3.1. Character Sets, Keywords & Data Types

Character Sets

The set of characters that used to form words, numbers, and expression in C is called character set. The combination of these characters form words, numbers and expressions in C. The characters in C are grouped into the following four categories:

- **Letters or Alphabets:** Upper Case / Lower Cases (A-Z, a-z)
- **Digits:** 0 to 9
- **Special Characters:** (,,;?,”’,\,/-,>,^&,%,*,/=,+!,{()}>,<,[~,! etc)
- **White Space:** (Blank space, Horizontal tab, Carriage return, New line, Form feed)

Keywords

Keywords are predefined words for C program. All keywords have fixed meaning and these meanings cannot be changed. They serve as basic building blocks for program statements. They are as: double, int, void, struct, typedef, char, default, do, goto, if else, return, for, while, switch etc. These words are used for pre-defined purpose and cannot be used as identifiers to declare variable name. Thus the keywords are also called reserved words.

Data Types

There are various data types for example 10 and 100.4 are data of different items. The data 10 is integer number where as 100.4 is fractional number. There are other varieties of data type supported by C, each may be represented differently within computers memory. The varieties of data type of C supports are:

- Primary data type
- User-defined data type and
- Derived data types

The primary data types and derived are discussed in this section. The user defined data types such as arrays, functions, structures and pointers are discussed in next chapters. There are five primary data types they are:

- Integer (int)
- Floating point type (float)
- Double-precision floating type (double)
- Character Type (char) and
- Void Type (void)

Integer Type

The integers are whole numbers, non fractional numbers. Generally integer requires 16 bytes (2 bit) of storage area. C has further three types of integer they are int, short, and long in all three data types there are two sign and unsigned forms. The appropriate data types are used according to our requirements.

Signed Integer(signed int)	**Unsigned integer(unsigned int)**
It represents both positive and negative integers.	It represents only positive integers
The data type qualifiers is signed int or int Variable are defined as signed int a, b;	The data qualifier is Unsigned int or unsigned. Variables are defined as unsigned int a,b;
By default all int are signed	Unsigned int have to declared explicitly
It reserves 16 bits(2 bytes) in memory	It reserves 16 bits(2 bytes)in memory
Int represents values of range -32768 to +32767 (**2^{-15} to 2^{15} -1)**	It represents integer values of range 0 to -2 ^{16} -1 (0 to 65535)
Its conversion character is %d	Its conversion character is %u

Signed short Integer(signed short int)	**Unsigned Short integer(unsigned short int)**
It represents both positive and negative integers.	It represents only positive integers
The data type qualifiers is signed int or short int Variable are defined as signed short int a, b;	The data qualifier is unsigned short int or unsigned. Variables are defined as unsigned short int a,b;
By default all short int are signed.	Unsigned short int have to declared explicitly
It reserves 16 bits(2 bytes) in memory	It reserves 16 bits(2 bytes)in memory
Int represents values of range -0 to 255(8 bits)	It represents integer values of range (-127 to 128) bits
Its conversion character is %d or %i	Its conversion character is %u

Signed long Integer(signed long int)	**Unsigned Long integer(unsigned long int)**
It represents both positive and negative integers.	It represents only positive integers
The data type qualifiers is signed long int or long int . Variable are defined as signed long int a, b;	The data qualifier is unsigned long int or unsigned. Variables are defined as unsigned long int a,b;
By default all short int are signed.	Unsigned long int have to declared explicitly
It reserves 32 bits(4 bytes) in memory	It reserves 32bits(4 bytes)in memory
Int represents values of range -2147483648 to +2147483647 (-2 ^{31} to +2 ^{31} -1)	It represents integer values of range 0 to +2 ^{32} -1 (0 to 4294967295)
Its conversion character is %ld	Its conversion character is %lu

Floating Point Types

Floating point types are factional numbers. They are defined in C with keyword float. Floating numbers reserve 32 bytes (4) bytes of storage with 6 digits of precision.

Float

- It reserves 4 bytes in memory
- It represents fractional numbers of range 3.4* 10-38 to -3.4 1+38
- The data type qualifier is float variable declaration as float a;
- Its conversion character is % f.

Double Precision Floating Point Type

When accuracy provided by a float number is not sufficient, the type double can be used to define the number. A double data type number used 64 bits (8 bytes) giving a precision of 14 digits. There are double expression numbers. To extent the precision further long double can be used which use 80 bits (10bytes) giving 18 digits of precision.

Character Type

A single character can be defined as a character type data. Characters are stored in 8 bits (1 byte). The qualifier is char. The qualifier signed and unsigned may be used with char. The unsigned char has values between 0 to 255. The sighed char has values form -128 to 127. The conversion character for this type is % c.

Void Data Type

The void type has no return values. This is usually used to specify a type of function when it does not return any value to the calling function.

User Defined Data Types

C supports type definition which allows defining an identifier that would represent and existing data type. The typedef statement is used to give new name to an existing data type. It allows user to define new data types that are equal to existing data types. It takes the general form

- ***typedef int integer;***
  Here integer symbolizes int data type. Now we can declare int variable a as integer a instead of int a.
- ***typedef float decimal;***
  Here float f is equivalent to decimal f. Thus typedef statement is used to alias existing data types as convenient.

3.2. Constants and Variables

Constant

Constant is quantity that does not change during the execution of program. C supports constant can be divided into different category. They are:

Integer Constants

Integer constant refers to a sequence of digits with not decimal point either positive or negative. If no sign precedes an integer constant, it is assumed to be positive. No commas or blank spaces are allowed within the integer constant. Example 345,456,-8765.
There are three type of integer namely decimal, octal and hexadecimal.
Decimal Integer Constant: The decimal integer constant are the set of digits 0 to 9 proceeded by an optional – or + signs. Here decimal does not imply fractional numbers. It represent numbers having base of 10.
Octal Integer Constants: An octal integer constant consists of any combination of digits for the set form 0 to 7, with leading 0. Valid example are 074, 0676.
Hexadecimal Integer Constants: A hexadecimal integer constant consists of any combination of digits form the set of 0 to 9 and ABCDEF with leading) 0X examples are 0x345, 0x123, 0xaA.

Real Constants

Real constants are often called floating point constants. They can be written into two forms- fractional form and exponential form. Real constants are two types.
Fractional Constant: Fractional form of constants must have at least one digit and a decimal point. It can either be positive or negative but default sing is positive. Commas or blank space are not allowed within a real constant. For example: 23.56, 45.9,-67.5003 are in fractional form.
Exponential Constant: In exponential representation, the real constant is represented in two parts mantissa and exponents. The digit before is called mantissa and after is called exponent. The mantissa part may be positive or negative sign but default is positive. Similarly, the exponents must have at least one digit which can either positive or negative for eg. +3.2e5, 9.1e6.

Character Constant

A character constant is single character alphabet digit or special symbol enclosed within single ‘ ‘ marks. For example ‘A’ is a valid character. The maximum length of a character constant can be only one character. It must enclose by single code for example: ‘a’,’g’.

String Constant

A string constant is a sequence of characters enclosed in double quotes. It may contain letters numbers or special characters or blank space. However it does not have an equivalent ASCII value. For example “Hello good morning”. This must be enclosed by double coated(“”).

Variable

A variable is a symbolic name which is used to store data items. Variables are defined in a computer program to represent an item of data input by the user, any intermediate calculations results. Unlike constant the value of variable can change during the execution of a program. The same variable can store different values at a different portion of program.

Variable name may constant of letters, digits, or underscore characters. Since a variable is an identifier, the rules for naming variables are similar to those of identifiers.

Some valid examples data type are: firstname, num1,x2, email_id etc can be used
Some invalid examples are: data type, 1nepal, (total), first name, %male. etc.

Need for Variable Declaration

The declaration of variable associates the variable with a specific data type. This means each variable must be declared or defined as what type it is. By doing this the compiler will allocate a memory space in the computer as required by data type.

Thus variables are assigned with specific data type according to the values to be stored. If we have integer values to store, we need to define a variable of type “int”. It gives the name of variable and its types should have matched properly, when we declared it. It allocates appropriate memory space according to the data types declared in front of it.

Rules for Variable Declaration

- The variable should be stored with only letters or underscore.(itn i; int_p;)
- The variable name should not be any keyword.(int auto,int int, int do;)
- The variable name is case sensitive characters.(if you declared a variable x then while accessing it do not use X: these two are different representation)
- No two variables of the same name are allowed.(int i; int i are not allowed in same program)
- It must starts form letters/alphabets.(int a1, char p2 not but int 1a are not allowed)

Preprocessor Directives

Preprocessor directive is a collection of special statements that are executed in the beginning of compilation process. They are written in source program before the main program. Preprocessor directives follows special syntax rules that are different form normal C syntax. They all begins with the symbol # (Hash) and do not require semicolon at end.

```
#include<stdio.h>  /*used for standard input and output*/
#include<conio.h> /* used for console input and output*/
#define PI 3.1416  /*defining symbol*/
#define TRUE 1     /*used for defining TRUE as 1*/
```

#define FALSE 0 /*used for defining FALSE as 0*/

These statements are called preprocessor directives as they preprocessed source program before compilation of any source code in the program. The other codes in the program are compiled sequentially line –by line.

Escape Sequence

An escape sequence is non-printing characters used in C. It is character combination of backslash followed by latter or by combination of digits. Escape sequences are typically used to specify actions such as carriage returns and tab movements on terminals and printers. They are also used to provide literal represents of characters that have special meanings such as double quotation mark (“”).

```
#include<stdio.h>
#include<conio.h>
void main(){
printf (“Hello\n I am testing escape sequence”);
getch();    }
```

Generally, any text written within double quotes (“”) in printf () function will be displayed on the screen. There are some more escape sequences .

\a	Alert purpose by an audible alert or beep sound
\b	To delete one character to the left of current cursor position
\f	Form feet to feed a blank page when the user prints the document
\n	Line feed next line
\r	Carriage return move cursor to the next line during typing
\t	Horizontal tabs to move cursor to next tab stop position
\0	For null characters

Symbolic Constant

A symbolic constant is a name that is used in place of sequence of characters. The character may represent numeric, a character constant or a string constant. When a program is compiled, each occurrence of symbolic constant is replaced by its corresponding character sequences.

3.3. Operators

Operator

An operator is a symbol that operates some sort of calculation. Operators are used in programming to perform certain mathematical or logical manipulations. For example, in a simple expression 9+67 the symbol “+” is called an operator which operates on two items 9 and 67 are operands. The data items that operate through operator are called operands. Here 9 and 67 are operands. According to the number of operands required for an calculation, we can classify operators into unary, binary and ternary.
Unary Operators: The operator which requires only one operand is known as unary operators. For example: ++ increment – decrement, a++,--a,++a, Here only one variable is used.
Binary Operators: The operators which requires two operand are known as binary operators for example + - * / >, < , a+b, a/b etc are binary operator.
Ternary Operators: The operators that require three operands are known as ternary operators for example “?:c” is conditional operator.

C operators can be classified into a number of categories

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Increment and decrement operator
- Conditional operators
- Conditional operators
- Bit wise operators
- Special operators

Arithmetic Operators

An arithmetic operator performs arithmetic operations. There are five arithmetic operators in C.

Operator	Meaning	Example	Output int A=20, B=6, C=6;
+	Addition	A+B	26
-	Subtraction	A-B	14
*	Multiplication	A*B	120
/	Division	A/B	3
%	Modulo	A%C	2(gives remainder)

The +,-,*, / all work in the same way in usual mathematics. The modulo operator (%) calculates remainder after division of one integer operand by another integer operand. So the remainder of 20 divided 6 will give remainder 2 but 8%2=0 gives zero as remainder.

Integer Arithmetic

When the operands in a single arithmetic expression such as a+b are integer, the expression is called an integer arithmetic expression. Integer arithmetic always yields an integer value. That is why the operation 20/6 in the above example yields 3. The factional positions of actual results (3.333333) are truncated.

Division Rules

int/int=int float/float=float
int/float=float float/int=float

During integer decision if both operands are of same sign, the results truncated towards zero.
4/7=0. However if one of the operands is negative then the direction of truncated is machine dependent. This is -4/7 may be 0 or 1 depending upon the machine. Similarly during modulo, the sign of the result is always the sign of the first operand. 20% 3=2, -20%3=-2, the modulo division operation cannot used on floating point data.

Relational Operator

Relational operators are used to compare two similar operands, and depending on their relation, take some actions. The relational operators compare their left hand side operand with the right hand side operand by using lesser than, greater than, lesser than or equal to relations. The value of relational expression is either 1 or 0 (if conditions is false 0 will produce where as if condition is true result become 1).

Operator	Meaning	Example	Output(int a=20,b=6)
<	Less Than	A<b	0
>	Greater than	a>b	1
<=	Less than or equal	A<=b	0
>=	Greater than or equal	a>=b	1
==	Equal to	A==b	0
!=	Not equal to	A!=b	1

Logical Operator

Logical operators are used to compare or evaluate logical and relational expressions. The operands of these operators must produce either 1 or 0. The whole result produce by these operators is also either true or false. There are three logical operators used in C programming.
(&&) Logical AND (||) Logical OR (!)Logical NOT

A	B	A&&B	A||B	!A	!B
0	0	0	0	1	0
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	1

Assignment Operator

Assignment operators are used to assign the result of an expression to a variable. The common assignment operator is =. There are other various shorthand's assignments they are:

```
a+=b        // a=a+b
a-=b        // a=a-b
a/=b        // a=a/b
a%=b        // a=a%b
```

- If two operands in assignment expressions are different data type, then the value of expression on the right will automatically be converted to the type of identifier on left.
- A floating-point value may be truncated if assignment to an integer.
- A double precision value may be rounded if assignment to floating point value.
- An integer quantity may be altered if assignment to a shorter integer variable.

Increment and Decrement Operators

The increment operator is to increase the variable of an operand by 1, and the decrement operator is used to decrease the value of an operand by 1. They take only one operand, so called unary operator.
The increment/decrement operators are prefix or postfix in prefix notation, the value of variable is first incremented or decremented and then the value is assigned to the variable.
Similarly, the postfixes, first the previous value of variable is used and then the value is incremented or decremented.

Conditional Operator

The conditional operator that takes three operands so it is also called ternary operator.
The syntax of this operator is ***Expression1? Expression2: Expression 3***.
Here, expression 1 is evaluated first, if Expression 1 is true, the value of Expression 2 is assigned to the large variable. Else Expression 1 is false then value of Expression3 is assigned to the large variable.

Comma Operator

The comma operator can be used to link related expressions together. Comma linked lists of expressions are evaluated from left to right and the value of the rightmost expression is the value of the combined expression.

N2=(n1=50, n2=10,n1+n2)

This will first assign the value 50 to n1, then again 10 is assign to n2 variables. The finally the sum is calculated then assigned to N2 variable (n1+n2 ie 60). Since comma operator has lower precedence of all operators.

Sizeof Operator

The sizeof operator is used with an operand to return the number of bytes it occupies. It is a compile time operand. The operand may be a constant, value or data type qualifier as following. The syntax sizeof(argument). Here sizeof operator gives its argument size.

Operator Precedence

The precedence is used to determine how an expression is evaluated and which order it will start to evaluation among various operators used. There are distinct levels of precedence and an operator may belong to one of these levels. The operators at the higher level of precedence are evaluated first. The operators of same precedence are evaluated either form left to right or right to left depending to the level . This is known as associative property of an operator.

Precedence	Operator	Description	Associative
1 (Highest)	()[]	Function call	Left to right
2	+,-,+ +,--,!,~,*,&, sizeof, type	Unary,increment/decrement, address, pointer, type cast	Right to left
3	*,/,%	Multiplication/ division/modulo	l-R
4	+,-	Addition/division	L-R
5	<<,<=<>,>=	Relational	L-R
6	==,!=	Relational	L-R
7	&,|<&&,||?:	Logical	L-R
8	=,*=,/=,+=	Assignment	Right to Left
9(Lowest)	,	Comma	L-R

Type Conversion in Expression

C allows mixing of constants and variables of different types in single expression. In such situation, one type must be converted to another before evaluation can be done. This kind of conversion is called type conversion. The common type conversion are two types implicitly or explicitly.

Implicit Type Conversion

When there are constants and variables of different types in an expression, C automatically converts any intermediate values to the proper type. This conversion assures that the expressions can be evaluated without losing any digits inside. This automatically convert typed is known as implicit type conversion.

During evaluation if operands are of different types, the lower type is automatically converted into the higher type before the operation proceeds. The conversion takes place according to rule called the conversion hierarchy.

The final result of an expression is converted to the type of variable on the left of the assignment signs before assignment the value to it. However the following changes are introduced before the final assignment.

Float to int causes truncation of the fractional part
Double to float causes rounded of digits
Long int to int causes dropping of the excess higher order bits.

Explicit Type Conversion

Instance where the type can be converted forcefully by the user himself is called explicit type conversion or casting value. The general rule to cast is (type name) expression.

N1=(int)9.50	9.50 is converted to integer by truncation
N2=(int)16.9/(int)6.5	Evaluation is done 16/6, resulting 2
N3=(double)sum/n	Sum is calculate to double and division is done in floating mode
N4= (int) (a+b)	The results is converted then assign
N3=sin((double)x)	X is converted to double and the sin value is calculated and assign

3.4. Input and Output Operations/Management

Formatting scanf() and printf() functions

A program without any input or output has not meaning; generally a program reads input data form keyboard or other files. The program then processes input data and result is displayed on the screen or monitor. Let's consider program to create mark sheet of student then program must take marks of five subjects then after getting some information the percentage and division will be calculated. Finally the required results is shown in the screen.

C has number of input and output functions, when a program needs data. We take the data through input functions and results to output devices through output functions. As keyboard is standard input devices, the input functions used to read data form keyboard are called standard input function.

The standard input functions which are scanf(), getchar(), getche(), getch(), gets(). Similarly output functions are printf(), putchar(), putch(), puts (). The standard library <stdio.h>provides functions for input and output. The instruction include<stdio.h> tells the compiler to search <stdio.h>from the standard i/o directory and place its contents in the program. The content of header files become part of source code. The input and output factions are two types: Formatted functions and Unformatted functions.

Formatted Function

Formatted functions allow to give input/ read form the keyboard and output displayed on screen to can be formatted according to user requirements. The input function scanf() and printf() are fall under this category. While displaying certain data on screen we can specify the number of digits after point, number of spaces before the data, and position where output is to be displayed are handled through there functions.

Formatted Input/Scanf()

Formatted input refers to an input data that has been arranged in a particular format for example 50,30.4, name. This line contains three types of data and must be read according to its format. The first part should move into variable int, the second must be variable in float and the third into char. This is possible in C by using scanf() function.
The built in function scanf() can be used to enter input data to the computer form the standard input devices. The functions can be used to enter any conditional or numerical values, single characters or string. In other word, it is used for runtime assignments of variables.

Syntax: scanf("control string",args,arg2,args);

The address of location where the data is stored signifies in control string. It represents pointer that indicates the address of data items within computer memory (&). The control string consists of individual groups of data format with one group for each input data items. Each data format must begin with a percentage sign (%).

Formatted Output/Printf()

The formatted output refers to the output of data that has been arranged in particular format. The printf() is a built in function which is used to output data from the computer output devices on screen. The printf statements can be used to control the alignment and spacing.

Syntax: printf ("control string", args,args,args);

The control string consists of four type items

- Characters that will be printed on the screen
- Format specification that define output format for display of each items
- Escape sequences that define output format for display
- Any combination of characters, format specification and escape sequences

The control string has as form:

printf (" %[flag][field][.precision] conversion character");

Flags [Optional]: The flag affects the appearance of the output. They must be placed just after percentage sign. -+,0, o, x or blank space are symbol . – indicates the left justified. + indicates +ve or –ve numbers. A blank space will precede each positive signed numerical data. o it refers octal similarly x refers hexadecimal numbers.
Output of Integer Numbers: %wd is used to display in a desired width of integer values. Where w is the minimum field width for the output and d specifies that the value printed is an integer.
Output of Single Character: A single character can be designed in a desired position using format %wc.
Precision [Optional]: The operation of precision field depends on data type of conversion. It must start with period (.).
Output of Real Numbers: The precision can further be extended using field width of form w.pf where w is field width and p is precision. The conversion character is %f.
Output String: The format specifies for output string is form of %w. p s where w is width for display and p instruct that only the first p character of string are displayed, it count white space too.

Unformatted Functions

Unformatted function do not allow user to read or display data in desired format. These library functions are available in the header files they are: getchar(), putchar(), gets(), puts(), getche(), getch() are unformatted functions.

getchar() and putchar(): The getchar function reads character form standard input devices. It takes the following format char p=getchar();. Similarly putchar () function display a character to standard out devices. It takes flowing format putchar (character variable);

getch(), getche() and putch(): The function getche() and getch() reads single character of instant of time, it is type without waiting for enter key to be hit. The difference between them is that getch() reads the character typed without echoing it on the screen, while getche () reads the character and echoes display it onto the screen. getch() format is char variable=getch(); the function putch() prints a character onto the screen. It has the form putch (char variable name);