Programming Fundamentals

Lab Manual-03

Installation Guide
https://youtu.be/EgwndDGnm5w

Introduction to C Programming

a. Constants and Variables.
b. Data Types and its declaration in C.
c. Format specifier.
d. Format specifier.
e. Operators.
f. Expressions and Equations.
g. Keywords/ Reserved Words.
h. Programming Exercises.

1. Constants and Variables

Constants: A specific alphabetical and/or numeric value that is never changed.
For Ex. **PI** - 3.14159

Variables: The value that can be changed.
For Ex. ShoeCost = 56.00 and ShoeCost = 35.00

2. Data Types

1. **int** - integer: a whole number.
   This data type is used to define an integer number (-…. -3, -2,-1,0,1,2,3….). A single integer occupies 2 bytes.
   For example:
   int a;
   declares that you want to create an **int** variable called **a**.
   To assign a value to our integer variable we would use the following C statement: **a=10;**

2. **float** - floating point value: i.e. a number with a fractional part.
   A float, or floating point, number has about seven digits of precision and a range of about 1.E-36 to 1.E+36. A float takes four bytes to store.

3. **double** - a double-precision floating point value.
   A double, or double precision, number has about 13 digits of precision and a range of about 1.E-303 to 1.E+303. A double takes eight bytes to store.

   Note: Single precision and Double precision basically differs in the number of digits represented after the decimal point. Double precision number will represent more digits after the decimal point than a single precision number. **Example:** Single precision – 32.75 and double precision – 32.7543

4. **char** - a single character.

   Used to define characters. A single character occupy 1 byte.
   To assign, or store, a character value in a **char** data type is easy - a character variable is just a symbol enclosed by single quotes.
   char a;
   char a = '10';

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295
float	4 bytes	1.2E-38 to 3.4E+38
double	8 bytes	2.3E-308 to 1.7E+308
Long double	10 bytes	3.4E-4932 to 1.1E+4932

Variable Naming Convention

Specific to a Company. Helps programmers to be a part of an environment and follow and specified conventions

Rules for Variable Naming Conventions

1. Name a variable according to what it represents. Create as short name as possible but one that clearly represents the variable.
2. Do not use spaces in a variable name.
3. Start a variable with a letter.
4. Do not use dash (-) or any symbol that is used as a mathematical operator.
5. Use the same variable name to represent a specific data.
6. Be consistent when using upper and lower-case characters.
7. Use the naming convention specified by the company where you work.

Data Item	Incorrect Variable Name	Problem	Corrected Variable Name
Hours worked	*Hours Worked*	Space between words	*HoursWorked*
Name of client	*CN*	Does not define data item	*ClientName*
Rate of pay	*Pay-Rate*	Uses a mathematical operator	*PayRate*
Quantity per customer	*Quantity/customer*	Uses a mathematical operator	*QuantityPerCustomer*
6% sales tax	*6%_sales_tax*	Starts with a number	*SixPercentSalesTax or SalesTax*
Client address	*Client_address_for_client_of_XYZ_corporation_in_California*	Too long	*ClientAddress*
Variable name Introduced as *Hours*	*Hrs*	Inconsistent name	*Hours*
Variable name Introduced as *Hours*	*Hours_worked*	Inconsistent name	*Hours*

3. Format Specifier

The format specifier is used during input and output. It is a way to tell the compiler what type of data is in a variable during taking input using scanf() or printing using printf().

List of format specifiers in C

Format Specifier	Description
%d	Integer Format Specifier
%f	Float Format Specifier
%c	Character Format Specifier
%s	String Format Specifier
%u	Unsigned Integer Format Specifier
%ld	Long Int Format Specifier

4. Escape Sequences

An escape sequence in C language is a sequence of characters that does not represent itself when used inside string literal or character. It is composed of two or more characters starting with backslash \. For example: \n represents new line.

Escape Sequence	Meaning	Elucidation
\n	New line	Used to shift the cursor control to the new line.
\t	Horizontal tab	Used to shift the cursor to a couple of spaces to the right in the same line.
\a	Audible bell	A beep is generated indicating the execution of the program to alert the user.
\r	Carriage Return	Used to position the cursor to the beginning of the current line.
\\	Backslash	Used to display the backslash character.

5. Operators

Operators are the data connectors within expressions and equations. They tell the computer how to process the data. They also tell the computer what type of processing (mathematical, logical, or whatever) needs to be done. The types of operators used in calculations and problem solving include mathematical, relational, and logical operators.

Operands ⇒ data that the operator connects and processes.
Resultant ⇒ result when the operation is completed.

1	Arithmetic Operators	5	Logical Operators
2	Increment and Decrement Operators	6	Conditional Operators
3	Assignment Operators	7	Bitwise Operators
4	Relational Operators	8	Special Operators

5.1 Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).

Operator	Meaning of Operator	Operator	Meaning of Operator
+	addition or unary plus	/	division
-	subtraction or unary minus	%	remainder after division (modulo division)
*	multiplication		

Example #1: Arithmetic Operators

```
#include <stdio.h>
int main()
{
   int a = 9,b = 4, c;
   c = a+b;
   printf("a+b = %d \n",c);
   c = a-b;
   printf("a-b = %d \n",c);
   c = a*b;
   printf("a*b = %d \n",c);
   c=a/b;
   printf("a/b = %d \n",c);
   c=a%b;
   printf("Remainder when a divided by b = %d \n",c);
   return 0;
}
```

Output: `a+b = 13, a-b = 5, a*b = 36, a/b = 2`

5.2 Increment and decrement operators

C Program to demonstrate the working of increment and decrement operators.

```
#include <stdio.h>
int main()
{
   int a = 10, b = 100;
   float c = 10.5, d = 100.5;
   printf("++a = %d \n", ++a);
   printf("--b = %d \n", --b);
   printf("++c = %f \n", ++c);
   printf("--d = %f \n", --d);
   return 0;
}
```

Output

```
++a = 11
--b = 99
++c = 11.500000
-- d = 99.500000
```

Here, the operators ++ and -- are used as prefix. These two operators can also be used as postfix like a++ and a--. Visit this page to learn more on how increment and decrement operators work when used as postfix.

5.3 Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as	Operator	Example	Same as
=	a = b	a = b	*=	a *= b	a = a*b
+=	a += b	a = a+b	/=	a /= b	a = a/b
-=	a -= b	a = a-b	%=	a %= b	a = a%b

Example #3: Assignment Operators

```
#include <stdio.h>
int main()
{
   int a = 5, c;
   c = a;
   printf("c = %d \n", c);
   c += a; // c = c+a
   printf("c = %d \n", c);
   c -= a; // c = c-a
   printf("c = %d \n", c);
   c *= a; // c = c*a
   printf("c = %d \n", c);
   c /= a; // c = c/a
   printf("c = %d \n", c);
   c %= a; // c = c%a
   printf("c = %d \n", c);
   return 0;
}
```

Output

```
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

5.3 Relational Operators

Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Operator	Meaning of Operator
==	Equal to	!=	Not equal to
>	Greater than	>=	Greater than or equal to
<	Less than	<=	Less than or equal to

```
#include <stdio.h>

int main()
{
  int a = 9;
  int b = 4;

  printf(" a >  b: %d \n", a > b);
  printf("a >= b: %d \n", a >= b);
  printf("a <= b: %d \n", a <= b);
  printf("a <  b: %d \n", a < b);
  printf("a == b: %d \n", a == b);
  printf("a != b: %d \n", a != b);

}
```

5.4 Logical Operators

Logical operators are commonly used in decision making in C programming.

Operator	Meaning of Operator	Example
&&	Logial AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c == 5) && (d > 5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c == 5) || (d > 5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression! (c == 5) equals to 0.

```c
#include <stdio.h>

int main()
{
    int num =10;

    //printing result with AND (&&) operator
    printf("%d\n",(num==10 && num>=5));
    printf("%d\n",(num>=5  && num<=50));
    printf("%d\n",(num!=10 && num>=5));
    printf("%d\n",(num>=20 && num<=50));

    return 0;
}
```

Output

```
1
1
0
0
```

6. Keywords/ Reserved Words

There are certain words reserved for doing specific task, these words are known as reserved word or keywords. These words are predefined and always written in lower case or small letter. These keywords cann't be used as a variable name as it assigned with fixed meaning. Some examples are **int, short, signed, unsigned, default, volatile, float, long, double, break, continue, typedef, static, do, for, union, return, while, do, extern, register, enum, case, goto, struct, char, auto, const** etc.

7. Expressions and Equations

Expressions	Equations
$A + B$ *A* and *B* are numeric. The resultant is numeric and is not stored.	$C = A + B$ *C*, *A*, and *B* are numeric. The resultant is stored in *C*.
$A < B$ *A* and *B* are numeric, character, or string. The resultant is logical and is not stored.	$C = A < B$ *A* and *B* are numeric, character, or string. The resultant is stored in *C*; *C* is logical.
A OR *B* *A* and *B* are logical. The resultant is logical and is not stored.	*C* = *A OR B* *C*, *A*, and *B* are logical. The resultant is stored in *C*.

8. Exercises:

1) Fill the table with Variables name and data types:

S.No	Data Item	Variable Name	Data Type
1	Inventory number		
2	Quantity		
3	Price		
4	Reorder quantity		
5	Obsolete item (yes/no)		

2) Print following shape using simple printf() statements

(1)

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

(2)

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

(3)

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

(4)

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

(5)

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

(6)

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

3) What is wrong with these variable names? Can you correct them?

a. City Name referencing the name of a city.
b. Client-name referencing a client name.
c. City/State referencing a city and state.
d. LN referencing a last name.
e. Street address
f. Q for a quantity of books
g. Street_Address_for_Joe's_Hardware_Supply_Incorporated_Client.

4)
Program Name: Basic arithmetic operations
Purpose: To apply different arithmetic operators
Problem Statement: Write a C language program to apply all arithmetic operations on these two numbers n1=40 and n2=20 and display the results.
Run your program multiple times to verify that the output is correct all the times.
Sample Output:

```
Addition of a, b is : 60
Subtraction of a, b is : 20
Multiplication of a, b is : 800
Division of a, b is : 2
Modulus of a, b is : 0
Press any key to continue . . .
```

5) Assume you have 3 integer variables in your program with names a, b and c. Write down a program which performs following tasks.

I. Declare and initialize each variable in a separate statement.
II. Declare each variable in separate statement, then initialize each variable separate statement.
III. Declare all variables in one statement. Then, initialize each variable in separate statement.
IV. Declare and initialize all variables in a single statement.
V. Only declare variables a, b and declare/initialize variable c in a single statement.

6) Perform each of the following operations in your program. Does your program run? If yes, then what is output, and if no, then what is the error generated?

I. Try to print the value of a variable, which is not declared in your program.
II. Try to print the value of a variable, which is declared but not initialized in your program.
III. Try to assign the value of a variable beyond its allowable range and then print its value.
IV. Try to assign an integer variable the value of a float and then print its value.

7) Declare and initialize two variables a, b and calculate the sum and store it in a variable c. Declare and initialize two variables a, b and calculate the sum without storing it in a variable.