BCA

CORE - 3

PROGRAMMING WITH C AND C++

BHARATHIAR UNIVERSITY
SCHOOL OF DISTANCE EDUCATION
COIMBATORE

CORE - 3 PROGRAMMING WITH C AND C++

Subject Description : This subject deals various methods programming using the C and C++ languages.

Goal : To learn the C and C++ programming languages.

Objective : On successful completion the students should have programming ability on C & C++.

UNIT – I : Structure of a C program – C character set – Delimiters – Keywords – Identifiers – Constants – Variables – Rules for defining variables – Data types – Declaring and initializing variables – Type conversion. Operators and Expressions – Formatted and Unformatted I/O functions – Decision statements – Loops – for, while, do...while statements.

UNIT – II : Arrays – String and its standard functions. Pointers – Functions – Structure and Union: Features of structure, Declaration and initialization of structure, Structure within structure, Array of structure, Union.

UNIT-III: C++ Declarations. Control Structures: - Decision Making and Statements: If..else, jump, goto, break, continue, switch case statements - Do - Functions in C++ - Inline functions – Function Overloading. Classes and Objects: Declaring Objects – Defining Member Functions – Static Member variables and functions – array of objects –friend functions – Overloading member functions – Bit fields and classes – Constructor and destructor with static members.

UNIT-IV: Operator Overloading: Overloading unary, binary operators – Overloading Friend functions – type conversion – Inheritance: Types of Inheritance – Single, Multilevel, Multiple, Hierarchal, Hybrid, Multi path inheritance – Virtual base Classes – Abstract Classes. Pointers – Declaration – Pointer to Class , Object – this pointer – Pointers to derived classes and Base classes – Arrays – Characteristics – array of classes – Memory models – new and delete operators – dynamic object – Binding , Polymorphism and Virtual Functions.

UNIT-V : Files – File stream classes – file modes – Sequential Read / Write operations – Binary and ASCII Files – Random Access Operation – Templates – Exception Handling - String – Declaring and Initializing string objects – String Attributes – Miscellaneous functions .

TEXT BOOKS:

1. PROGRAMMING IN ANSI C, E. Balagurusamy, TMH, 1998

2. OBJECT-ORIENTED PROGRAMMING WITH C++, E. Balagurusamy, TMH, 1998

REFERENCE BOOKS:

1. **The Sprit of C – Mullish Cooper,** Jaico Publications.

2. **OBJECT-ORIENTED PROGRAMMING WITH C++,** D.Ravichandran, 2nd ed, TMH.

CONTENTS

UNIT - V

UNIT - I

Lesson - I
CHARACTER SET

1.0 Aim and Objectives

In this unit we have discussed about the structure of a C program its character set and the delimiters used in c language. After reading this lesson one should be able to

- know about the characters
- know about special characters allowed
- delimiters used in a C program
- structure of C program

1.1 Introduction

C is a high level language developed by Dennis Ritchie. It was implemented in 1972 at Bell Laboratories. This language came from "Basic Combined Programming Language" (BPCL) called "B" which was developed in 1960 at Cambridge University. This language was modified and named as "C".

Programs written in "C" are efficient and fast. They have good built-in-functions and operators so that they can be used to write complex programs ."C" is well suited for structured programming (programs are written in terms of functions, modules or block so that debugging ,testing and maintenance are easier)

1.2 Structure of C program

C is a free form language, giving more freedom to the programmer. C program statements are written in lowercase letters, and is very much case sensitive. It has nearly 32 keywords, which is combined with the formal C syntax. A key word shall not be used for any other purpose in a C program.

The documentation section consists of a set of comment lines giving the name of the program, the author and other details. The link section provides instruction to the compiler to link functions from the system library.

The definition section defines all symbolic constants. Some variables are used in more than one function those are called global variables and are declared in global declaration. It may be seen that this form of C program consists of many user – defined functions (f1 ()..... fn ()).

A form of a c program will be like this :

```
Documentation section
Link section
Definition section
Global declaration section
main() function section
{
Declaration part ;
Executable part;
}
Subprogram section
Function1()
{
.....
.....
}
Function2()
{
.....
.....
}
.
.
.
.
Functionn()
{
.....
....
}
}
```

This is a general form and all C programs consist of one or more user defined functions. Every C program must have one main() function, and it is the first function called when program execution begins. It may also be seen that opening and closing braces are used at appropriate places, either to distinguish functions to code blocks. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main () function is the logical end of the program. All commands, executable parts, and statements in a C program end with a semicolon.

1.3 C Character Set

The characters are used to form words, numbers and expressions. In C the characters are grouped as

a) Letters : consists of upper case letters A to Z and lowercase letters a to z. It is noted that upper and lower case letters are not interchangeable as C is strongly case sensitive.

b) Digits: consists of numbers from 0 to 9

c) Special characters are given as below:

, comma	& ampersand
; Semicolon	^ caret
: Colon	* asterick
? question mark	+ plus sign
‘ Single quotation	- minus sign
“ double quotation	< less than sign
! exclamation mark	> greater than sign
| vertical bar	( ) left & right parenthesis
/ slash	[ ] left & right brackets
\ backslash	{} left & right braces
~ tilde	# hash
_ underscore	$ dollar sign
% percent sign	

d) White spaces : consists of blank space, new line, form feed, carriage return and horizontal tab.

1.4 Sum up

In this lesson
History of c programming has been discussed.
The structure of a C program and its functions are given.
The character sets those which are used in a C program are described.

1.5 References

1. Programming in ANSI C, E.Balagurusamy, TMH, 1998

2. The sprit of C – Mullish Cooper, Jaico Publications

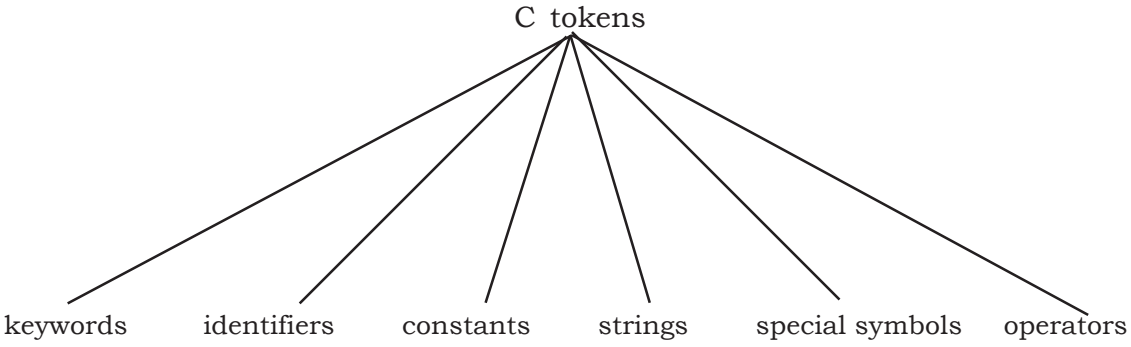
Lesson II
KEYWORDS, CONSTANTS AND VARIABLES

2.0 Aim & Objectives

In this lesson we have discussed about the keywords, identifiers, constants and variables of a C program. After reading this lesson one should be able to know about the keywords, identifiers, constants and variables, know about variables and rules for defining variables

2.1 Introduction

In a text individual words and punctuation marks are called tokens. In C program individual units are known as tokens. C has six types of tokens, they are



2.2 Keywords

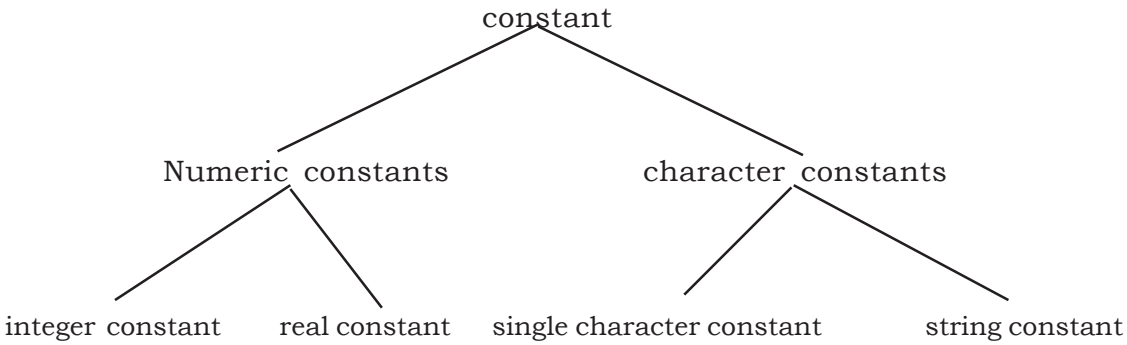
These have fixed meaning which cannot be changed. Keywords serve as basic building blocks for program statements. Keywords are written in lower case. List of keywords are : auto , break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, size of, static, struct, switch, typedef, union, unsigned, void, volatile, while etc....

2.3 Identifiers :

These refer to the names of variables, functions and arrays. These are user – defined names and consists of a sequence of letters and digits. The first character is a letter. Underscore is allowed.

2.4 Constants :

Constants are those that have fixed values and that do not change during the execution of a program. It supports several types of constants. They are as follows



Integer constants

Integer constants refer to a sequence of digits without a decimal point. There are 3 types of integers they are decimal, octal and hexadecimal constants.

Decimal constants: consists of a set of digits 0 to 9 with a optional + or – sign.

Eg : 123, -123 , + 123

Octal constants: consists of digits from 0 to 7

Eg : 123, 01233

Hexadecimal constants: consist of value from 0 to 15. the letters A through F is used to represent 10 through 15. the sequence of digits are preceded by 0x or 0X.

Eg : 0x2, 0X9F

Real constants:

Real constants are represented by numbers containing fractional part. They are also called floating point constants.

Eg : 1.25, + 25.63, - 4675.05

There should be digits after the decimal point. A real number may also be expressed in exponential notation

Mantissa e exponent

The mantissa is either a real number expressed in decimal form. The exponent is an integer number with a optional plus or minus

Eg: 1.23E2, 1.23e2, -1.2e-1, 1.5e+5

String character constants

A string character constant contains a sequence of characters enclosed in double quotes. The characters may be letters, numbers, special characters and blank space.

Eg: “hello”, “5+3”

Back slash character constant

The backslash characters are used in output functions. These are also called as escape sequences.

Constants	**meaning**
‘\a’	audible alert (bell)
‘\b	back space
‘\f’	form feed
‘\n’	new line
‘\r’	carriage return
‘\t’	horizontal tab
‘\v’	vertical tab
‘\”	single quote
‘\”’	double quote
‘\?’	question mark
‘\\’	back slash
‘\0’	null

2.5 Variable:

It is a data name which is used to store a data value. A variable has different values at different times during execution. A variable name is user defined. It may consist of letters, digits and an underscore character.

Eg: average, height, count, totat_1, abc_def

2.6 Rules:

1) Variables must begin with a letter
2) The length of variables can be a maximum of 31 but it is dependent on the compiler
3) Upper case and lower case are significant eg. TOTAL is different from total
4) The variable should not be a keyword
5) White space is not allowed.

2.7 Sum up

From this lesson

- One can understand keywords, identifiers, constants and variables
- What is a variable how to initialize and usage of variables is said
- Rules for defining variables have been given
- One will be able to define a variable, constant and identifiers]

2.8 References

1. Programming in ANSI C, E.Balagurusamy, TMH, 1998
2. The sprit of C – Mullish Cooper, Jaico Publications

Lesson III
DATA TYPES

3.0 Aim And Objectives

In this lesson different types of data types their usage, how to declare a variable, how to initialize a variable and to covert one data type to another has been discussed.

After reading this lesson one should be able to :

- know about primary, user defined and derived data types
- know about declaration and initialization of variables
- type conversions

3.1 Introduction

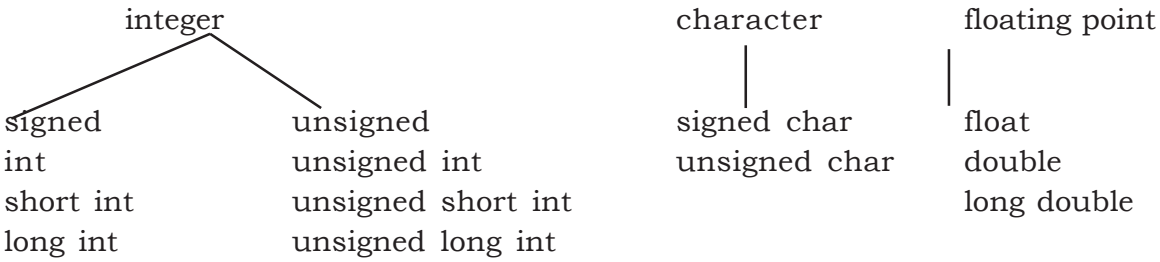
C language is rich in its data types. Storage representations and machine instructions to handle constants differ from machine to machine. The variety of data types available allows the programmer to select the type appropriate to the needs of the application as well as the machine.

3.2 Data Types

All C compilers support five fundamental data types, namely integer (int), character (char), floating point (float), double precision floating point (double) and void. Many of them also offer extended data types such as long int and long double.

3.3 Primary Data Types

The four fundamental data types are



Integer type

Integers are whole numbers with a range of values supported by a particular machine. Size and range of data types on a 16 bit machine

Type	size	range
char or signed char	8	-128 to 127
unsigned char	8	0 to 255
int or signed int	16	-32,768 to 32,767
unsigned int	16	0 to 65535
short int	8	-128 to 127
unsigned short int	8	0 to 255
long int	32	-2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295
float	32	3,4E-38 to 3.4E+38
double	64	1.7E-308 to 1.7E+308
long double	80	3.4E-4932 to 1.1E+4932

Floating point types

These are stored in 32 bits with a 6 digits of precision. When the accuracy provided by a float number is not sufficient the type double can be used which has 64 bits giving a precision of 14 digits.

Character types

Characters are usually stored in 8 bits. These can be defined by a character type data.

Void types

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

3.4 Declaration of variables

First the variable names are decided then the variables should be declared to the compiler. Declaration tells the compiler what the variable name is. It specifies what type of data the variable will hold.

Primary type declaration

A variable can be used to store a value of any data type.

Syntax: **Data type v1,v2,.....vn;**

v1, v2.....vn are the names of the variables. The variables are separated by commas. The statement ends with a semicolon.

Eg: int number;
float total;
char a;

3.5 User defined type declaration

This function is that the user can define an identifier that would represent an existing data type. This user-defined data type identifier can later be used to declare variables.

Syntax : **typedef type identifier;**

Eg: typedef int units;
typedef float marks;
units batch1, batch2;

The batch1, batch 2 are of data types int. the main advantage of typedef is that we can create meaningful data type names for increasing the readability of the program.

3.6 Enumerated data type

This data type is used to declare variables that have one of the values enclosed within the braces

Syntax : **enum identifier v1,v2.....vn;**

Eg : enum day { Monday, Tuesday,.....Sunday};
enum day week_st, week_end;
week_st = Monday;
week_end= Friday;

Assigning values to variable

Values can be assigned to variable using the assignment operator =

Syntax: **variable_name = constant;**

Data type variable name = constant;

Eg : total = 100;
initial-value = 0;
choice = 'y';
int x = 0;
char choice = 'y';
float pie = 3.14;

3.7 Declaring a variable as constant

The variable may remain constant during the execution of a program for this type of variables we declare as

Eg: const float pie = 3.14;

Declaring a variable as volatile

It can be used to tell explicitly to the compiler that the variable's value may be changed at any time by some external sources

Eg : volatile int date;

Defining symbolic constants

These constant may appear repeatedly in a number of places in a program. This is used for better modifying and for better understandability.

Syntax : `#define symbolic-name value of constant`

Eg: #define MAX 100
#define PI 3.1459

Rules

1. Symbolic names have the same form as variable names, but they are written in capitals
2. No blank spaces are left between # and define
3. One blank spaces is left between #define and the symbolic name and constant
4. #define statement does not end with a semicolon.
5. Symbolic names do not have a data type
6. It may be anywhere in the program before it is referenced in the program.

3.8 Type conversion in expressions

Automatic type conversion

When the expressions have different types of operands the lower type is automatically converted to the higher type before the operation proceeds.

E.g. :
```
int i,x;
float f;
double d;
long int l;
x = l/i + i * f-d;
```

The final answer is double and is stored in x which is int.

All short and char are automatically converted to int.

(i) If one of the operands is long double, the other will be converted to long double and the result will be long double.

(ii) If one is double, the other is converted to double and the result is stored in double.

The same applies to float and unsigned long int.

(iii) If one of the operand is long int and the other is unsigned int then:

(a) If unsigned can be converted to long int the result will be long int.

(b) Else both the operands will be unsigned long int.

(iv) If one of the operand is long int and the other will be converted to long int and the result will be long int.this applies to unsigned int also.

However: - 1) float to int causes truncation of the fractional part

2) Double to float causes rounding of digits

3) Long int to int causes dropping of the excess higher order bits

Casting a value

In certain instances we force a type conversion in a way that is different from the automatic coversion

Syntax : (type-name) expression

Type name is one of the c data type and expression may be any constant, variable or an expression.

Eg: x = (int) 7.5

3.9 Sum up

After reading this lesson one should be able to:

- Declare variable with the data types like primary, user defined and derived data types.
- Use variable of different data types with its data types and its initialization.
- Familiarly use the data types by knowing its type conversions.

References

1. Programming in ANSI C, E.Balagurusamy, TMH, 1998
2. The Sprit of C – Mullish Cooper, Jaico Publications

Lesson IV

OPERATORS AND EXPRESSIONS

4.0 Aim and objectives

In this lesson we have discussed about the operators, expressions, formatted i/o functions and unformatted i/o functions. After reading this lesson one should be able to

- Know about all the arithmetic, relational, logical, assignment, increment, decrement, Conditional, bitwise and special operators.
- Know about expressions
- Know about formatted and unformatted i/o functions

4.1 Introduction

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. The operators are classified into a number of categories. They are

1) Arithmetic operators
2) Relational operators
3) Logical operators
4) Assignment operators
5) Increment & decrement operators
6) Conditional operators
7) Bitwise operators
8) Special operators

4.2 Arithmetic operators

The basic arithmetic operators are

Operators	meaning
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	modulo division

The unary minus operator multiplies its single operand by -1.

E.g. : a-b, a*b, -a*b

Here a and b are called **operands.**

Integer arithmetic

The operands in a single arithmetic expression are integers then the expression is called integer expression and the operation is called integer arithmetic

Eg: a-b=10 where a= 14 & b= 4

Real arithmetic

An arithmetic operation involving only real operands is called real arithmetic. A real operand has the values either in decimal or exponential notation.

Eg: x= 2.5 y=3.5

z=x*y;

Mixed mode arithmetic

When one of the operands is real and the other is in integer then such an expression is called a mixed – mode expression. If either one of the operand is real type then the result is real.

15/10.0 = 1.5 or 15/10 = 1

4.3 Relational operators

When two values are compared and then decision is made it is called relational operation. The relational operators are as follows

Operator	meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than equal to
==	is equal to
!=	is not equal to

Syntax : **ae-1 relational operator ae-2**

ae-1 and ae-2 are arithmetic expressions

eg: a>b or (a*b) > (c>d)

Relational expression are used in decision statement like if and while. Arithmetic operators have a higher priority over relational operators.

4.4 Logical operators

These operators are used to test more then one condition and make decisions. The logical operators are as follows

Operator	meaning
&&	AND
||	OR
!	NOT

An expression which combines two or more relational expressions is termed as logical expression or a compound relational expression

Op-1	op-2	Value of the expression op-1&&op-2	op-1 || op-2
Non-zero	non-zero	1	1
Non-zero	0	0	1
0	non-zero	0	1
0	0	0	0

Eg: (age >55 && salary <1000)

4.5 Assignment operators

It is used to assign the result of an expression to a variable. The operator is "=".

Syntax **V op = exp;**

V is a variable, exp is an expression, op is a binary arithmetic operator. It is also called short hand operator

Eg : a = b +1;

x+=3; is equal to x = x+3;

short hand assignment operator

a= a+1	a+ =1
a= a-1	a-=1
a=a*n+1	a*= n+1
a=a/n+1	a/=n+1
a=a%b	a%=b

Advantages

a) the statement is more efficient
b) the statement is easier to read
c) the operand which appears on the left hand side need not be repeated.

4.6 Increment & decrement operators

There are two very useful operators they are increment and decrement operators++ and --

The ++ operator adds 1 and -- subtracts 1.

Eg: m=5; y=++m;

The ++m means it is called prefix operator and m++ is called post fix operator. In prefix operator the value is added up and then the result will be assigned to the variable. In post fix operator the value is first assigned and then it is incremented.

The same way the decrement is operated. -- m is the predecrement operator and m -- is called the post decrement operator

4.7 Conditional operator

"?" is called the ternary operator. ? : is called conditional operator. Syntax and explanation is as follows.

Syntax **exp1 ? exp2 : exp3;**

Exp1, exp2 and exp3 are expressions.

The? : Works as follows. 'exp1' is evaluated first, if it is true 'exp2' is evaluated if it is false 'exp3' is executed.

E.g.: a=10, b=15;

x= (a>b)? a:b;

Here the value of x is assigned the value of b.

4.8 Bitwise operators

These operators are used for manipulation of data at bit level. These are used for testing the bits or shifting them right or left. These are not applied to float or double.

Operator	meaning
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left
>>	shift right
~	one's complement

4.9 Special operators

Special operators like comma operator, size of operator, pointer operator(& and *) and member selection operators(. and →)

Comma operator

It is used to link the related expressions together these are evaluated left to right.

Eg: value = (x= 10, y=5, x+y);

Here the value of 10 is assigned to x, 5 to y and the final value 15 is stored in value.

Size of operator

It is a compile time operator and is used with an operand, it returns the number of bytes the operand occupies. The operand may be a variable, a constant or a data type.

Eg : m = sizeof (sum);
n = sizeof(long int);
k = sizeof(2352);

This operator is normally used to determine the lengths of arrays and structures. It is used to allocate memory space dynamically to variables during execution of a program.

4.10 Arithmetic expressions

It is a combination of variables, constants and operators.

Evaluation of expressions

Syntax : **variable = expressions;**

E.g.: x = a * b - c;

The a * b - c is the expression. The values for these variables are already defined and the result is stored in x.

Precedence of arithmetic operators

An arithmetic expression without parenthesis will be evaluated from left to right using rules of precedence of operators. There are two priority levels they are

High priority	* / %
Low priority	+ -

While evaluating an expression first it is evaluated from left to right then during the first pass the high priority operators are applied and then the low priority operators are applied.

Eg: x = a – b / 3 + c * 2 - 1
a=9,b=12 & c=3.

The expression becomes x= 9 – 12 / 3 + 3 * 2 - 1
First pass: step 1 x = 9 - 4 + 3 * 2 - 1
step 2 x = 9 - 4 + 6 - 1
Second pass: step 3 x = 5 + 6 - 1
step 4 x = 11 - 1
step 5 x = 10

Operator precedence and associativity

The precedence is used to determine how an expression involving more than one operator is evaluated. The operators with high priority or high level are evaluated first. The operators of the same priority or precedence are evaluated from left to right or from left to right depending on the level. This is known as associativity property of an operator.

Eg : if (x==10 + 15 && y<10)

Here + has higher priority so it is evaluated first. Second both == or < (i.e.) the relational operations are checked next then as then above two operations have the same precedence they are evaluated from left to right.

Mathematical functions

Mathematical functions like cos, sqrt, log etc are frequently used

Function	**meaning**
Trigonometric	
acos(x)	arc cosine of x
asin(x)	arc sin of x
atan(x)	arc tangent of x
atan2(x,y)	arc tangent of x/y
cos(x)	cosine of x
sin(x)	sine of x
tan(x)	tangent of x
Hyperbolic	
cosh(x)	hyperbolic cosine of x
sinh(x)	hyperbolic sine of x
tanh(x)	hyperbolic tangent of x
Other functions	
ceil(x)	x rounded up to the nearest integer
exp(x)	e to the power of (ex)
fabs(x)	absolute value of x

floor(x)	x rounded down to the nearest integer
fmod(x,y)	remainder of x/y
log(x)	natural log of x, x>0
log10(x)	base 10 log of x, x>0
pow(x,y)	x to the power of y (xy)
sqrt(x)	square root of x, x>=0

4.11 Formatted I/O operations

Reading a character

The simplest of all input/output operations is reading a character from the standard input unit. Reading a character can be done by using the function getchar.

Syntax **variable_name = getchar ();**

The variable name is a character data type and the getchar() function waits for a key to be pressed.

E.g.: char name;

name = getchar ();

Here getchar is assigned some value which is of a single character.

Writing a character

To write a character putchar function is used.

Syntax

putchar (variable_name);

Variable name is a character data type. This function is used to display the character present in the variable_name.

E.g. : answer = 'y';
putchar (answer);

This statement will display 'y' on the screen.

Formatted input

When more than one value is to be given as an input the formatted input statement are used.

Syntax **scanf("control string ",arg1,arg2, ...argn);**

The control string specifies the fields format and the arguments arg1,arg2,.....argn specify the address of locations where the data is stored. The control string includes field specifications consisting of the conversion character %, a data type character and an optional number, specifying the field width blanks, tabs or new lines.

The blanks, tabs or new lines may be ignored.

E.g.:

Input an integer number

scanf ("%d %d", &a, &b);

Where a and b are integer data type.

Input a real number

scanf ("%f %f",&x,&y);

Where x and y are floating point constants.

Input a character strings

char name [10];

scanf ("%c",name);

Where name can hold 10 character values.

Reading mixed data type

scanf("%d %c %f %s", &count , code, &rate, name);

Here count is of integer data type, code of character data type which can accept one character, rate of float data type and name of string data type which can accept string of characters.

Formatted output

The printf function is used for printing captions and numerical results.

Syntax : printf ("control string ", arg1,arg2.....argn);

Control string consists of three type of items:

1. Character that will be printed on the screen as they appear.
2. Format specifications that define the output format for display of each item.
3. Escape sequence character such as \n,\t and \b.

The control string indicates how may arguments follow and what their types are. The arguments arg1,arg2.....argn are the variables whose values are formatted and printed according to the specifications of the control string.

E.g. :

printf(“programming in c”);
printf(“ \n”); new line
printf(“%d”,x); value of x
printf “the sum of two numbers is = %d”, x); the calculated value of x with the comment.

Output of integer numbers

The format specification for printing an integer number is :

%wd

w is the width of the output. E.g. printf(“%2d”,x);

Output of real numbers

Syntax : **%w.pf** **%w.pe**

The “p” indicates the number of digits to be displayed after the decimal point.

The “w” stands for the width of the number including the decimal point and the digits after that.

Eg : printf (“%7.4 f “, num);
if num = 98.1234 then the output is 98.1234
printf (“%10.2e”,num);

The output will b e 9.88e+01

Printing of a single character

Syntax : **%wc**

Here the single character will be displayed in the desired position. The character will be right justified in the field of w columns. If a left justification is required then a minus sign is included.

Eg : %5c

Here the given character is printed in the 5th position.

Printing of strings

Syntax : **%w.ps**

The 'w' is the width of the field and 'p' instructs that only the first 'p' characters of the strings are to be displayed. This display is right justified.

Eg : if NEW DELHI 110001 is to be printed which is of 16 characters including blank space then the o/p is

%s N E W D E L H I 1 1 0 0 0 1

%20S N E W D E L H I 1 1 0 0 0 1

%20.10 S N E W D E L H I

%.5 S N E W D

Other format codes

%c	single character
%d	decimal integer
%e	floating point value
%f	floating point value
%g	floating point value
%h	short integer
%i	decimal, hexadecimal or octal integer
%o	octal integer
%s	string
%u	unsigned decimal integer
%x	hexadecimal integer

4.12 Sum up

After reading this lesson one should be able to

- Know about all the arithmetic, relational, logical, assignment, increment, decrement, conditional, bitwise and special operators.
- Know about expressions
- Know about formatted and unformatted i/o functions

References

1. Programming in ANSI C, E.Balagurusamy, TMH, 1998

2. The sprit of C – Mullish Cooper, Jaico Publications

Lesson V
DECISION MAKING AND BRANCHING

5.0 Aim and objectives

In this unit we have discussed about all the decision making and looping statements along with the control structures used in C language.

After reading this lesson one should be able to

- Program using simple looping
- Know about decision making and control structures
- Know about branching statements
- Do all the programs

5.1 Introduction

Decision making and branching

Normally the statements are executed sequentially. In some situations we may have to change the order of execution based on certain situations or we may have to repeat a group of statements until certain conditions are satisfied. This is where we go in for decision making (or) control statements. They are as follows.

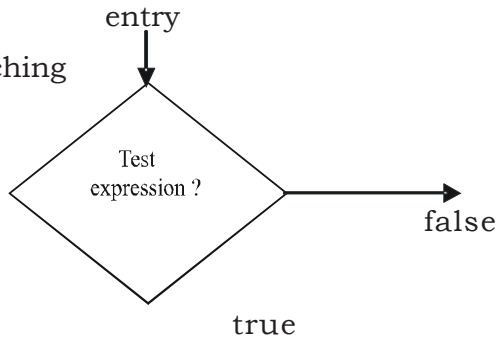
1. if statement
2. switch statement
3. conditional operator statement
4. go to statement

5.2 Decision making with if statement

The if statement is a powerful decision making statement which is used to control the flow of execution of statements. This statement allows a two-way decision making(i.e.) the computer evaluates the expression first and then depending on whether the value of the expression is true or false it transfers the control to a particular statement.

Syntax : if (test expression)

Two – way branching



The if statement can have different forms based on the complexity of conditions to be tested. They are

1. simple if statement
2. if.....else statement
3. nested if .....else statement
4. else if ladder

Simple if statement

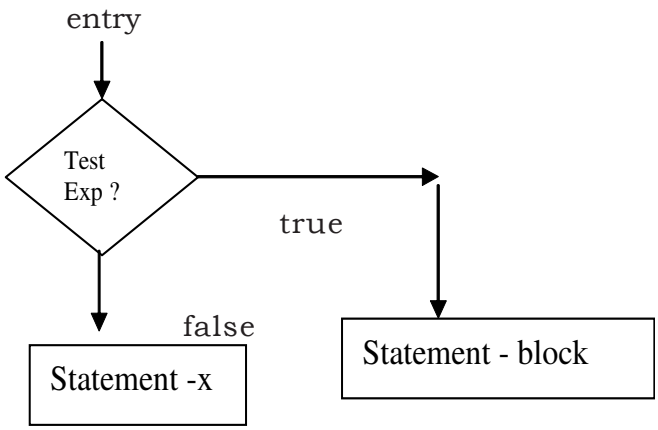
Syntax :

```
if ( test conditions)
{
Statement block;
}
Statement x;
```

The statement block may a single statement or a group of statements. If the test expression is true statement block will be executed, if the test expression is false then the statement x will be executed.

Note: When the condition is true both the statement block and the statement x are executed in sequence.

Flow chart of simple if control



E.g.:
```
if (mark >50)
{
tot_marks  = mark + 20 ;
}
printf("%d",tot_marks);
.....
.....
```

If ……else statement

This is an extension of the simple if statement.

Syntax

```
if (test expression)
{
true block statement(s)
}
else
{
false block statement(s)
}
statement x;
```

If test expression is true the true block statement is executed and immediately after that the statement x is executed. If the test expression is false then the false block statements are executed.