

d = ut + 1/2 (at^2)

Where,

d = distance travelled, u = initial velocity in m/sec^2, a = acceleration in m/sec^2

Use scanf() to read the values of 'u', 'a' and 't' from the user

Exercise – 5

Write a C program to evaluate the following expression:
answer = $(a+b)^N$

Hint: Use the pow() function present in math.h
Usage example:
x = pow(2,3)
Value of x would then be 8

Exercise – 6

Write a C program that takes two numbers from the user as an input and it calculates and displays the sum, multiplication, division and subtraction of these numbers.