Unit 1

C Fundamentals : Constants, Variable & Keywords in C, Operators, Bitwise Operations, Decision control & Looping Statements.

Arrays & Pointers : Arrays, Functions, Recursive Functions, Pointers, String Manipulations, Structure, Union, ~~Functio~~ Enumeration, MACROS

File Handling : File operations - Open, Close, Read, Write & Append.

CO1 : Solve Mathematical problems using C programming Language

* Introduction to C

C is a general-purpose programming language created by Dennis Ritchie at the Bell Laboratories in 1972. It is a very popular language, despite being old. C is strongly associated with UNIX, as it was developed ~~by~~ to write the UNIX operating system.

* Features of C

1) Simple and Efficient

The basic syntax style of implementing C language is very simple and easy to learn. This makes the language easily comprehensible and enables a programmer to redesign or create a new application. C is usually used as an introductory language to introduce programming to school students because of this feature.

2) Portability

Portability means executing the same application on different system. In C language three files are generated; the source code file has .c extension, the compiled file has .obj extension and the executable file has .exe extension. The program written and compiled on one Operating system say Windows, can be able to run on other windows based systems.

3) Structured Programming Language

C language is a structured programming language because we can create functions in the C language. Using functions we can separate a particular operation from the main program and then use it again and again. A structured language is not just about having the ability to create functions, but it supports loops, conditional statements, etc. All of this we will cover in detail in the upcoming tutorials.

4) Powerful.

C language is a very powerful programming language. It has a broad range of features like support for many data types, operators, keywords, etc; allows structuring of code using functions, loops, decision-making statements, then there are complex data-structure like structures, arrays, etc and

pointers, which makes C quite resourceful and powerful, etc.

Using the C language you can easily read, write and create files. This may sound like a basic feature today, but in the early 1990s, this was a game-changer.

5) Rich Steindard Library.

The C programming language has a rich library which offers useful built-in functions. These libraries are called Header files in C language. Apart from these functions, you can also add or define user-defined functions to library. These functions help in solving numerous types of problems easily and also help in better & clean coding.

6) Separate Compilation

C language code is compiled and then it is run. A small piece of code will compile faster while a large code will take time to get compiled. In C language you can break your code and put it in multiple source code files. C language will compile the files separately & then link them together for execution. This makes compilation fast. Another plus point of this is, multiple programmers / developers can work on different code files while working on a single project.

7) Middle-level language

The C programming language brings together the best of both worlds. A low level language is generally fast, powerful but hard to understand and write code in. Whereas a High level language is easy for us to understand and write code in, it is also highly portable, but it is generally slow and is unable to directly talk to the system hardware. Hence, the C programming language is said to be Middle level programming language, allows manipulation of bits, bytes & addresses, hence providing low-level access to computer systems, while being easy to use, portable and supporting all other features of a High level language, etc.

8) Syntax Based language

Like most high level languages, for example, Java, C++, C#, the C language has a syntax, there are proper rules for writing the code, and the C language strictly follows it. If you write anything that is not allowed, you will get a compile-time error, which happens when the compiler is unable to compile your code because of some incorrect code syntax.

Page :
Date :

9) Format Free language

The C language is a format free language. There are no line numbers needed in the C language code, or we can say that line number holds no significance. There is no need to place statements on a specified location on a line

10) Compiled language

Every program written in C should be compiled first and then execute. Without compilation the code will not execute because C is compiled based programming language.

11) Case Sensitive language.

In C, uppercase and lowercase characters are different. That means 'if' is not same as 'IF' in C language.

* The C character set

Alphabets	A, B ..... Y, Z
	a, b, ... y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 9
Special Symbols	~ ` ! @ # % ^ & & *
	( ) _ - + = | \ { } [ ]
	: ; " < >, . ? /

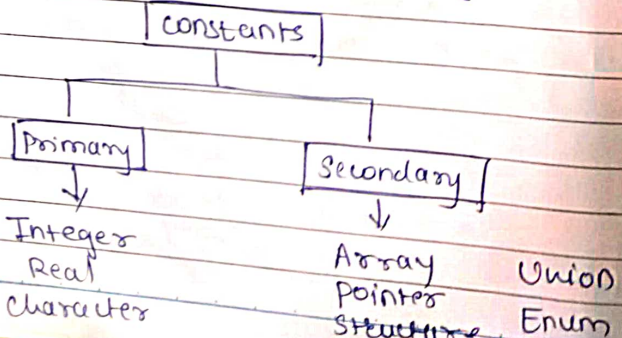
* Constants, Variables and Keywords

– Constant

The specific alphabetical or numerical value that never gets changed during the processing of instructions is called as constant.
Constants are also called literals.
Constants can be any of the data types.
It is considered best practice to define constants using only upper-case names.
C constants can be divided into two major categories:

a) Primary Constants
b) Secondary Constants

These constants are further categorized as below



We can create constants in C programming language by using two of concepts mentioned below

- By using '#define' preprocessor

```
#define CONSTNAME value;
```

- By using 'const' keyword

```
const datatype constname = value;
  or
const datatype constname;
```

– Variables

An entity that may vary during program execution is called a variable. Variable names are names given to locations in memory where values are stored. These location can contain integer, real or character values or constants

Rules for constructing Variable names

1) A variable name is any combination of 1 to 31 alphabets, digits or underscore.
   Some compilers allow variable names whose length could be upto 247 characters.
   Still, it would be safer to stick to the rule of 31 characters.
2) The first character in the variable name must be an alphabet.
3) No commas or blanks are allowed within variable name.
4) No special symbol other than underscore.
5) Variable names are case sensitive

The Variable names should be informative. It should describe the purpose of it by its name. It is mandatory to declare a variable which is going to be used in program.

Syntax for declaring variable

data-type variablename;

example
int marks;
char ch;
float salary;

Variable Initialization : means assigning a value to the variable.

Syntax:
data_type variablename = value;

example
int a = 10;

Types of variables

1) Local variable : These variables are declared and initialized within function. The scope of these variables is within the function where they are declared and initialized and can't be accessed outside of the function.

2) Global variable : The global variables are declared outside the main function. The scope of these variables is throughout the program. Every function in the program can access them.

Camlin

Keywords

Keywords are basically the reserved words which have special meaning. The meaning of keywords cannot be changed. All the keywords are written in lowercase letters.

The keywords cannot be used as variable names because if we do so, we are trying to assign a new meaning to the keyword, which is not allowed.

There are only 32 keywords available in C.
27 by Dennis Ritchie
5 by ANSI (American National Standard Institute)

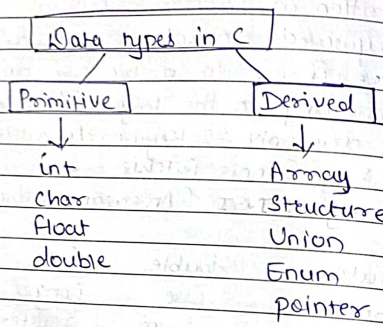
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile

1) Data type keywords : int, char, float, double
2) Qualifier keywords : signed, unsigned, short, long
3) Loop control structure keywords : for, while, do
4) User defined keywords : typedef, enum
5) Jumping control keywords : break, continue, goto
6) Storage class keywords : auto, register, extern
7) Function keywords : void, return
8) Decision keywords : if, else, switch, case, default
9) Derived keywords : struct, union
10) Other keywords : const, volatile, sizeof

Camlin

* Data Types

In C programming, data types are declarations for variables. This determines the type and size of data associated with variables.



1. Integer type.

Int or signed int	2 bytes	-32768 to 32767
unsigned int	2	0 to 65535
Short int or signed short int	1	-128 to 127
unsigned short int	1	0 to 255
Long int or signed long int	4	-2147483648 to 2147483647
unsigned long int	4	0 to 4294967295

2. Floating point.

Float	4 bytes	3.4E-38 to 3.4 E+38
Double	8	1.7 E-308 to 1.7 E+308
long double	10	3.4 E-4932 to 1.1 E+4932

3. Character Type

char or signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255

* Identifiers

- C identifiers represent the name in C program for example, variables, functions, arrays, structure union, labels, etc.
- ~~An identifier~~ Rules for constructing C identifiers.

a) The first character of an identifier should be either an alphabet or an underscore and then it can be followed by any of character, digit or underscore.
b) It should not begin with any numerical digit.
c) In identifiers, both uppercase & lowercase letters are distinct. Therefore we can say that identifiers are case sensitive.
d) Commas or blank spaces cannot be specified within an identifier.
e) Keywords cannot be represented as identifier.
f) The length of identifiers should not be more than 31 characters.
g) Identifiers should be written in such a way that it is meaningful, short and easy to read.

* Identifiers

* Difference between Keyword & Identifier

Keyword	Identifier
1) Keyword is pre-defined word	1) Identifier is a user-defined word
2) It must be written in lowercase letter	2) It can be written in both lowercase and uppercase
3) It does not contain underscore	3) It can contain underscore character
4) It is collection of alphabetical characters	4) It is collection of alphanumeric characters.

* Operators

An operator is simply a symbol that is used to perform operations.

1) Arithmetic Operators : Addition, Subtraction, division, multiplication.
2) Relational operators : ==, !=, >, <, >=, <=
3) Shift operators
4) Logical operators : Logical AND (&&), Logical OR (||), Logical NOT (!)
5) Bitwise operators : &, |, ^, ~, <<, >>
6) Ternary operator (?:) on 3 operands
   A ? B : C means if A is true then perform B otherwise perform C.
7) Assignment operator =, +=, -=, *=, /=, %=, <<=, >>=, &=, ^=, |=

— Operator Precedence in C

Precedence means priority. Each operator has some priority. If we use multiple operators in a single expression, operator precedence helps us to solve the sequence of operations to be performed in the expression. So, in the expression, Operator which has highest precedence will be evaluated first & lowest precedence operator will be evaluated at last.

Category	Operator	Associativity
Postfix	( ) [ ] -> ++ --	Left to Right
Unary	+, -, !, ~, ++, --, &, sizeof	Right to left
Multiplicative	* / %	L to R
Additive	+ -	L to R
Shift	<< , >>	L to R
Relational	<, <=, >, >=	L to R
Equality	==, !=	L to R
Bitwise AND	&	L to R
Bitwise XOR	^	L to R
Bitwise OR	|	L to R
Logical AND	&&	L to R
Logical OR	||	L to R
Conditional	?:	R to L
Assignment	=, +=, -=, *=, /=, %=, &=, ^=, |=	R to L
Comma	,	L to R

* Bitwise Operations:

1) The bitwise AND (&) : 2 nos. as operands & does AND on every bit. The result of AND is 1 only if both bits are 1.
2) The bitwise OR (|) : 2 nos. as operands & does OR on every bit. The result of OR is 1 if any of 2 bits is 1.
3) The bitwise XOR (^) : The result of XOR is 1 if 2 bits are different.
4) The left shift (<<) : left shifts the bits of first operand, 2nd operand decides the number of places to shift.
5) The right shift (>>) : right shifts the bits of first operand, the second operand decides the number of places to shift.
6) The bitwise NOT (~) : inverts all bits of it.

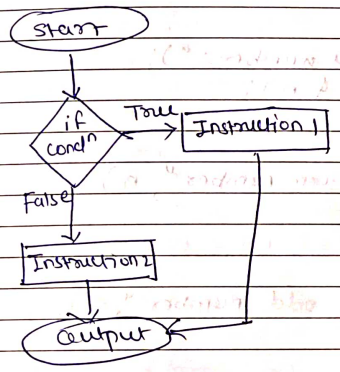
* Decision control & Looping Statements:

Control statements are broadly divided into 4 categories:
Decision Making Statements,
Selection Statements,
Looping Statements
Jump Statements.

(A) Decision making statements are used to check different given conditions depending upon which the flow of control can be decided. There are 4 decision making statements:
if, if-else, else if ladder & nested if else.

1] if statement:

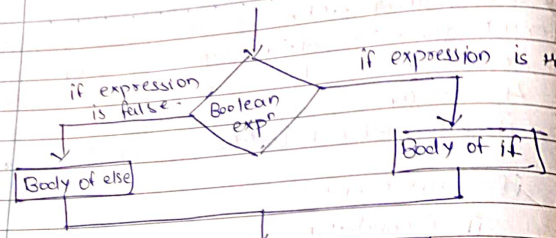
```
if (condition 1).
{
    statements;
}
```




```
# include <stdio.h>
int main()
{ int n = 0;
printf ("Enter a no:");
scanf ("%d", &n);
if (n%2 == 0)
{
printf ("%d is even
        number", n);
}
return 0;
}
```

2] if - else statement:

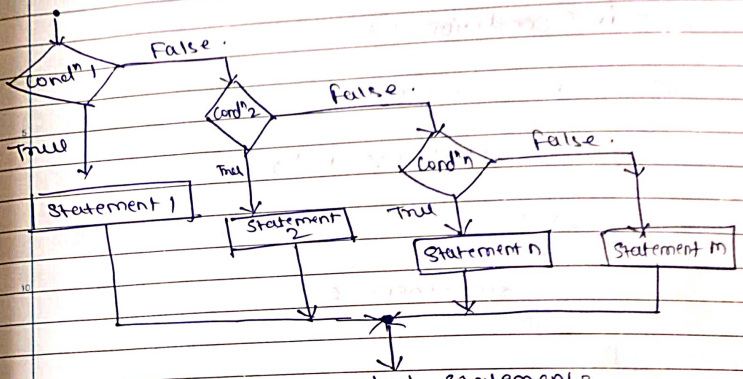
```
if (condition 1)
{
    statement 1;
}
else
{
    statement 2;
}
```




```
# include <stdio.h>
Void main()
{ int n=0;
printf ("enter a number:");
Scanf ("%d", &n);
if (n%2==0)
{
  printf ("%d is even number", n);
}
else
{
  printf ("%d is odd number", n);
}
}
```

3] else if ladder

```
if (condition 1)
{ Statement 1; }
else if (condition 2)
{ Statement 2; }
. . .
else if (condition n)
{ Statement n; }
else { Statement m; }
```




```
# include <stdio.h>
  int main()
{  int marks;
   printf (" Enter your marks ");
   Scanf ("%d", &marks);
   if (marks > 85 && marks <= 100)
   {
     pf (" you scored a A ");
   }
   else if (marks > 60 && marks <= 85).
       pf (" you scored a B+ ");
   else if (marks > 40 && marks <= 60)
       pf (" you scored B ");
   else if (marks > 30 && marks <= 40)
       pf (" you scored C ");
   else
       pf (" you are fail ");
}
```

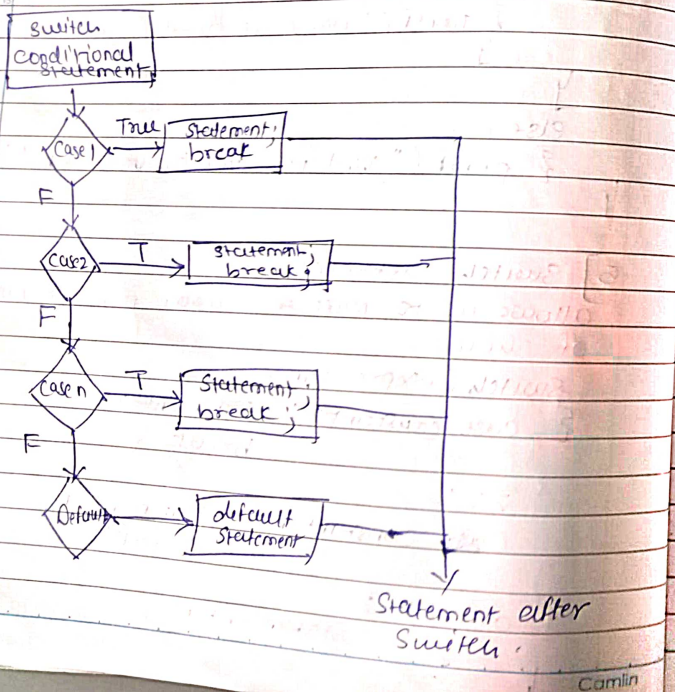
```
#include <stdio.h>
int main()
{
  int var1, var2;
  printf("Input the value of var1: ");
  scanf("%d", &var1);
  printf("Input the value of var2: ");
  scanf("%d", &var2);
  if (var1 != var2)
  {
    printf("var1 is not equal to var2 \n");
    if (var1 > var2)
    { printf("var2 is greater than var1 \n"); }
    else
    { printf("var2 is greater than var1 \n"); }
  }
  else
  { printf("var1 is equal to var2 \n"); }
}
```

5] Switch Statement

allows us to make a decision from the number of choices.

```
switch (expression)
{ case constant1: Statement 1;
                  break;
  . . .
  case constant n: Statement n;
                   break;
  default:
          Statement m;
}
```

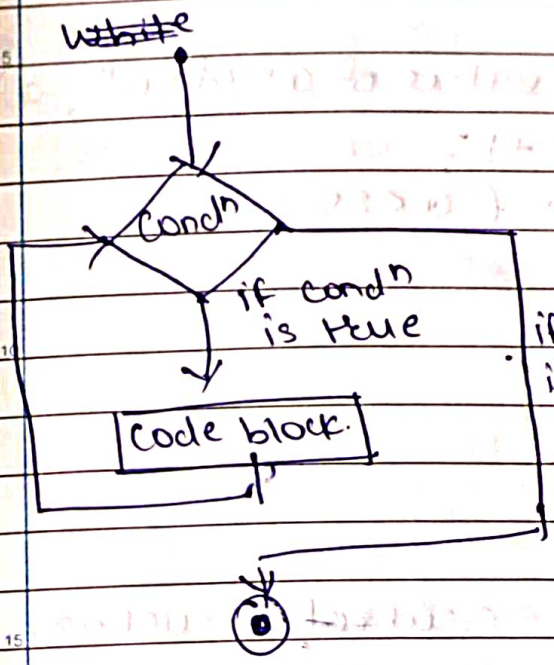
- Break : Keyword, is used to stop the execution inside a switch block. It helps to terminate the switch block and break out of it. If we do not use break statement, all statements after the matching label are also executed.
- Default : keyword, is used to specify the set of statements to execute if there is no case match.
- The expression is evaluated once and compared with values of each case label.
- If there is a match, the corresponding statement after the matching label are executed
- If there is no match, the default statements are executed




```
#include <stdio.h>
void main()
{ char operation;
  double n1, n2;
  pf ("Enter an operator (+, -, *, /): ");
  sf ("%c", &operation);
  pf ("Enter two operands: ");
  sf ("%lf %lf", &n1, &n2);
  switch (operation)
  {
    case '+':
      pf ("%.1lf + %.1lf = %.1lf", n1, n2, n1+n2);
      break;
    case '-':
      pf ("%.1lf - %.1lf = %.1lf", n1, n2, n1-n2);
      break;
    case '*':
      pf ("%.1lf * %.1lf = %.1lf", n1, n2, n1*n2);
      break;
    case '/':

    default:
      pf ("Error! operator is not correct");
  }
```

When the condition become false, the program control passes to the line immediately following the loop.

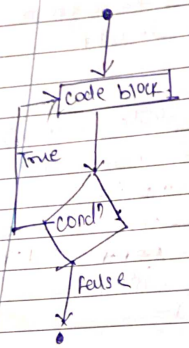



```
# include<stdio.h>
Void main()
{ int i = 1;
  while (i<= 10)
  { printf ("%d \n", i)
    i++;
  }
}
```

2] do-while loop

Unlike for & while loops, which test the loop condition at the top of the loop, the do-while loop in C programming checks its condition at the bottom of the loop. A do-wh loop is similar to a while loop, except the fact that do-while would execute its statements at least once, even if the condition fails for the first time. The while, on other hand will not execute its statements if the condition fails for first time. Syntax:

```
do
{ statements;
} while (condition);
```



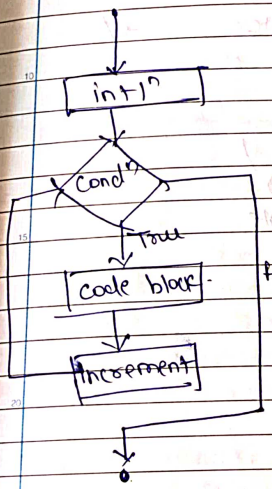

```
# include <stdio.h>
int main()
{  int a=10;
   do
   {
   pf ("value of a: %d \n", a);
   a = a+1;
   } while (a<20);
}
```

3] for loop

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. Syntax:

```
for (initialize counter; test counter; increment counter)
{
    statements;
}
```

- initialize counter : executed first and only once. This step allows to declare and initialize any loop control variables.
- test counter : if it is true, the body of loop is executed. If it is false, the body of loop does not execute and flow of control jumps to next statement just after 'for' loop.
- After the body of for loop executes, the flow of control jumps back to increment counter. This statement can be left blank, as long as semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes & process repeats itself. If it is false, for loop terminates




```
# include <stdio.h>
int main()
{ int a;
  for (a=10; a<20; a++)
  {
    pf ("value of a : %d
        \n", a);
  }
  return 0;
}
```

4] Nesting of loops

```
# include < stdio.h>
void main()
{ int n;
  pf (" Enter value of n: ");
  sf ("%d", &n);
  for (int i=1; i<=n; i++)
  { for (int j=1; j<=10; j++)
    {   printf ("%d\t", (i*j));
    } pf ("\n"); } }
```

* General structure of C program

- Documentation section
- Preprocessor commands
- Functions
- Variables
- Statements & Expressions
- Comments.

```
/* Developed by: ---- Date: ---- */
#include <stdio.h>          <- preprocessor commands
void main()
{
  /* My first C prog */
  pf("\n Hello world");
  // single line comments
  /* multiple
     line
     comments */
}
```

- Rules in C programming

- Execution of every C program starts with main() function.
- Each C program contains only one main() function
- Every statement in program ends with semicolon
- Each opening braces should have its closing braces
- C can contain comments anywhere in program.

- Header files in C

Header file contain the library functions which are required by program to execute it properly. The #include <file_name> is used to include header files.
This statement is always written at beginning of program.
stdio.h - standard input output operation
conio.h - to deal with console related operations.
math.h - to perform mathematical operations

- main() function.

- Every program must have main() function
- main() function is entry point of program execution. It is not possible to run program without main() function.
- The body of main() function is written within { and } braces.

- Documentation Section.

Contains information about program and its author which is provided in comments.

- Preprocessor commands

Tells a C compiler to include required library files. These files are included before compiling the code.

- Function Section

C is block structured language. It uses functions to divide code into blocks.
Below main() fun other user defined fun can be defined.

- Declaration Section
- Statements & expressions
- Comments

are ignored by compiler & are used to add additional information about program or its statements.

* Scanf() function.

This function is used to accept input from user.

This function reads input from stdin and then scans that input according to provided format

Syntax:

scanf("format specifier", &variable-name);

format specifier - which type of data should be accepted from user

%c read single character from user
%d read a decimal integer from user
%e/%f/%g - read a floating point value from user
%h - reads a short integer
%i - reads decimal, hexadecimal or octal integer from user.
%s - read a string from user
%u - unsigned decimal integer

* Arrays.

- An array is a finite collection of similar elements stored in adjacency memory locations
  or
- Array is a collection of variables of the same type that are referred by a common name.
- An array with n number of elements is referenced using an index that range from 0 to n-1. The lower index of an array is called its lower bound and highest index is called the upper bound.
- The elements of an array arr[n] containing n elements are referenced as
  arr[0], arr[1], ... arr[n-1]
  0 - lower bound , n-1 upper bound.

- Declaration of array

An array can be declared just as any other variable in C i.e. ~~data~~ data type followed by array name. The only addition in the declaration is the subscript in bracket which indicates the number of elements it will hold.
ex.

```
int age[20];
float sal[10];
char grade[10];
```

- The elements of an array can be easily processed as they are stored in contiguous memory location.
  For ex int arr[5]
  This is stored as

100	102	104	106	108
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]

ex
```
# include <stdio.h>
void main()
{ int arr[10], i;
  for (i=0; i<10; i++)
  {
    printf("Enter the %d number \n", i+1);
    scanf("%d", &arr[i]);
  }
  for (i=0; i<10; i++)
    printf("number %d is %d \n", i+1, arr[i]);
}
```

- Array Initialization.
  Arrays can be initialized at the time of declaration. For example,
  int age[5] = {8, 10, 5, 15, 20};

Age →	[0]	[1]	[2]	[3]	[4]
Value →	8	10	5	15	20
Address →	100	102	104	106	108

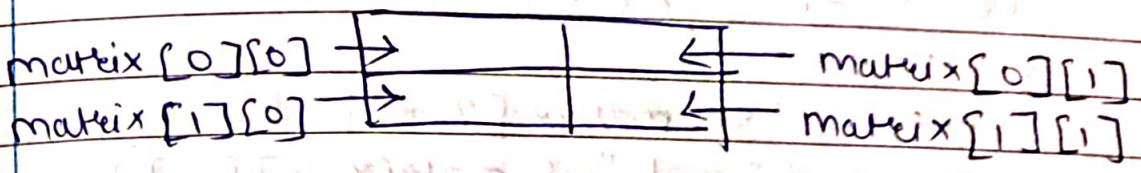
Syntax

data_type array-name [row size] [column size];

ex

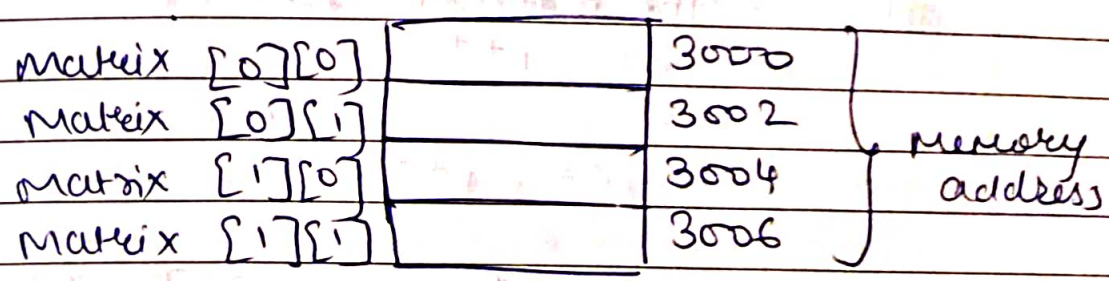
int matrix [2][2];

2×2 matrix stores 4 values



But in memory it will form different structure

Memory arrangement of 2D array



Syntax for initialization of 2D array

array_name [row_number] [column_number] = value

ex

matrix [1][1] = 9; matrix[0][0] = 7;

Matrix [0][1] = 6; matrix [1][0] = 5;

This array stores 2×2×2 = 8 values

Syntax for initializing multidimensional array is :

array-name [row-no] [row-no] [column-no] = value

- Entering data in multidimensional array

```
pf ("\n Enter data : \n");
for (i=0 ; i<2 ; i++)
{
   for (j=0 ; j<2 ; j++)
  {
     for (k=0 ; k<2 ; k++)
     {
        pf ("\n array [%d][%d][%d] : ",
                    i,j,k);
        sf ("%d", &array[i][j][k]);
     }
  }
}
```

* Functions.

- A block of code which is intended to perform a specific task is called as function.
- A C program is a collection of one or more functions.
- If a C program contains only one function it must be main().
- If C program contains more than one function, then one of these functions must be main()
- Each function in a program is called in the sequence specified by the function calls in main()
- After each function has done its thing control returns to main(). When main() runs out of statements & function calls, the program ends

- Definition of function.

Elements of function - 6

- Function header (return type, function name, formal parameters)
- Function body (local variables declaration, function body / statements, return statement)

1) return_type : The return type is a data type of value which the function returns.
2) Function_name :
3) List of formal parameters
   it is optional
   It holds the values sent by calling function.

The list consists of type of parameters and name of parameters &
Function Signature is formed by the function name and list of formal parameters. Here order, type & number of formal parameter matters.

4) Function body :
   Set of statements that define operation of function. It is enclosed in { and }
   It consists 3 parts :
   - Declaration of local variable
   - Executable statements
   - Return values

Syntax of function

```
return-type function_name (list of formal param)
{
    local variable declaration;
    Statement 1;
    Statement 2;
    —
    —
    return statement;
}
```

Ex

```
int add (int a, int b)
{  int result
   result = a+b;
   return result;
}
```

- Declaration of function.

Before using function in program it should be declared.
The declaration of function is called as function prototyping

return_type function_name (list of formal parameters);

It is possible to have different parameter names in function prototype from the names having in function difinition.
But it is mandatory to match return type in function prototype with return type in function difinition.
function declaration is done in global section i.e. after including the headers files and above all functions including main()

- Function call.

To use defined function we must call it.
Syntax
function_name (list of actual parameters);

Before calling any function it should be either declared or difined first and then function call can be made, otherwise compiler will generate an error.

Camlin

- calling function and called function
The function which calls another function is referred as calling function and the function which gets called is referred as called function.

- Actual & formal parameters

Parameters used in function call are referred as actual parameters whereas parameters used in function definition & function declaration referred as formal parameters.

```
#include<stdio.h>
void greater (int, int); → fun prototype
void main()  → calling fun
{
    int x=4, y=5;
    greater(x,y); → fun call
}
void greater (int a, int b) → called fun
{
    if (a>b)
    {
    pf ("\n %d is greater number", a);
    }
    else
    {
    pf ("\n %d is greater number", b);
    }
}
```

- Parameters in function call.

Methods of parameter passing
- call by value
- Call by reference.

call by value:
The values of actual parameters are copied to function's formal parameters. In other words, the value of variable is used in function call
- There are 2 copy of parameters stored in different memory locations
- One is original copy & other is function copy.
- Any changes made inside functions are not reflected in actual parameters of caller

```
# include <stdio.h>
void swap (int x, int y);
void main()
{ int a=10, b=30;
  swap(a,b);
  pf ("In main: a=%d b=%d \n", a, b);
}
void swap (int x, int y)
{ int t;
  t = x;
  x = y;
  y = t;
pf ("Inside fun: x=%d y=%d \n", x, y);
}
```

Thus actual value of a & b remain unchanged even after exchanging.

call by reference.
The address of actual parameters is passed to the function as formal parameters.
- Both the actual & formal parameters refer to same locations.
- Any changes made inside function are actually reflected in actual parameters of caller

```
# include <stdio.h>
void swap (int *, int *);
void main()
{ int a=10, b=20;
  swap (&a, &b);
  printf ("Inside caller: a=%d
           b=%d", a, b);
}
void swap (int *x, int *y)
{ int t;
  t = *x;
  *x = *y;
  *y = t;
pf ("Inside function: a=%d, b=%d",
    *x, *y);
}
```

Thus actual value of a & b get changed after exchanging.

call by value	call by reference
1) While calling a function, we pass the values of variables to it	While calling a function, instead of passing values of variables, we pass the address of variables
2) In this method, the value of each variable in the calling function is copied into corresponding dummy variables of the called function	In this method, the address of ~~local~~ actual variables in the calling function is copied into the dummy variables of called function.
3) With this method, the changes made to the dummy variables in the called function have no effect on the values of actual variables in the calling function	With this method, using addresses we would have access to the actual variables & hence we would be able to manipulate them.
4) In call-by-values, we cannot alter the values of actual variables through function calls.	In call by reference, we can alter the values of variables through function call.
5) Values of variables are passed by simple	~~pointers~~ variables are necessary to define to store the address values of variables.
6) This method is preferred when we have to pass some small values that should not change	This method is preferred when we have to pass a large

Advantages of Functions

1) We can avoid rewriting same logic/code again & again.
2) We can call a function any number of times & any place in program.
3) We can track large C program easily when it is divided into multiple functions.
4) As huge code is divided into functions it is easy to read, debug, test, maintain & modify function individually.
5) C allows us to use library functions & also allows extending the standard library functions by adding user defined functions into C library so that the functions can be used by other program.

Scope rules

Every variable has its scope, means it can be used only by the block in which it is declared. Other codes can't access that variable. This is referred as scope & lifetime of that variable in program.

Types of variables

- Global variable : Scope is throughout prog. declared outside all functions (including main())
- Local variable : Scope is limited to function in which it is declared

Camlin

```
# include <stdio.h>
void display (int a);
int a = 50;          → global
void main()
{
  int i = 20;        → local
  pf ("In main fun \n a = %d \n i = %d", a, i);
  display (i);
}
void display (int j)
{
  int k = 35;        → local
pf ("In display fun \n a = %d \n j = %d
        \n k = %d", a, j, k);
}
```

O/P

In main fun	In display fun
a = 50	a = 50
i = 20	j = 20
	k = 35

— Recursive function.

Process of calling a fun inside itself is called as recursion. The fun which call itself is called as recursive function.

```
# include <stdio.h>
int fact (int);
void main()
{  int n, f;
   pf ("\n Enter no: ");
   sf ("%d", &n);
   f = fact (n);
   pf ("Factorial value = %d", f);
}
int fact (int num)
{  int f;
   if (num == 1)
   {  return 1; }
   else
   {  f = num * fact (num-1);
   }
   return (f);
}
```

* Pointers.

Pointer is a variable which is used to store memory address of another variable.

```
# include <stdio.h>
Void main()
{ int var1;
  char var2[10];
  pf ("Addr of var1 %x \n", &var1);
  pf ("Addr of var2 %x \n", &var2);
}
```

Like any variable or constant, you must declare a pointer before using it to store any variable address.

- Advantages of pointers.
  - Pointers are used for dynamic memory allocation & deallocation.
  - An array of a structure can be accessed efficiently with pointers.
  - Pointers are useful for accessing memory locations.
  - Pointers reduce length & complexity of programs.
  - They increase the execution speed & thus reduce the program execution time.
  - Pointers provide efficient tool for manipulating dynamic data structures such as structures, linked list, queue, stacks & trees.

- Declaring pointer variables.

In the program pointers are declared at some location when we declare variables i.e. in declaration section.

Syntax

data_type * pointer_name;

This tells the compiler 3 things about the variable pointer_name.

1) * asterik tells that it is a pointer variable.
2) It points to memory location.
3) It points to a variable of type data type.

for ex. int *p
float *x.
char *cptr.

- Initialization of pointer

Assigning the address of certain variable to pointer is called as pointer initialization.

Syntax:

pointer_name = &variable_name;

& (address of operator) is used to find out memory location of a variable.

ex
```
int *ptr;
int a;
ptr = &a;
```

— Rules of pointers

1) A pointer variable can be assigned the address of another variable.
2) A pointer variable can be assigned the values of another pointer variable.
3) A pointer variable can be initialized with NULL or zero value.
4) A pointer variable can be prefixed or post fixed with increment or decrement operators.
5) A integer value may be added or subtracted from a pointer variable.
6) A ponter variable cannot be multiply by a constant.
7) Two pointer variables cannot be added
8) Two pointer variables cannot be multiplied

```
#include <stdio.h>
void main()
{  int no=50;
   int *p;
   p=&no;
printf("value stored in variable no: %d", no);
printf("\n address of variable no: %d", &no);
printf("\n value stored in pointer ip: %d", p);
printf("\n value pointed by pointer
        *P is: %d", *p);
}
```

* String Manipulation.

An array with character (char) data type is called as character array.
character array forms a string.
string is collection of characters, numbers & special symbols.
At the end of character array or string '\0' (NULL value) is stored.

— Declaration of array of characters.
syntax
```
char array_name[size];
```
example
```
char array[6];
```

— Initialization of array of character
syntax:
```
array-name[subscript]=value;
```

```
#include <stdio.h>
void main()
{
  char array[10];
  int i;
  printf("\n Enter 10 characters");
  for(i=0; i<10; i++)
     scanf("%c", &array[i]);
  printf("\n Entered array is");
  for(i=0; i<10; i++)
     printf("%c", array[i]);
}
```

```c
#include <stdio.h>
void main()
{ int l=0, i=0;
  char str[10];
  int flag = 0;
  printf ("Enter string: ");
  scanf ("%s", str);
  while (str[l] != NULL)
  {  l++; }
  l--;
  while (i<l)
  {
    if (str[i] == str[l])
    {
        flag = 1;
    }
    else
    {  flag = 2;
       break;
    }
    i++;
    l--;
  }
  if (flag == 1)
  {
    printf ("%s is palindrome", str);
  }
  else
  {
  printf ("%s is not palindrome", str);
  }
}
```

```c
#include <stdio.h>
void main()
{
   strcopy();
   strlength();
}
void strcopy()
{
   char str1[20], str2[20];
   int i;
   printf ("\n Enter a string:");
   gets (str1);
   for (i=0; str1[i] != 0; i++)
   {
      str2[i] = str1[i];
   }
   printf ("\n copied string is %s", str2);
}
void strlength()
{
   char str1[20];
   int i;
   printf ("\n Enter a string:");
   gets (str1);
   for (i=0; str1[i] != 0; i++)
   {
   printf ("\n length is: %d", i);
}
```

To access variables, we have to declare a variable of this structure type.

ex.

```
struct Student
{   int roll_no;
    char name[25];
    float percentage;
} s;
```

Or `struct Student s;`

s is structure variable.

To access & initialize the structure members:

`struct_variable.member_name;`

ex.

```
s.roll_no = 1;
s.name = 'Neha';
s.percentage = 75.20;
```

— Structure within structure.

```
struct product
{   char p_name[40];
    int p_id;
    float price;
    int qty;
    struct mdate
    {   int date;
        int month;
        int year;
    } dom;
} p;
```

```
p.p_name = 'Harddisk';
p.p_id = 1001;
p.price = 1000;
p.qty = 2;
p.mdate.date = 12;
p.mdate.month = 4
p.mdate.year = 1995;
```

— Array of Structure

```
Struct Student
{ int rollno;
  char name[25];
  float percentage;
} s[5];
```
↳ will hold ~~reco~~ details of 5 books

— Array Vs Structure

Array	Structure
1) An array is a collection that consists of elements of same data type	A structure is a collection that consists of elements of different data types
2) Array is declared using []	Struct is declared using Struct keyword
3) Array uses subscripts or [] to access elements	Structure uses the '.' (dot operator) to access element.
4) The size of array is fixed	Size of a structure is not fixed
5) Traversing through and searching for elements in an array is quick & easy.	Traversing through & searching for element in a structure is slow & complex.
6) An array is always stored in contiguous memory locations	A structure may or may not be stored in contiguous memory locations.

* Union

syntax

```
Union tag-name
{
  datatype & variable1;
  datatype variable2;
  datatype variable3;
};
```

example

```
Union Student
{
  int marks;
  char name[10];
  float average;
};
```

```
#include <stdio.h>
#include <string.h>
union student
{ char name[20];
  char subject[20];
  float percentage;
};
void main()
{ union student record1;
  union student record2;
  strcpy(record1.name, 'Raju');
  strcpy(record1.subject, 'Maths');
  record1.percentage = 86.50;
  printf("Name : %s\n", record1.name);
  printf("Subject : %s\n", record1.subject);
  printf("percentage : %.2f\n", record1.percentage);
}
```

Camlin

— Union vs Structure.

Union	Structure
1) The keyword union is used to define a union	The keyword struct is used to define a structure
2) When a variable is associated with a union, the compiler allocates the memory by considering the size of largest memory. So, size of union is equal to size of largest member.	When a variable is associated with structure, the compiler allocates the memory for each member. The size of structure is greater than or equal to sum of sizes of its members
3) Memory allocated is shared by individual members of union	Each member within structure is assigned unique storage area
4) Altering the value of any of the member will alter other member values	Altering the value of member will not affect other members of structure
5) Only one member can be accessed at a time.	Individual member can be accessed at a time
6) Only first member of union can be initialized	Several members of structure can be initialize at once.

Camlin

* ENUM (Enumeration)

Enum is user defined
It is suitable in situation when we already know finite list of values that a data type can take.
The values for enum cannot be input by user or output on screen.

ex
```
enum months { jan, feb, mar, apr, may };
enum days { sun, mon, tue, wed, thu };
enum rys { bicycle, scooter, car };
```

```
# include <stdio.h>
enum months { jan, feb, mar, apr, may };

void main()
{ months m1, m2;
  m1 = jan;
  m2 = apr;
  int diff = m2 - m1;
  printf(" Months between %d ", diff);
  if (m1 > m2)
  printf (" \n m2 comes before m1");
}
```

Here jan=0, feb=1, mar=2, apr=3, may=4
Or
enum months { jan=1, feb, mar, apr, may };
Value of next element in the list is previous value plus one.

Camlin

MACROS.

- MACRO is nothing but a piece of code which has a particular name known as macro name. Whenever there is use of macro name, C preprocessor replaces it with the definition (value) of the macro.
- Two types of Macros

1) Object-like macros / simple macros

```
#define macro-name macro-body
```

macro-body is token sequence that macro name stands for. The macro body is also known as replacement list or expansion.

ex:

```
#define MAX-size 1000

int list[MAX-size];
```

The C preprocessor will replace the MAX-size by 1000 as

```
int list[1000];
```

```
#define RGB-COLORS red, \
    green, \
    blue
enum RGB { RGB-COLORS };
```

Multiple line MACRO

```
#include <stdio.h>
#define PI 3.1415
Void main()
{
    float radius, area;
    printf("Enter radius");
    scanf("%d", &radius);

    area = PI * radius * radius;
    printf("Area = %.2f", area);
}
```

2) Function like MACROS / complex MACRO or macros with arguments.

```
#include <stdio.h>
#define PI 3.1415
#define circlearea(r) (PI*r*r)
Void main()
{   int radius;
    float area;
    printf("Enter radius");
    scanf("%d", &radius);
    area = circlearea(radius);
    printf("Area = %d", area);
}
```

* File Handling in C :

- File helps us to store the program output permanently which can be accessed anytime in future
- Traditional file processing system in a computer based system in which all the information is stored in various computer files.

– Field
A field is an item of stored data. A field could be name, date, an address, etc.
A field has name (identifier) & a type (defines type of data that will be stored in field).

– Record.
A record is the collection of fields that relate to a single entity.
ex Student records with fields student's name, address, date of birth, etc.

– File.
A file is a collection of related records.
ex. Student file might include all of the records of students enrolled at school.
Files are stored on secondary storage devices such as hard disks, CD-ROMs

- Opening or creating file
fopen() function used with required access modes like:

"r", "w", "a", "r+", "w+"
"a+"

For binary file, append 'b' at last.
For ex instead of "w" use "wb"

For performing operations on file, a special pointer called file pointer is used which is declared as:

```
FILE *fp;
fp = fopen ("filename.txt", "w")
```

- Reading from a file.

i) fgetc (file_pointer) : reads character by character
ii) fgets (str, no-of-chars, file-pointer) : reads data string by string

iii) fscanf (FILE *stream, const char * format-string argument-list)
: read set of characters from file.

- Writing a file.
i) fputc (ch, file-pointer) : writes char by char
ii) fputs (str, file-pointer) : write string in file
iii) int fprintf (FILE * stream, const char *format-string [, arguments ...])

Page :
Date :