LECTURE NOTES
ON
C Programming & Data Structures

S. Syed Abuthahir

Assistant Professor

Department of Computer Science and Engineering

Madanapalle Institute of Technology and Science

Syllabus

18CSE102 C PROGRAMMING AND DATA STRUCTURES

Course Description: This course includes C program basics, control structures, arrays, files, pointers and data structures.

Course Objectives:

1. To make the student understand problem solving techniques and their applications
2. Students will be able to understand the syntax and semantics of C programming language
3. Develop algorithms for manipulating stacks, queues, searching and sorting.

UNIT I: C PROGRAMMING Structure of C Program, C Tokens: Variables, Data types, Constants, Identifiers, key words and Operators, Expressions. Control Structures: Conditional Statements (Simple if, if-else, Nested -ifelse, Switch). Iterative Statements (for, While, Do-While), Jump Statements (break, Continue).

UNIT II: FUNCTIONS & ARRAY Functions Introduction, User defined function, accessing a function, Function prototypes, Recursion, storage classes Arrays: Defining an array, processing an array, one dimensional arrays, two dimensional arrays. Searching: Linear and Binary search Sorting: Bubble Sort and Insertion Sort.

UNIT III: POINTERS AND STRUCTURE Pointers: Fundamentals of pointer, Pointer Declarations, Parameter passing: Pass by value, Pass by reference – Example Program: Swapping of two numbers and changing the value of a variable using pass by reference. Dynamic memory allocation. Structures: Defining a structure, processing a structure.

UNIT IV: DATA STRUCTURES Classification of Data Structure, Stack and Queues: stack, stack operations, stack implementations using arrays.Queue, queue operations, queue implementations using array, types of queues, applications of stack and queue.

UNIT V: STRINGS & FILES Declaring and Defining a string, Initialization of strings, Strings Library functions Files: File Definition, Opening and closing a data file, Reading and Writing a data file, Files I/O Functions.

Course Outcomes: Upon successful completion of the course, students will be able to

1. Understand problem solving techniques for a wide-range of problems.

2. Design and implement applications using functions, arrays, searching and sorting techniques.

3. Design and implement applications using pointers, structure and list.

4. Choose appropriate data structure depending on the problem to be solved.

5. Design and implement applications using Strings, Pointers and File processing.

Text Books:

1. The C Programming Language, Kernighan and Ritchie, 2 nd Edition, Prentice Hall, India 1988.

2. Alfred V. Aho, John E. Hopcroft and Jeffry D. Ullman, Data Structures and Algorithms, Pearson Education, New Delhi, 2006.

References:

1. Programming in ANSI C, E. Balagurusamy, Sixth Edition, Tata Mc-Graw Hill Publishing Co.Ltd.-New Delhi

2. Problem Solving & Program Design in C, Hanly, Jeri R and Elliot. B Koffman, Pearson Education,5th edition, 20007.

3. K. N. King ,"C Programming ": A Modern Approach, 2nd Edition 2nd Edition

4. Byron Gottfried , Jitender Chhabra , Programming with C (Schaum's Outlines Series)

Mode of Evaluation:Assignments, Internal Mid Examinations, External End Examination.

UNIT I: C PROGRAMMING

1. **Structure of C Program.**
2. **C Tokens: Variables, Data types, Constants, Identifiers, key words and Operators, Expressions.**
3. **Control Structures: Conditional Statements (Simple if, if-else, Nested - ifelse, Switch).**
4. **Iterative Statements (for, While, Do-While).**
5. **Jump Statements (break, Continue).**

INTRODUCTION:

A Computer is an electronic device that can perform activities that involve Mathematical, Logical and graphical manipulations. Generally, the term is used to describe a collection of devices that function together as a system.

It performs the following three operations in sequence.

1. It receives data & instructions from the input device.
2. Processes the data as per instructions.
3. Provides the result (output) in a desired form.

Basic Elements of a Computer System: Basic elements of a computer system are Mouse, Keyboard, monitor, memory, CPU, motherboard, Hard Disk, Speakers, Modem, power supply and processor.

Major Components of Computer:- Computer is basically composed of essentially the following: 1. Hardware 2. Software

1. Hardware: Computer hardware is the collection of various physical components of the computer, like the computer itself, the input-output devices.

2. Software: Software is set of instructions usually termed as programs which are required for processing activities of the computer.

COMPUTER LANGUAGES There are three types of programming languages.

1. Machine Languages: Computers respond only to machine language. This language is in terms of binary codes (0,1).

2. Assembly Languages: It uses mnemonic codes rather than numeric codes (as in machine languages). Ex. Add or A is used as a symbol for addition. It requires translators to convert into machine language. Like machine language, writing program in assembly language is also time consuming. These are also machine dependent.

3. High Level Languages (HLL): These are referred as problem oriented languages (POL). These are referred as third generation languages.

The advantages of these languages are

1. The high level languages are convenient for writing programs as they can be written without any codes. These languages follow rules like ―English‖ language.
2. Because of their English like nature, less time is required to write a program.
3. They are machine independent. A program written in any HLL can be run on computers of different types without any modifications.

ALGORITHMS

Algorithm is nothing but **step-by-step solution of a problem** which is written in a normal language is called “Algorithm”.

Or

Algorithm is an **effective procedure for solving a problem in a finite number of steps**

Ex 1: Algorithm to calculate area of square

```
Algorithm:  Step 1 : Start
             Step 2 : Read value for a (side)
             Step 3 : [Compute] Area = A * A
             Step 4 : Output Area
             Step 5 : Stop
```

Ex 2: Algorithm to find the average of three numbers

```
Algorithm:  Step1 : Start
             Step 2 : Enter Three Numbers A, Band C
             Step 3 : Compute Average = (A+B+C)/3
             Step 4 : Print Average
            Step 5 : Stop
```

Ex3: - Algorithm for finding the maximum of two numbers:

```
Algorithm: step1: start
                Step2: Read three numbers (A,B)
                Step3: If A>B , then goto step4 otherwise goto step5
                Step4: print ‘A’ as maximum otherwise
                Step5: Print ‘B’ is maximum.
                Step6: Stop.
```

Ex4:-Write an algorithm to find sum of n natural numbers.

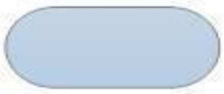
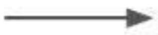
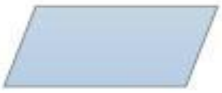
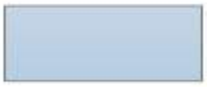
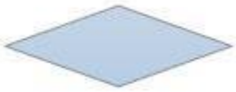
```
Algorithm:      step1: start
                Step2: Read N
                Step3: sum=0, i=1
                Step4: if i<n then goto step5 otherwise goto step 6
                Step5: sum=sum+i
                        I= i+1 then goto step3
                step6: print sum
                step7: stop.
```

Flow chart

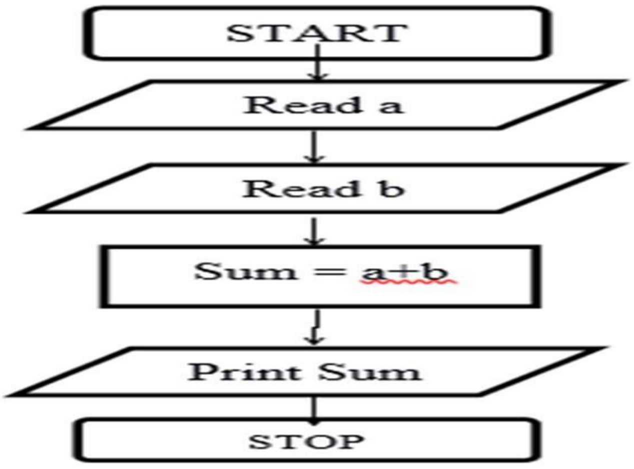
Graphical or symbolic representation of an algorithm is “Flow chart”

(Or)

Diagrammatic representation of the step-by-step solution to a given problem

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

Ex 1: Read two numbers and produce the sum of those numbers.



INTRODUCTION TO C LANGUAGE

1. C was developed at Bell Laboratories in 1972 by Dennis Ritchie.

2. C is a general-purpose computer programming language.

Significant Features of C Language

1. C is a powerful, flexible language that provides fast program execution.

2. C is a Procedural Language i.e. the programmer is required to provide step-by-step instructions for the CPU (central processing unit).

3. The success of C is due to its simplicity, efficiency, flexibility and small memory requirements.

4. Low Level features: C's power and fast program execution come from its ability to access low level commands, similar to assembly language, but with high level syntax.

5. Portability: C programs are portable i.e. they can run on any compiler with little or no modification. Compiler and Preprocessor makes it possible for C program to run it on different PC.

6. Modular Programming: It is a software design technique that increases the extent to which software is composed of separate parts, called modules. C program consist of different modules that are integrated together to form a complete program.

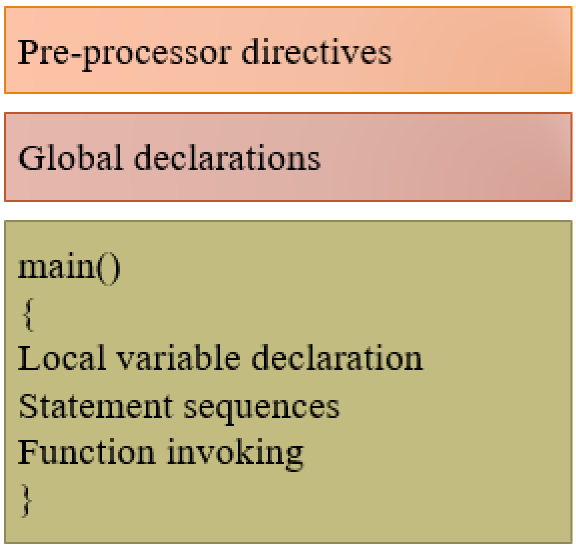
7. Efficient Usage of Pointers: C supports efficient use of pointers and pointers has direct access to memory.

8. Standard Library Concept:

9. Elegant Syntax:

10. Structured Programming Language

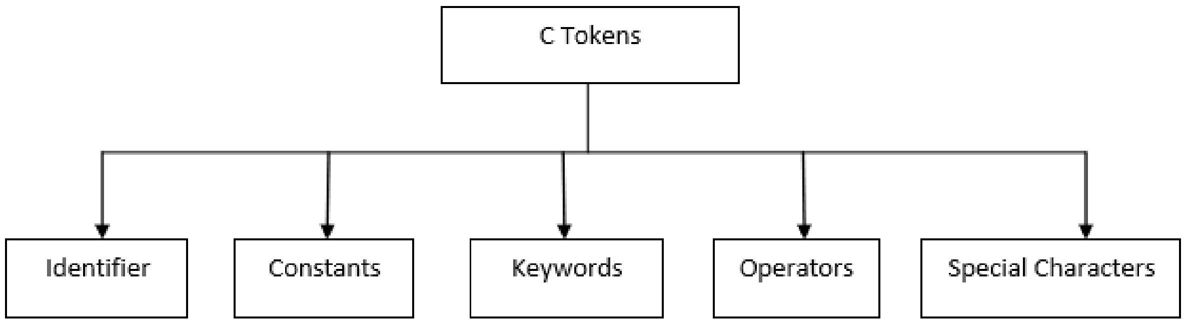
Structure of C program:



Sections	**Description**
Documentation section	We can give comments about the program, creation or modified date, author name etc in this section Example : /* comment line1 comment line2 comment 3 */
Link Section	Header files that are required to execute a C program are included in this section
Definition Section	In this section, variables are defined and values are set to these variables.
Global declaration section	Global variables are defined in this section. When a variable is to be used throughout the program, can be defined in this section.
Function prototype declaration section	Function prototype gives many information about a function like return type, parameter names used inside the function.
Main function	Every C program is started from main function and this function contains two major sections called declaration section and executable section.
User defined function section	User can define their own functions in this section which perform particular task as per the user requirement.

C Tokens

A token is the basic building block of a C program which can be recognized by the compiler.



1 Identifier

Identifier refers to name given to entities such as variables, functions, structures etc.

Identifier must be unique. They are created to give unique name to a entity to identify it during the execution of the program. For example:

int money;

int account balance;

Here, money and accountBalance are identifiers.

Also remember, identifier names must be different from keywords. You cannot use int as an identifier because int is a keyword.

Rules for naming identifiers

1. A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
2. The first letter of an identifier should be either a letter or an underscore.
3. There is no rule on how long an identifier can be. However, you may run into problems in some compilers if identifier is longer than 31 characters.

2. Constants

Constant in C means the content whose value does not change at the time of execution of a program.

Different Types of C Constants :

Constant	Type of Value Stored	Example
Integer Constant	Constant which stores integer value	5, 9, 35
Floating Constant	Constant which stores float value	5.25,0.22E-5, -2.0
Character Constant	Constant which stores character value	‘a’, ’x’
String Constant	Constant which stores string value	“hello”, “sample”, ”good”

3.Keywords

These are reserved words used in programming. There are 32 keywords predefined by ANSI C.
e.g. int n;
Here int is keyword and it indicates n is of type integer.

Keywords used while declaring variables	**Control flow related keywords**	**Storage class related keywords**	**User defined data type related keywords**	**Special operator related keyword**
1. Int 2. Char 3. Float 4. Double 5. Long 6. Short 7. Signed 8. Unsigned 9. Volatile 10. Void 11. Const	12. If 13. Else 14. Switch 15. Case 16. Default 17. Do 18. While 19. For 20. Return 21. Break 22. Continue 23. Goto	24. Auto 25. Static 26. Register 27. Extern	28. Enum 29. Typedef 30. Struct 31. Union	32. Sizeof

4. OPERATORS

An operator is used to describe an operation applied to one or several objects.

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

Arithmetic Operators:

Operator	Meaning of Operator
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Remainder after division (modulo division)

Increment and Decrement Operators:

Operator	Meaning of Operator

++	Increment operator (unary)
	a++; //post increment
	++a; //pre increment

--	Decrement operator (unary)
	a--; //post decrement
	--a; //pre decrement

Assignment Operators:

There are two types of assignment operators.

1. Simple Assignment
2. Compound Assignment

Operator	Meaning of Operator
Simple Assignment	
=	Assignment Operator e.g. x=5 5 is assigned to x
Compound Assignment	
+=	a += b is equivalent to a = a + b
-=	a -= b is equivalent to a = a - b
*=	a *= b is equivalent to a = a * b
/=	a /= b is equivalent to a = a / b
%=	a %= b is equivalent to a = a % b
&=	a &= b is equivalent to a = a & b
|=	a |= b is equivalent to a = a | b

^=	a ^= b is equivalent to a = a ^ b
<<=	a <<= b is equivalent to a = a << b
>>=	a >>= b is equivalent to a = a >> b

Relational Operators:

Relational Operators are used to check the relationship between two operands. If the relation is true, it returns value 1 and if the relation is false, it returns value 0. Relational operators are used in decision making and loops in C programming.

Operator	**Meaning of Operator**
==	Equal to e.g. 5 == 3 returns false or 0
>	Greater than e.g. 5 > 3 returns true or 1
<	Less than e.g. 5 < 3 returns fase or 0
!=	Not equal to 5 != 3 returns true or 1
>=	Greater than or equal to e.g. 5 >= 3 returns true or 1
<=	Less than or equal to e.g. 5 <= 3 returns false or 0

Logical Operators:

Logical operators are used to combine expressions containing relation operators. In C, there are 3 logical operators:

Operator	**Meaning of Operator**
&&	Logical AND e.g. if c=5 and d=2 then ((c == 5) && (d > 5)) returns false or 0
||	Logical OR e.g. if c=5 and d=2 then ((c == 5) || (d > 5)) returns true or 1
!	Logical NOT e.g. if c=5 then !(c == 5) returns false or 0

Conditional Operator:

Conditional operator takes three operands and consists of two symbols **?** and **:** . Conditional operators are used for decision making in C.

For example: c = (c > 0) ? 10 : 20;

If c is greater than 0, value of c will be 10 but, if c is less than 0, value of c will be 20.

Bitwise Operators:

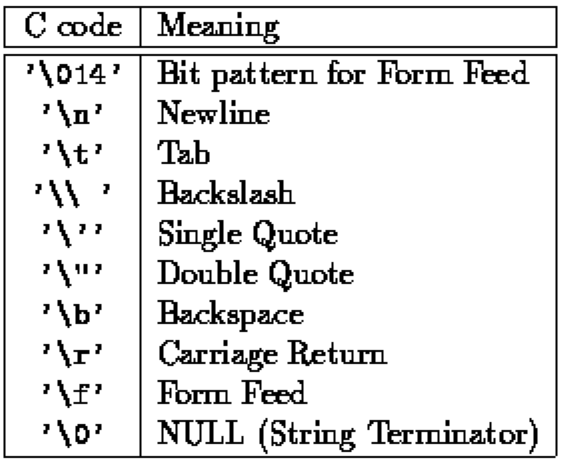
A bitwise operator works on each bit of data.

Operator	Meaning of Operator
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive OR
<<	Shift Left
>>	Shift Right
~	Bitwise Complement (One's Complement)

Special Operators:

Operator	Meaning of Operator
,	Comma operators are used to separate the variables or expressions. Example: int a,b,c =5*3;
sizeof	It is a unary operator which is used in finding the size of data type, constant, arrays, structure etc. e.g. sizeof int = 2 bytes sizeof float = 4 bytes } (based on MS-DOS OS) sizeof double = 8 bytes sizeof char = 1 byte
.	Access Operator is used to access any member of structure or union.
→	This operator is used to access any member of structure or union using pointer.

5.Special characters in C



C code	Meaning
'\014'	Bit pattern for Form Feed
'\n'	Newline
'\t'	Tab
'\\ '	Backslash
'\''	Single Quote
'\"'	Double Quote
'\b'	Backspace
'\r'	Carriage Return
'\f'	Form Feed
'\0'	NULL (String Terminator)

Variables:

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

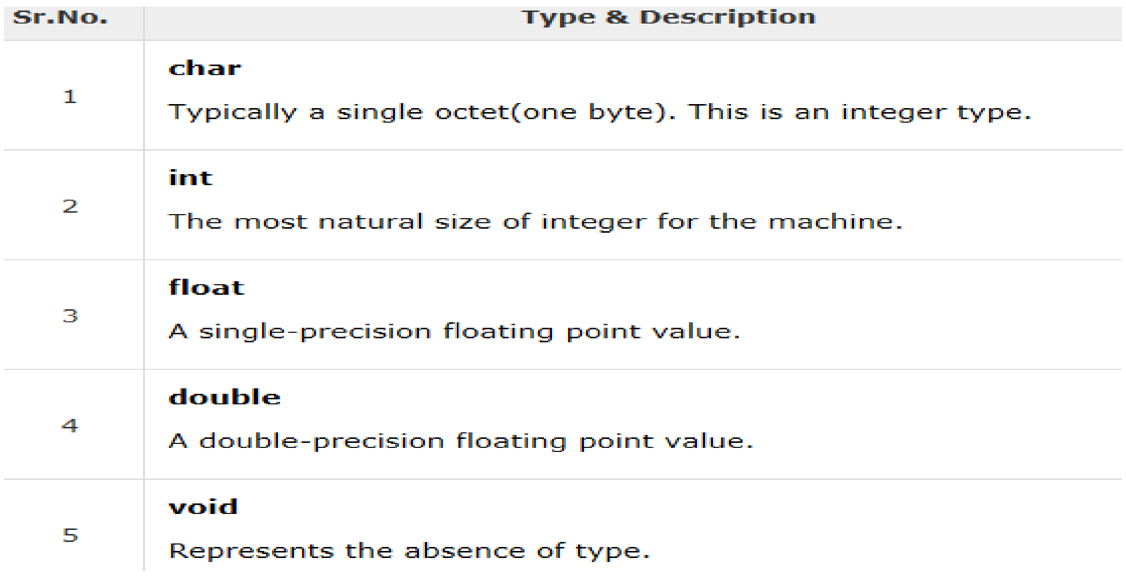
A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows −

```
type variable_list;
```

Some examples are −

```
int d = 3, f = 5;          // definition and initializing d and f.
byte z = 22;               // definition and initializes z.
char x = 'x';             // the variable x has the value 'x'.
```

The following basic variable types −



Sr.No.	Type & Description
1	**char** Typically a single octet(one byte). This is an integer type.
2	**int** The most natural size of integer for the machine.
3	**float** A single-precision floating point value.
4	**double** A double-precision floating point value.
5	**void** Represents the absence of type.

Datatypes

Data Types:

The type of the value that a variable can store in the memory can be defined using data type. ANSI C supports four types of data types

1. Primary data type (or) Primitive (or) basic (or) fundamental
2. User-defined data type
3. Void data type
4. Derived data type

Primitive Data Types:

The primitive data types are also called primary data types (or) basic data type's .The primitive data types of C language are classified into following categories named as integer, character, floating point and double.

Integer:

- Integers are whole numbers with a range of values, range of values are machine dependent.
- Generally an integer occupies 2 bytes memory space and its value range limited to -32768 to +32767.
- A signed integer use one bit for storing sign and rest 15 bits for number.
- To control the range of numbers and storage space, C has three classes of integer storage namely short int, int and long int in both signed and unsigned forms.

Signed :	signed short int
	signed int
	signed long int
Unsigned :	unsigned short int
	unsigned int
	unsigned long int

- A short int requires half the amount of storage than normal integer.
- Unsigned integers are always positive and use all the bits for the magnitude of the number. Therefore, the range of an unsigned integer will be from 0 to 65535.
- The long integers are used to declare a longer range of values and it occupies 4 bytes of storage space.

1. For signed data type $(-2^{n-1})$to $(+2^{n-1}-1)$
2. For unsigned data type 0 to $(2^{n}-1)$

Size and range of integer data type:

Type	Size (bits)	Range	Place holder
Signed short int	8	-128 to 127	%d(or)%i
int or signed int	16	-32768 to 32767	%d(or)%i
long int (or) signed long int	32	-2^{31} to $2^{31}-1$	%ld
unsigned short int	8	0 to 255	%u
unsigned int	16	0 to 65535	%u
unsigned long int	32	0 to 2^{32} -1	%lu

Character:

- Character type variable can hold a single character and are declared by using the keyword **char**.
- As there are singed and unsigned int (either short or long), in the same way there are signed and unsigned chars; both occupy 1 byte each, but having different ranges.
- Unsigned characters have values between 0 to 255; signed characters have values from –128 to 127.

Size and range of character data type

Type	size (bits)	Range	Place holder
char (or) signed char	8	-128 to 127	%c
unsigned char	8	0 to 255	%c

Floating point: The float data type is used to store fractional numbers (real numbers) with 6 digits of precision. Floating point numbers are denoted by the keyword **float**. When the accuracy of the floating point number is insufficient, we can use the double to define the number.

Double: The double is same as float but with longer precision (14 digits of precision) and takes double space (8 bytes) than float.

Long double: long double is used for extended precisions. The size if the long double is 10bytes(80bits).

Size and range of integer data type

Type	Size (bits)	Range	Place holder
float	32	3.4E-38 to 3.4E+38	%f
double	64	1.7E–308 to 1.7 E+308	%f
long double	80	3.4E–4932 to 1.1 E+4932	%lf

<u>User defined Data Types:</u>

We have two types

1. typedef
2. enumerator

<u>typedef:</u>

- **C supports** a feature known as **"type definition"** that allows user to define an identifier that would represent an existing data type.

The general syntax of the typedef is shown below

Syntax: **typedef type identifier;**

or

typedef existing_data_type new_user_define_data_type;

typedef: typedef is a key word that allows the programmer to create a new data type.

Type->Refers to an existing data type

Identifier->The new name given to the data type

Example: typedef int units;

typedef float marks;

Advantages of typedef data type:

The main advantage of typedef is that we can create meaningful data type names for increasing the readability of the program.

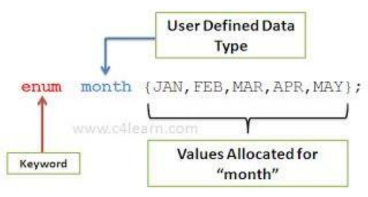
<u>Enumerated data Type:</u>

- Enumerated Types allow us to create our own symbolic names for a list of related ideas.
- It is defined as

Syntax:

enum Identifier{Value 1,value 2,…..value n};

Example:



```
enum month {JAN,FEB,MAR,APR,MAY};
```

enum: It is the keyword which allows the user to create our own symbolic names for a list of ideas.

Control structures

Conditional Statements (simple if, if-else, Nested-if-else, Switch)

- C program executes program sequentially.
- Sometimes, a program requires checking of certain conditions in program execution.
- These statements are called as 'Decision Making Statements' or 'Conditional Statements.' Followings are the different conditional statements used in C.

1. if Statement
2. if-else Statement
3. Nested if-else Statement
4. Switch Case 3

The conditional control statements are divided into two types:

1. Decision : simple-if, if-else, nested-if-else, else-if-ladder.
2. Selection: switch

1.if statement

Syntax:

```
if (test expression)
{
     statement/s to be executed if test expression is true;
}
```

If the test expression is true then, statements for the body if, i.e, statements inside parenthesis are executed. But, if the test expression is false, the execution of the statements for the body of if statements are skipped.

e.g.

```
int main()
{
     int a=40,b=40;
     if (a == b)
     {
          printf("a and b are equal");
     }
}
```

Output: a and b are equal

2. if else statement

<u>if-else:</u> The if-else statement is the extension of simple-if statement. The if else statement in C programming language is used to execute a set of statements if condition is true and execute another set of statements when condition is false. The syntax is

Syntax:

```
if(test_expression)
{
        True _block_statements;
}
else
{
        False _block_statements;
}
```

Example2: Write a C program to check the student is pass (or) fail using if-else.

```
#include<stdio.h>
        #include<conio.h>
        void main()
        {
                int marks=50;
                clrscr();
                if(marks>=40)
                {
                        printf("Student is Pass");
                }
                else
                {
                        printf("Student is Fail");
                }
                getch();
        }
```

<u>**Nested–if-else:**</u> The statements within the if statement can contain another if statement and which in turn may contain another if and so on is called nested if-else statement. The nested if statement in C programming language is used when multiple conditions needs to be tested. The inner statement will execute only when outer if statement is true otherwise control won't even reach inner if statement.

Syntax:

```
if (condition1)
{
true_block_statements1;
}
            else if(condition2)
{
                        true_block_statements2;
            }
else if(condition3)
{
             true_block_statements3;
 }
 else if(conition4)
 {
              true_block_statements4;
 }
 else
 {
              default statement;
 }
```

Example: Write a program to read three numbers and find the largest one by using else-if-

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a, b, c
clrscr ( ) ;
printf("Enter 1st number:");
scanf("%d", &a);
printf("Enter 2nd number:");
scanf("%d", &b);
printf("Enter 3rd number:");
scanf("%d", &c);
if ((a>b) && (a>c))
{
```

```
printf("Highest Number is: %d", a);
}
else if ((b>a) && (b>c))
{
printf("Highest Number is: %d", b);
}
else
{
printf("Highest Numbers is: %d", c);
}
}
```

Selection:

Switch statement: A switch statement allows a variable or value of an expression to be tested for equality against a list of possible case values and when match is found, the block of code associated with that case is executed. The syntax of switch statement is:

Syntax:

```
switch (expression)
{
  case     constant1: statements_block1;
                                break;
  case    constant2: Statements_lock2;
                          break;
  default:  default_block
}
```

switch: It is the keyword indicating the start of the switch statement.

expression: It is an expression (or) variable (or) constant that results in an integer value (or) character value with which the comparisons will be done for equality with the case constants inside the switch statement.

case: It is the keyword indicating each of the alternatives for the value of the expression in the switch statement. It will have a break as the last statement for each case.

break: It is the statement that transfers the execution control to go outside the switch statement when it is executed. If we don't write the break statement as the last statement of each case block, then the execution control will not break and will also execute the next cases below, even though they are not matching to the expression's resulting value.

default: It is the keyword used as the last case option and it will get executed when none of the above case alternatives are not equal to the value given by the switch expression.

Example:

```
#include <stdio.h>
#include<conio.h>
void main()
{
        char grade ;
        printf(“enter your grade”);
        scanf(“%c”, &grade);
                switch(grade)
                {
                        case  'A' :
                                printf("Excellent!\n" );
                                 break;
                        case 'B' :
                                printf("good!\n" );
                                 break;
                        case  'C' :
                                printf("Well done\n" );
                                break;
                        case  'D' :
                                printf("You passed\n" );
                                break;
                        case  'F' :
                                printf("Better try again\n" );
                                break;
                        default :
                                printf("Invalid grade\n" );
                }
                printf("Your grade is  %c\n", grade );
}
```

Looping Control Statements (or) Repetition (or)Iterative statements: Loop control statements in C are used to execute a block of code several times until the given condition is true. Whenever we need to execute some statements multiple times, we need to use a loop statement.

2. The various types of loop control statements are: **for, while and do while.**

So, there are 2(two) types of looping control statements.

1. Entry controlled loop

2. Exit controlled loop

Entry controlled loop: In such type of loop, the test condition is checked first before the loop is executed. Examples of Entry controlled loop are: for and while.

Exit controlled loop: In such type of loop, the loop is executed first, then condition is checked, the loop executed at least one time. Examples of exit controlled loop is do-while.

for: Repeats a block of code multiple times until a given condition is true. Initialization, looping condition and update expression (increment/decrement) is part of for loop. The syntax is

```
Syntax:  for(initialization;condition;inc/dec)
              {
                     Body of the loop;
              }
```

1. for is a keyword (reserved word)

2. ***initialization*** :The initialization is usually an assignment statement that sets the loop control variable. For Example: i=1 and count=0 Here i, count are loop control variables.

3. ***condition*** **:**The condition is a relational expression that determines when the loop exits
Example: i>10 that determines when the loop will exit. if the condition is true, the body of the loop is executed; otherwise the loop is terminated.

4. ***increment/decrement*****:** The increment/decrement defines how the loop control variable changes each time the loop is repeated.
Example: i=i+1

These three major sections must be separated by semicolons. The for loop continues toexecute as long as the condition is true.

Example : Write a C program to print n natural numbers using for loop.

```
#include <stdio.h>
   void main()
{
         int i,n;
         printf(“enter n value”);
         scanf(“%d”, &n);
         for(i=1; i< n; i++)
         {
                   printf("value of i: %d\n", i);
         }
}
```

while: Repeats a block of code multiple times until a given condition is true.

Syntax:

```
while (condition)
{
         Body of the loop
}
```

1. **while** is keyword
2. **condition**: The Test condition is evaluated and if the condition is true.
3. **Body of the loop:** The body of the loop may have one or more statements.

Example: Write a C program to print sum of n numbers using while loop.

```
#include<stdio.h>
void main()
{
int i = 1,sum = 0,n;
printf(“enter the value of n”);
scanf(“%d”,&n);
while(i<=n)
{
sum = sum + i;
i = i + 1;
}
printf(“Total : %d “,sum);
}
```

<u>do-while:</u> Similar to while loop, but it tests the condition at the end of the loop body. The block of code inside do while loop will execute at least once. The syntax of the do-while loop is :

syntax:

```
do
{
Body of the loop;
        } while (condition);
```

1. Where do and while are two keywords.
2. Here the body of the loop is executed first. At the end of the loop, the condition in the while statement is evaluated. If the condition is true, the program continues to execute the body of the loop once again. This process continues as long as the condition is true. When the condition becomes false, the loop will be terminated and the control goes to the statements that appears immediately after the while statement. Remember the body of the loop is always executed at least once.
3. There is semicolon at the end of while(condition); in do-while loop

Example:

```
#include <stdio.h>
#include<conio.h>
void main()
{
        int a = 10;
        do
        {
                printf("value of a: %d\n", a);
                a = a + 1;
          }while( a < 15 );

  }
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
```

Unconditional Control Statements (or) Loop Control and Jump Statements: Loop control statements alter the normal execution path of a program. Loop control statements are used when we want to skip some statements inside loop or terminate the loop immediately when some condition becomes true.

break: The break statement is used to stop the execution of loop and switch case statements. It means we can use break statement inside any loop(for, while and do-while). It terminates the loop immediately and program control resumes at the next statement following the loop. We can use break statement to terminate a case in the switch statement.

Syntax : break;

Example:

```
#include<stdio.h>
    #include<conio.h>
    void main()
     {
int i;
clrscr();
for(i=1;i<10 ; i++)
{
if(i>5) break;
printf("%d\t",i); // 5 times only
}
getch();
}
```

```
Output: 1       2       3       4       5
```

continue: The continue statement is used for skipping part of loop's body. It means We can use continue statement inside any loop(for, while and do-while). It skips the remaining statements of loop's body and starts next iteration.

Syntax: continue;

Example1:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i;
        int j = 8;
```

```
                for( i = 0; i <= j; i ++ )
                {
                        if( i == 5 )
                        {
                                continue;
                        }
                        printf("Hello %d\n", i );
                }
                getch();
        }
```

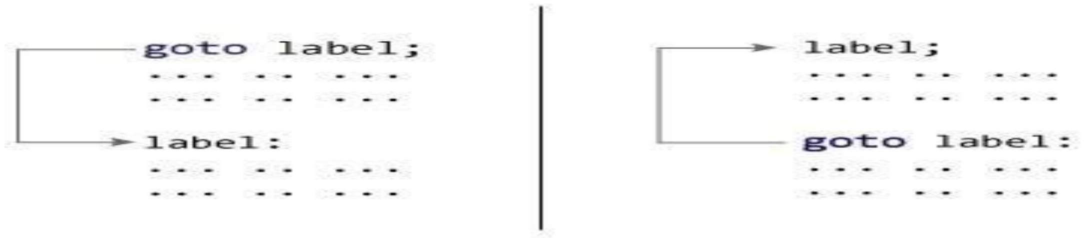
```
Output:     Hello 0
            Hello 1
            Hello 2
            Hello 3
            Hello 4
            Hello 6
            Hello 7
            Hello 8
```

goto:

1. C supports the “goto‟ statement to jump unconditionally from one point to another in the program.
2. The goto requires a label in order to identify the place where the jump is to be made.
3. A label is any valid variable name and must be followed by a colon(:).
4. The label is placed immediately before the statement where the control is to be transferred.
5. The label can be anywhere in the program either before or after the goto label statement.
6. During running of a program, when a statement like “goto begin;”is met, the flow of control
    1. will jump to the statement immediately following the label “begin:” this happens
    2. unconditionally.
7. goto‟ breaks the normal sequential execution of the program.
8. If the “label:” is before the statement “goto label;” a loop will be formed and some

statements will be executed repeatedly. Such a jump is known as a „backward jump‟.

9. If the “label:” is placed after the “goto label;” some statements will be skipped and the jump is known as a “forward jump”.



Example:

```
#include<stdio.h>
void main()
{
    int x;
    printf(“EnterNumber:”);
    scanf(“%d”, &x);
    if(x%2==0)
    {
        goto even;
    }
    else
    {
        goto odd;
    }
    even:   printf(“\n   %d   is   Even   Number”);
return;
    odd: printf(“ \n %d is Odd Number”);
}
```

Output:

```
Enter a Number: 55
55   is   Odd
Number.
```

Return: The return statement forces a return from a function and can be used to transfer a value back to the calling routine. It has these two forms:

syntax:

```
return;
        Or
return value;
        Or
return expression;
```

Example1:

```
#include<stdio.h>
        void main()
        {
                int number;
                clrscr();
                printf("Enter an integer\n");
                scanf("%d",&number);
                printf("Integer entered by you is %d\n", number);
                return;
        }
```

Output:

```
Enter a number          5
Number entered by you is 5
```

Example2:

```
#include<stdio.h>
        int main()
        {
                int number;
                clrscr();
                printf("Enter an integer\n");
                scanf("%d",&number);
                printf("Integer entered by you is %d\n", number);
                return 0;
        }
```

Output:

```
Enter a number          5
Number entered by you 5
```

Example3:

```
#include<stdio.h>
        #include<conio.h>
        int sum();
        void main()
        {
                int addition;
                addition = sum();
                printf("\nSum of two given values = %d", addition);
                return;
        }
        int sum()
        {
                int a = 50, b = 80, sum;
                sum = a + b;
                return sum;
        }
```

Output:

```
Sum of two given values = 130
```

/* End of first chapter please refer class work ad lab observation for more examples*/