Contents :

Basics of C

by
H. Faheem Ahmed

WHAT IS C ?

C language is a general purpose and structured pragramming langauge developed by 'Dennis Ritchie' at AT &T's Bell Laboratories in 1972 in USA. It is also called as 'Procedure oriented programming language.'

The C language has following numerous features as:

- Easy to learn

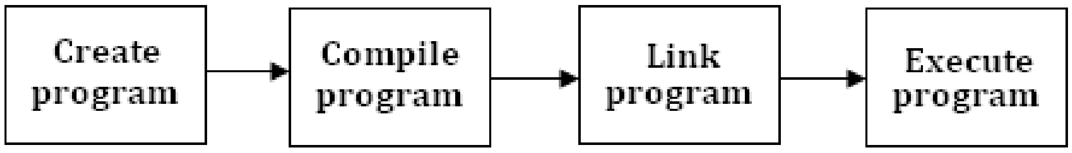
- Structured language
- It produces efficient programs.
- It can handle low-level activities.
- It can be compiled on a variety of computers.

Facts about C

- C was invented to write an operating system called UNIX.
- C is a successor of B language which was introduced around 1970
- The language was formalized in 1988 by the American National Standard Institue (ANSI).
- By 1973 UNIX OS almost totally written in C.
- Today C is the most widely used System Programming Language.
- Most of the state of the art software have been implemented using C

Execution of C Program :

C program executes in following 4 (four steps).



1. Creating a program :

An editor like notepad or wordpad is used to create a C program. This file contains a source code which consists of executable code. The file should be saved as **'*.c'** extension only.

2. Compiling the program :

The next step is to compile the program. The code is compiled by using compiler. Compiler converts executable code to binary code i.e. object code.

3. Linking a program to library :

The object code of a program is linked with libraries that are needed for execution of a program. The linker is used to link the program with libraries. It creates a file with **'*.exe'** extension.

4. Execution of program :

The final executable file is then run by dos command prompt or by any other software.

Structure of C Program: The basic **structure of C program** is as follow:

document section
links section

```
definition section
global declaration section

void main()
{
variable declaration section
function declaration section
executable statements;
}

function definition 1
{         --------------------
          --------------------
}

function definition n
{         --------------------
          --------------------
}
```

where,

Document Section : It consists of comment lines which include name of a program, author name, creation date and other information.

Links Section (File) : It is used to link the required system libraries or header files to excute a program.

Definition Section : It is used to define or set values to variables.

Global variable declaration Section : It is used to declare global or public variable.

void main() : Used to start of actual C program. It includes two parts as declaration part and executable part.

Variable declaration section : Used to declare private variable.

Function declaration section : Used to declare functions of program from which we get required output.

Then, executable statements are placed for execution.

Function definition section : Used to define functions which are to be called from main().

Example:

```
/* First C-program */
#include <stdio.h>
int a,b;
main()
{         int i,j;
          printf( “\n%d %d”,i,j);
}
```

Constants in C :

A constant is an entity that doesn't change during the execution of a program. Followings are the different types of constants : Example: const int a=20;

• Octal constants are written with a leading zero	015
• Hexadecimal constants are written with a leading 0x	0x15
• Long constants are written with a trailing L	15L.

1. Real Constant :

- It must have at least one digit.
- It must have a decimal point which may be positive or negative.
- Use of blank space and comma is not allowed between real constants.
- Example: +194.143, -416.41

2. Integer Constant :

- It must have at least one digit.
- It should not contain a decimal place.
- It can be positive or negative.
- Use of blank space and comma is not allowed between real constants.
- Example: 1990, 194, -394

3. Character Constant :

- It is a single alphabet or a digit or a special symbol enclosed in a single quote.
- Maximum length of a character constant is 1.
- Example: 'T', '9', '$'

4. String Constant :

- It is collection of characters enclosed in double quotes.
- It may contain letters, digits, special characters and blank space.
- Example: "Islamiah College, Vaniyambadi."

Character Set :

A character refers to the digit, alphabet or special symbol used to data represetation.

1. Alphabets : A-Z, a-z
2. Digits : 0-9
3. Special Characters : ~ ! @ # $ % ^ & * ( ) _ + { } [ ] - < > , . / ? \ | : ; " '
4. White Spaces : Horizontal tab, Carriage return, New line, form feed

Identifier :

Identifier is the name of a variable that is made up from combination of alphabets, digits and underscore.

Variable :

Variable is a name of memory location where we can store any data. It can store only single data (Latest data) at a time. In C, a variable must be declared before it can be used. Variables can be declared at the start of any block of code, but most are found at the start of each function. . It is opposite to constant.

Rules for varibales:

- First character should be letter or alphabet.
- Keywords are not allowed to use as a variable name.
- White space is not allowed.
- C is case sensitive i.e. UPPER and lower case are significant.
- Only underscore, special symbol is allowed between two characters.
- The length of indentifier may be upto 31 characters but only only the first 8 characters are significant by compiler.

Local Variables

Local variables are declared within the body of a function, and can only be used within that function only.

```
void main( )
{         int a,b,c;
}
void fun1()
{         int x,y,z;
}
```

Here a,b,c are the local variable of void main() function and it can’t be used within fun1() function. And x, y and z are local variable of fun1().

Global Variabless

A global variable declaration looks normal, but is located outside any of the program's functions. This is usually done at the beginning of the program file, but after preprocessor directives.

```
int a,b,c;
void main()
{
}
void fun1()
{
}
```

Here a,b,c are global variables, and these variable can be accessed (used) within a whole program.

Keywords :

Keywords are the system defined identifiers.

All keywords have fixed meanings that do not change.

White spaces are not allowed in keywords.

Keyword may not be used as an identifier.

It is strongly recommended that keywords should be in lower case letters.

There are totally **32(Thirty Two) keywords** used in a C programming.

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

Escape Sequence Characters (Backslash Character Constants) in C:

C supports some special escape sequence characters that are used to do special tasks.

These are also called as 'Backslash characters'. Some of the escape sequence characters are as follow:

\n - New Line \b - Backspace \t - Horizontal tab \f - Form feed

Data Types in C :

When we use a variable in a program then we have to mention the type of data. This can be handled using data type in C. Followings are the most commonly used data types in C.

Keyword	**Format Specifier**	**Size**	**Data Range**
char	%c	1 Byte	-128 to +127
int	%d	2 Bytes	-32768 to +32767
long int	%ld	4 Bytes	-2^{31} to $+2^{31}$
unsigned int	%u	2 Bytes	0 to 65535

float	%f	4 Bytes	$-3.4e^{38}$ to $+3.4e^{38}$
double	%lf	8 Bytes	$-1.7e^{38}$ to $+1.7e^{38}$
long double	%Lf	12-16 Bytes	$-3.4e^{38}$ to $+3.4e^{38}$

Qualifier : When qualifier is applied to the data type then it changes its size or its sign.

Size qualifiers : **short, long**

Sign qualifiers : **signed, unsigned**

Enum Data Type :

This is an user defined data type having finite set of enumeration constants. The keyword 'enum' is used to create enumerated data type.

Syntax: enum [data_type] {const1, const2, ..., const n};

Example: enum mca{ software, web, seo};

enum days{sun,mon,tue,wed,thu,fri,sat};

Typedef : It is used to create new data type. But it is commonly used to change existing data type with another name.

Syntax: typedef [data_type] synonym;

Example: typedef int integer;

integer rno;

Operators in C :

Operator is a symbol that is used to perform mathematical operations.

Operator Name	**Operators**
Assignment	=
Arithmetic	+, -, *, /, %
Logical	&&, ||, !
Relational	<, >, <=, >=, ==, !=
Shorthand	+=, -=, *=, /=, %=

Unary	++, --
Conditional	() ? : ;
Bitwise	&, |, ^, <<, >>, ~

Arithmetic Operators:

```
main()
{           int a = 21;
            int b = 10;
            int c ;
            c = a + b;//c is 31
            c = a - b; //c is 11
            c = a * b;          //c is 210
            c = a / b; //c is 2
            c = a % b;          //c is 1
            c = a++;  //c is 21
            c = a--;   //c is 22
}
```

Bitwise Operators: Bitwise operator works on bits and perform bit by bit operation.

Assume if A = 60; and B = 13; Now in binary format they will be as follows:

```
            A       =       0011 1100

            B       =       0000 1101

Then        A&B     =       0000 1100               //and

            A|B     =       0011 1101               //or

            A^B     =       0011 0001               //xor

            ~A      =       1100 0011               //complement
```

Example:

```
main()
{           unsigned int a = 60;            /* 60 = 0011 1100 */
            unsigned int b = 13;            /* 13 = 0000 1101 */
            int c = 0;
            c = a & b;                      /* 12 = 0000 1100 */
            c = a | b;                      /* 61 = 0011 1101 */
            c = a ^ b;                      /* 49 = 0011 0001 */
            c = ~a;                         /*-61 = 1100 0011 */
            c = a << 2;                     /* 240 = 1111 0000 */
            c = a >> 2;                     /* 15 = 0000 1111 */
}
```

Conditional Operator: Syntax: condition ? expression1 : expression2 ;
if condition is true, then expression1 is executed else expression 2 is executed.
Example:

```
main()
{           int a , b;
```

```
        a = 10;
        b = (a == 1) ? 20: 30;
        printf( "Value of b is %d\n", b );
        b = (a == 10) ? 20: 30;
        printf( "Value of b is %d\n", b );
}
Value of b is 30
Value of b is 20
```

Precedence of C Operators:

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedenace than + so it first get multiplied with 3*2 and then adds into 7.

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

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

Decision Making Statements / Conditional Statements :

C program executes program sequentially. Sometimes, a program requires checking of certain conditions in program execution. C provides various key condition statements to check condition and execute statements according conditional criteria. 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

if Statement :

This is a conditional statement used in C to check condition or to control the flow of execution of statements. This is also called as 'decision making statement or control statement.' The execution of a whole program is done in one direction only.

Syntax:

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

In above syntax, the condition is checked first. If it is true, then the program control flow goes inside the braces and executes the block of statements associated with it. If it returns false, then program skips the braces. If there are more than 1 (one) statements in if statement then use { } braces else it is not necessary to use.

```
void main()
{           int a;
            a=5;
            clrscr();
            if(a>4)
                        printf("\nValue of A is greater than 4 !");
            if(a==4)
                        printf("\n\n Value of A is 4 !");
}
```

if-else Statement :

This is also one of the most useful conditional statement used in C to check conditions.

Syntax:

```
if(condition)
{           true statements;
}
else
{           false statements;
}
```

In above syntax, the condition is checked first. If it is true, then the program control flow goes inside the braces and executes the block of statements associated with it. If it returns false, then it executes the else part of a program.

```
void main()
{           int no;
            printf("\n Enter Number :");
            scanf("%d",&no);
            if(no%2==0)
                        printf("\n\n Number is even !");
            else
                        printf("\n\n Number is odd !");
}
Enter Number : 11
Number is odd !
```

Nested if-else Statement :

It is a conditional statement which is used when we want to check more than 1 conditions at a time in a same program. The conditions are executed from top to bottom checking each condition whether it meets the conditional criteria or not. If it found the condition is true then it executes the block of associated statements of true part else it goes to next condition to execute.

Syntax:

```
if(condition)
{           if(condition)
            {           statements;
            }
            else
            {           statements;
            }
}
else
{           statements;
}
```

In above syntax, the condition is checked first. If it is true, then the program control flow goes inside the braces and again checks the next condition. If it is true then it executes the block of statements associated with it else executes else part.

```
void main()
{           int no;
            printf("\n Enter Number :");
            scanf("%d",&no);
            if(no>0)
            {           printf("\n\n Number is greater than 0 !");
            }
            else
```

```
          {       if(no==0)
                  {       printf("\n\n It is 0 !");
                  }
                  else
                  {       printf("Number is less than 0 !");
                  }
          }
}
```

```
Enter Number : 0
It is 0 !
```

switch case Statement :

This is a multiple or multiway brancing decision making statement. When we use nested if-else statement to check more than 1 conditions then the complexity of a program increases in case of a lot of conditions. Thus, the program is difficult to read and maintain. So to overcome this problem, C provides 'switch case'.

Switch case checks the value of a expression against a case values, if condition matches the case values then the control is transferred to that point.

Syntax:

```
                  switch(expression)
                  {
                          case expr1:
                                  statements;
                                  break;
                          case expr2:
                                  statements;
                                  break;



                                  . . . . .
                                  . . . . .



                                  . . . . .
                          case exprn:
                                  statements;
                                  break;
                          default:
                                  statements;
                  }
```

In above syntax, switch, case, break are keywords.

expr1, expr2 are known as 'case labels.'

Statements inside case expression need not to be closed in braces.

break statement causes an exit from switch statement.

default case is optional case. When neither any match found, it executes.

```
void main()
{           int no;
            printf("\n Enter any number from 1 to 3 :");
            scanf("%d",&no);
            switch(no)
            {           case 1:
                                    printf("\n\n It is 1 !");
                                    break;
                        case 2:
                                    printf("\n\n It is 2 !");
                                    break;
                        case 3:
                                    printf("\n\n It is 3 !");
                                    break;
                        default:
                                    printf("\n\n Invalid number !");
            }
}
```

```
Enter any number from 1 to 3 : 3
It is 3 !
```

```
Enter any number from 1 to 3 : 5
Invalid number !
```

Rules for declaring switch case :

- The case label should be integer or character constant.
- Each compound statement of a switch case should contain break statement to exit from case.
- case labels must end with (:) colon.

Advantages of switch case :

- Easy to use.
- Easy to find out errors.
- Debugging is made easy in switch case.
- Complexity of a program is minimized.

Looping Statements / Iterative Statements :

'A loop' is a part of code of a program which is executed repeatedly.

A loop is used using condition. The repetition is done until condition becomes true.

A loop declaration and execution can be done in following ways.

- Check condition to start a loop
- Initialize loop with declaring a variable.
- Executing statements inside loop.
- Increment or decrement of value of a variable.

Types of looping statements :

Basically, the types of looping statements depends on the condition checking mode. Condition checking can be made in two ways as : Before loop and after loop. So, there are 2(two) types of looping statements.

- Entry controlled loop
- Exit controlled loop

1. Entry controlled loop :

In such type of loop, the test condition is checked first before the loop is executed.

Some common examples of this looping statements are :

- **while loop**
- **for loop**

2. Exit controlled loop :

In such type of loop, the loop is executed first. Then condition is checked after block of statements are executed. The loop executed atleat one time compulsarily.

Some common example of this looping statement is :

- **do-while loop**

While loop :

This is an entry controlled looping statement. It is used to repeat a block of statements as long as the condition is true.

Syntax:

```
while(condition)
{           statements;
            increment/decrement;
}
```

In above syntax, the condition is checked first. If it is true, then the program control flow goes inside the loop and executes the block of statements associated with it. At the end of loop increment or decrement is done to change in variable value. This process continues until test condition satisfies.

```
void main()
{           int a;
            a=1;
            while(a<=5)
            {           printf("\n Hello");
                        a+=1      // i.e. a = a + 1
            }
}
 Hello
 Hello
 Hello
 Hello
 Hello
```

For loop :

This is an entry controlled looping statement.

In this loop structure, more than one variable can be initilized. One of the most important feature of this loop is that the three actions can be taken at a time like variable initialization, condition checking and increment/decrement. The for loop can be more concise and flexible than that of while and do-while loops.

Syntax:

```
for(initialisation; test-condition; incre/decre)
{
            statements;
}
```

In above syntax, the given three expressions are seperated by ';' (Semicolon)

Features :

- More concise
- Easy to use
- Highly flexible
- More than one variable can be initilized.
- More than one increments can be applied.
- More than two conditions can be used.

```
void main()
{           int a;
            for(i=0; i<5; i++)
            {           printf("\n\t Hello");  // 5 times
            }
}
Hello
Hello
```

```
Hello
Hello
Hello
```

Do-While loop :

This is an exit controlled looping statement.

Sometimes, there is need to execute a block of statements first then to check condition. At that time such type of a loop is used. In this, block of statements are executed first and then condition is checked.

Syntax:

```
do
{           statements;
            (increment/decrement);
}while(condition);
```

In above syntax, the first the block of statements are executed. At the end of loop, while statement is executed. If the resultant condition is true then program control goes to evaluate the body of a loop once again. This process continues till condition is true. When it becomes false, then the loop terminates.

Note: The while statement should be terminated with ; (semicolon).

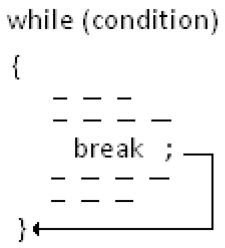
```
void main()
{           int a;
            a=1;
            do
            {           printf("\n\t Hello");  // 5 times
                        a+=1;      // i.e. a = a + 1
            }while(a<=5);
            a=6;
            do
            {           printf("\n\n\t Technowell");  // 1 time
                        a+=1;      // i.e. a = a + 1
            }while(a<=5);
}
```

```
 Hello
 Hello
 Hello
 Hello
 Hello

 Technowell
```

Break Statement :

Sometimes, it is necessary to exit immediately from a loop as soon as the condition is satisfied. When break statement is used inside a loop, then it can cause to terminate from a loop. The statements after break statement are skipped.

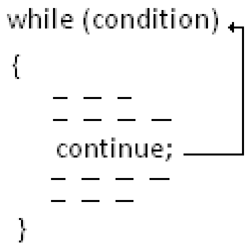


```
void main()
{         int i;
          for(i=1;  ; i++)
          {         if(i>5)
                    break;
                    printf("%d",i);  // 5 times only
          }
}
12345
```

Continue Statement :

Sometimes, it is required to skip a part of a body of loop under specific conditions. So, C supports 'continue' statement to overcome this anomaly.

The working structure of 'continue' is similar as that of that break statement but difference is that it cannot terminate the loop. It causes the loop to be continued with next iteration after skipping statements in between. Continue statement simply skips statements and continues next iteration.



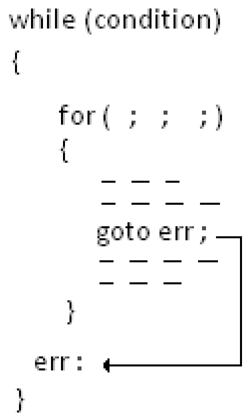
```
void main()
{         int i;
          for(i=1; i<=10; i++)
          {         if(i==6)
                    continue;
                    printf("\n\t %d",i);  // 6 is omitted
          }
}
          1
          2
          3
          4
          5
          7
          8
          9
          10
```

Goto Statement :

It is a well known as 'jumping statement.' It is primarily used to transfer the control of execution to any place in a program. It is useful to provide branching within a loop.

When the loops are deeply nested at that if an error occurs then it is difficult to get exited from such loops. Simple break statement cannot work here properly. In this situations, goto statement is used.

Syntax : goto [expr];


```
while (condition)
{
    for ( ;  ;  ; )
    {
       _ _ _
       _ _ _ _
       goto err ;
       _ _ _ _
       _ _ _
    }
  err :
}
```

```
void main()
{           int i=1, j;
            while(i<=3)
            {           for(j=1; j<=3; j++)
                        {           printf(" * ");
                                    if(j==2)
                                    goto stop;
                        }
                        i = i + 1;
            }
            stop:
                        printf("\n\n Exited !");
}
```

```
 *  *

 Exited
```

Functions in C :

A function is a block of code that performs a specific task. C program does not execute the functions directly. It is required to invoke or call that functions. When a function is called in a program then program control goes to the function body and executes the statements in the function. Function prototype tells compiler that we are going to use a function which will have given name, return type and parameters.

Types of functions : There are two types of functions as:

1. Built in Functions **2. User Defined Functions**

1. Built in Functions :
These functions are also called as 'library functions'. These functions are provided by system. These functions are stored in library files. e.g.

scanf() printf() strcpy() strlwr() strcmp() strlen() strcat()

2. User Defined Functions :

The functions which are created by user for program are known as 'User defined functions'.

Functions with no arguments and no return values.

Functions with arguments and no return values.

Functions with arguments and return values.

```
void add(int x,int y)
{          int result;
           result = x+y;
           printf("Sum of %d and %d is %d.\n\n",x,y,result);
}
void main()
{          add(10,15);
           add(55,64);
           add(168,325);
}
```

Passing Parameters to a Function

There are two ways to pass parameters to a function:

- **Pass by Value:** mechanism is used when you don't want to change the value of passed paramters. When parameters are passed by value then functions in C create copies of the passed in variables and do required processing on these copied variables.
- **Pass by Reference** mechanism is used when you want a function to do the changes in passed parameters and reflect those changes back to the calling function. In this case only addresses of the variables are passed to a function so that function can work directly over the addresses.

Example : Pass by value

```
int main()
{          int a = 10;
           int b = 20;
           printf("Before: Value of a = %d and value of b = %d\n", a, b );
           swap( a, b );
           printf("After: Value of a = %d and value of b = %d\n", a, b );
}

void swap( int p1, int p2 )
{          int t;
           t = p2;
           p2 = p1;
           p1 = t;
           printf("Value of a (p1) = %d and value of b(p2) = %d\n", p1, p2 );
}
```

Output:

Here the values of a and b remain unchanged before calling swap function and after calling swap function.

```
Before: Value of a = 10 and value of b = 20
Value of a (p1) = 20 and value of b(p2) = 10
After: Value of a = 10 and value of b = 20
```

Example : Pass by reference

```
int main()
{          int a = 10;
           int b = 20;
           printf("Before: Value of a = %d and value of b = %d\n", a, b );
           swap( &a, &b );
           printf("After: Value of a = %d and value of b = %d\n", a, b );
}

void swap( int *p1, int *p2 )
{          int t;
           t = *p2;
           *p2 = *p1;
           *p1 = t;
           printf("Value of a (p1) = %d and value of b(p2) = %d\n", *p1, *p2 );
}
```

Output:

Here the values of a and b are changes after calling swap function.

```
Before: Value of a = 10 and value of b = 20
```

Value of a (p1) = 20 and value of b(p2) = 10
After: Value of a = 20 and value of b = 10

Advantages :

- It is easy to use.
- Debugging is more suitable for programs.
- It reduces the size of a program.
- It is easy to understand the actual logic of a program.
- Highly suited in case of large programs.
- By using functions in a program, it is possible to construct modular and structured programs.

Recursion (Recursive Function) :

A recursive function is one which calls itself.

Example:

Features :

- There should be at least one if statement used to terminate recursion.
- It does not contain any looping statements.

Advantages :

- It is easy to use.
- It represents compact programming strctures.

Disadvantages :

- It is slower than that of looping statements because each time function is called.

Storage Class :

'Storage' refers to the scope of a variable and memory allocated by compiler to store that variable. Scope of a variable is the boundary within which a varible can be used. Storage class defines the the scope and lifetime of a variable.

From the point view of C compiler, a variable name identifies physical location from a computer where varaible is stored. There are two memory locations in a computer system where variables are stored as : Memory and CPU Registers.

Functions of storage class :

To detemine the location of a variable where it is stored ?

Set initial value of a variable or if not specified then setting it to default value.

Defining scope of a variable.

To determine the life of a variable.

Types of Storage Classes :

Storage classes are categorised in 4 (four) types as,

- Automatic Storage Class
- Register Storage Class
- Static Storage Class
- External Storage Class

Automatic Storage Class :

- Keyword : auto
- Storage Location : Main memory
- Initial Value : Garbage Value
- Life : Control remains in a block where it is defined.
- Scope : Local to the block in which variable is declared.

Syntax :

```
auto [data_type] [variable_name];
```

Example :

```
auto int a;
```

```
#include <stdio.h>
#include <conio.h>
void main()
{
        auto int i=10;
        clrscr();
        {
                auto int i=20;
                printf("\n\t %d",i);
        }
        printf("\n\n\t %d",i);
        getch();
}
```

Output :

```
        20

        10_
```

Register Storage Class :

- Keyword : register
- Storage Location : CPU Register
- Initial Value : Garbage
- Life : Local to the block in which variable is declared.

- Scope : Local to the block.

Syntax :

    register [data_type] [variable_name];

Example :

    register int a;

When the calculations are done in CPU, then the value of variables are transferred from main memory to CPU. Calculations are done and the final result is sent back to main memory. This leads to slowing down of processes.

Register variables occur in CPU and value of that register variable is stored in a register within that CPU. Thus, it increases the resultant speed of operations. There is no waste of time, getting variables from memory and sending it to back again.

It is not applicable for arrays, structures or pointers.

It cannot not used with static or external storage class.

Unary and address of (&) cannot be used with these variables as explicitly or implicitly.

```
#include <stdio.h>
#include <conio.h>

void main()
{
        register int i=10;
        clrscr();
        {
                register int i=20;
                printf("\n\t %d",i);
        }
        printf("\n\n\t %d",i);
        getch();
}
```

Output :

```
        20

        10_
```

Static Storage Class :

- Keyword : static
- Storage Location : Main memory
- Initial Value : Zero and can be initialize once only.

- Life : depends on function calls and the whole application or program.
- Scope : Local to the block.

Syntax :

    static [data_type] [variable_name];

Example :

    static int a;

There are two types of static variables as :

a) Local Static Variable
b) Global Static Variable

Static storage class can be used only if we want the value of a variable to persist between different function calls.

```
#include <stdio.h>
#include <conio.h>

void main()
{
        int i;
        void incre(void);
        clrscr();
        for (i=0; i<3; i++)
        incre();
        getch();
}

void incre(void)
{
        int avar=1;
        static int svar=1;
        avar++;
        svar++;
        printf("\n\n Automatic variable value : %d",avar);
        printf("\t Static variable value : %d",svar);
}
```

Output :

```
Automatic variable value : 2	Static variable value : 2

Automatic variable value : 2	Static variable value : 3

Automatic variable value : 2	Static variable value : 4_
```

External Storage Class :

- Keyword : extern
- Storage Location : Main memory
- Initial Value : Zero
- Life : Until the program ends.
- Scope : Global to the program.

Syntax :

```
extern [data_type] [variable_name];
```

Example :

```
extern int a;
```

The variable access time is very fast as compared to other storage classes. But few registers are available for user programs.

The variables of this class can be referred to as 'global or external variables.' They are declared outside the functions and can be invoked at anywhere in a program.

```
#include <stdio.h>
#include <conio.h>

extern int i=10;
void main()
{
	int i=20;
	void show(void);
	clrscr();
	printf("\n\t %d",i);
	show();
	getch();
}
void show(void)
{
	printf("\n\n\t %d",i);
}
```

Output :

```
	20

	10_
```

Array :

Array is a collection of homogenous data stored under unique name. The values in an array is called as 'elements of an array.' These elements are accessed by numbers called as 'subscripts or index numbers.' Arrays may be of any variable type.

Array is also called as 'subscripted variable.'

Types of an Array :

1. One / Single Dimensional Array
2. Two Dimensional Array

Single / One Dimensional Array :

The array which is used to represent and store data in a linear form is called as 'single or one dimensional array.'

Syntax:

```
<data-type> <array_name> [size];
```

Example:

```
int a[3] = {2, 3, 5};
char ch[20] = "TechnoExam" ;
float stax[3] = {5003.23, 1940.32, 123.20} ;
```

Total Size (in Bytes):

total size = length of array * size of data type

In above example, a is an array of type integer which has storage size of 3 elements. The total size would be 3 * 2 = 6 bytes.

* Memory Allocation :

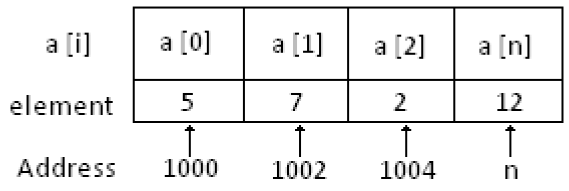


Fig : Memory allocation for one dimensional array

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int a[3], i;;
        clrscr();
        printf("\n\t Enter three numbers : ");
        for(i=0; i<3; i++)
        {
                scanf("%d", &a[i]);  // read array
        }
        printf("\n\n\t Numbers are : ");
        for(i=0; i<3; i++)
        {
                printf("\t %d", a[i]);  // print array
        }
        getch();
}
```

Output :

```
        Enter three numbers : 9 4 6

        Numbers are :     9         4         6_
```

Features :

- Array size should be positive number only.
- String array always terminates with null character ('\0').
- Array elements are countered from 0 to n-1.
- Useful for multiple reading of elements (numbers).

Disadvantages :

- There is no easy method to initialize large number of array elements.
- It is difficult to initialize selected elements.

Two Dimensional Array :

The array which is used to represent and store data in a tabular form is called as 'two dimensional array.' Such type of array specially used to represent data in a matrix form.

The following syntax is used to represent two dimensional array.

Syntax:

<data-type> <array_nm> [row_subscript][column-subscript];

Example:

```
        int a[3][3];
```

In above example, a is an array of type integer which has storage size of 3 * 3 matrix. The total size would be 3 * 3 * 2 = 18 bytes.

It is also called as 'multidimensional array.'

*** Memory Allocation :**

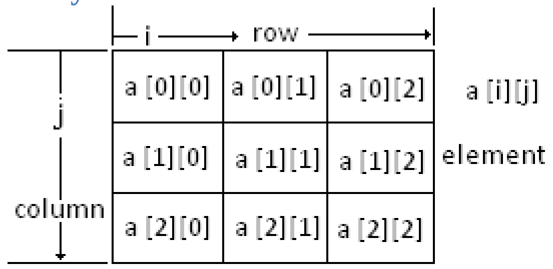


Fig : Memory allocation for two dimensional array

Program :

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int a[3][3], i, j;
        clrscr();
        printf("\n\t Enter matrix of 3*3 : ");
        for(i=0; i<3; i++)
        {
                for(j=0; j<3; j++)
                {
                scanf("%d",&a[i][j]);  //read 3*3 array
                }
        }
        printf("\n\t Matrix is : \n");
        for(i=0; i<3; i++)
        {
                for(j=0; j<3; j++)
                {
                printf("\t %d",a[i][j]);  //print 3*3 array
                }
          printf("\n");
        }
        getch();
}
```

Output :

```
        Enter matrix of 3*3 : 3 4 5 6 7 2 1 2 3

        Matrix is :
        3       4       5
        6       7       2
        1       2       3_
```

Limitations of two dimensional array :

- We cannot delete any element from an array.
- If we dont know that how many elements have to be stored in a memory in advance, then there will be memory wastage if large array size is specified.

Structure :

Structure is user defined data type which is used to store heterogeneous data under unique name. Keyword 'struct' is used to declare structure.

The variables which are declared inside the structure are called as 'members of structure'.

Syntax:

```
struct structure_nm
{
        <data-type> element 1;
        <data-type> element 2;
        - - - - - - - - - - -
        - - - - - - - - - - -
        <data-type> element n;
}struct_var;
```

Example :

```
struct emp_info
{
        char emp_id[10];
        char nm[100];
        float sal;
}emp;
```

Note :

1. Structure is always terminated with semicolon (;).

2. Structure name as emp_info can be later used to declare structure variables of its type in a program.

*** Instances of Structure :**

Instances of structure can be created in two ways as,

Instance 1:

```
struct emp_info
{
		char emp_id[10];
		char nm[100];
		float sal;
}emp;
```

Instance 2:

```
struct emp_info
{
		char emp_id[10];
		char nm[100];
		float sal;
};
struct emp_info emp;
```

In above example, emp_info is a simple structure which consists of stucture members as Employee ID(emp_id), Employee Name(nm), Employee Salary(sal).

* Aceessing Structure Members :

Structure members can be accessed using member operator '**.**' . It is also called as '**dot operator**' or '**period operator**'.

```
structure_var.member;
```

```
#include <stdio.h>
#include <conio.h>

struct comp_info
{
		char nm[100];
		char addr[100];
}info;

void main()
{
		clrscr();
		printf("\n Enter Company Name : ");
		gets(info.nm);
		printf("\n Enter Address : ");
		gets(info.addr);
		printf("\n\n Company Name : %s",info.nm);
		printf("\n\n Address : %s",info.addr);
		getch();
}
```

Output :

```
Enter Company Name : TechnoExam, Technowell Web Solutions
Enter Address : Sangli, Maharashtra, INDIA

Company Name : TechnoExam, Technowell Web Solutions
Address : Sangli, Maharashtra, INDIA
```

Array in Structures :

Sometimes, it is necessary to use structure members with array.

```
#include <stdio.h>
#include <conio.h>

struct result
{
	int rno, mrks[5];
	char nm;
}res;

void main()
{
	int i,total;
	clrscr();
	total = 0;
	printf("\n\t Enter Roll Number : ");
	scanf("%d",&res.rno);
	printf("\n\t Enter Marks of 3 Subjects : ");
	for(i=0;i<3;i++)
	{
		scanf("%d",&res.mrks[i]);
		total = total + res.mrks[i];
	}
	printf("\n\n\t Roll Number : %d",res.rno);
	printf("\n\n\t Marks are :");
	for(i=0;i<3;i++)
	{
		printf(" %d",res.mrks[i]);
	}
	printf("\n\n\t Total is : %d",total);
	getch();
}
```

Output :

```
	Enter Roll Number : 1

	Enter Marks of 3 Subjects : 63 66 68


	Roll Number : 1
```

```
          Marks are : 63 66 68

          Total is : 197_
```

Structure With Array :

We can create structures with array for ease of operations in case of getting multiple same fields.

```
#include <stdio.h>
#include <conio.h>

struct emp_info
{
          int emp_id;
          char nm[50];
}emp[2];

void main()
{
          int i;
          clrscr();
          for(i=0;i<2;i++)
          {
                    printf("\n\n\t Enter Employee ID : ");
                    scanf("%d",&emp[i].emp_id);
                    printf("\n\n\t Employee Name : ");
                    scanf("%s",emp[i].nm);
          }
          for(i=0;i<2;i++)
          {
            printf("\n\t Employee ID : %d",emp[i].emp_id);
            printf("\n\t Employee Name : %s",emp[i].nm);
          }
          getch();
}
```

Output :

```
          Enter Employee ID : 1

          Employee Name : ABC

          Enter Employee ID : 2

          Employee Name : XYZ

          Employee ID : 1
          Employee Name : ABC
          Employee ID : 2
          Employee Name : XYZ_
```

Structures within Structures (Nested Structures) :

Structures can be used as structures within structures. It is also called as 'nesting of structures'.

Syntax:

```
struct structure_nm
{
        <data-type> element 1;
        <data-type> element 2;
        - - - - - - - - - - -
        - - - - - - - - - - -
        <data-type> element n;

        struct structure_nm
        {
                <data-type> element 1;
                <data-type> element 2;
                - - - - - - - - - - -
                - - - - - - - - - - -
                <data-type> element n;
        }inner_struct_var;
}outer_struct_var;
```

Example :

```
struct stud_Res
{
        int rno;
        char nm[50];
        char std[10];

        struct stud_subj
        {
                char subjnm[30];
                int marks;
        }subj;
}result;
```

In above example, the structure stud_Res consists of stud_subj which itself is a structure with two members. Structure stud_Res is called as 'outer structure' while stud_subj is called as 'inner structure.' The members which are inside the inner structure can be accessed as follow :

```
result.subj.subjnm
result.subj.marks
```

```
#include <stdio.h>
#include <conio.h>

struct stud_Res
{
        int rno;
        char std[10];
        struct stud_Marks
```

```
		{
			char subj_nm[30];
			int subj_mark;
		}marks;
}result;

void main()
{
		clrscr();
		printf("\n\t Enter Roll Number : ");
		scanf("%d",&result.rno);
		printf("\n\t Enter Standard : ");
		scanf("%s",result.std);
		printf("\n\t Enter Subject Code : ");
		scanf("%s",result.marks.subj_nm);
		printf("\n\t Enter Marks : ");
		scanf("%d",&result.marks.subj_mark);
		printf("\n\n\t Roll Number : %d",result.rno);
		printf("\n\n\t Standard : %s",result.std);
		printf("\nSubject Code : %s",result.marks.subj_nm);
		printf("\n\n\t Marks : %d",result.marks.subj_mark);
		getch();
}
```

Output :

```
		Enter Roll Number : 1

		Enter Standard : MCA(Sci)-I

		Enter Subject Code : SUB001

		Enter Marks : 63


		Roll Number : 1

		Standard : MCA(Sci)-I
Subject Code : SUB001

		Marks : 63_
```

Pointer :

Pointer is a variable which holds the memory address of another variable. Pointers are represented by '*'. It is a derive data type in C. Pointer returns the value of stored address.

Syntax:

```
		<data_type> *pointer_name;
```

In above syntax,
* = variable pointer_name is a pointer variable.
pointer_name requires memory location
pointer_name points to a variable of type data type.

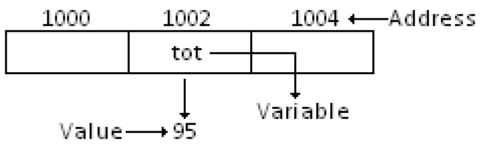
How to Use ?

```
int *tot;
```

Illustration :

```
int tot = 95;
```

Figure :



In above example, the statement instructs the system to find out a location for integer variable quantity and puts the values 95 in that memory location.

*** Features of Pointer :**

* Pointer variable should have prefix '*'.
* Combination of data types is not allowed.
* Pointers are more effective and useful in handling arrays.
* It can also be used to return multiple values from a funtion using function arguments.
* It supports dynamic memory management.
* It reduces complexity and length of a program.
* It helps to improve execution speed that results in reducing program execution time.

```
#include <stdio.h>
#include <conio.h>

void main()
{
        int a=10;
        int *ptr;
        clrscr();
        ptr = &a;
        printf("\n\t Value of a : %d", a);
        scanf("\n\n\t Value of pointer ptr : %d", *ptr);
        printf("\n\n\t Address of pointer ptr : %d", ptr);
        getch();
}
```

Output :

```
        Value of a : 10

        Value of pointer ptr : 10

        Address of pointer ptr : -12_
```

Union :

Union is user defined data type used to stored data under unique variable name at single memory location.

Union is similar to that of stucture. Syntax of union is similar to stucture. But the major **difference between structure and union is 'storage.'** In structures, each member has its own storage location, whereas all the members of union use the same location. Union contains many members of different types, it can handle only one member at a time.

To declare union data type, 'union' keyword is used.

Union holds value for one data type which requires larger storage among their members.

Syntax:

```
union union_name
{
        <data-type> element 1;
        <data-type> element 2;
        <data-type> element 3;
}union_variable;
```

Example:

```
union techno
{
        int comp_id;
        char nm;
        float sal;
}tch;
```

In above example, it declares tch variable of type union. The union contains three members as data type of int, char, float. We can use only one of them at a time.

*** Memory Allocation :**

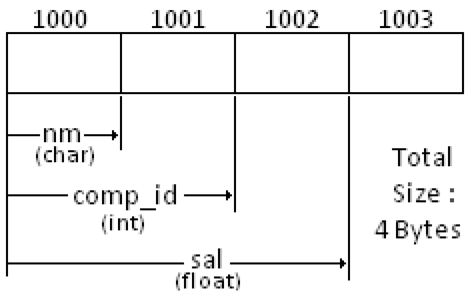


Fig : Memory allocation for union

To access union members, we can use the following syntax.

```
tch.comp_id
tch.nm
tch.sal
```

```
#include <stdio.h>
#include <conio.h>

union techno
{
		int id;
		char nm[50];
}tch;


void main()
{
		clrscr();
		printf("\n\t Enter developer id : ");
		scanf("%d", &tch.id);
		printf("\n\n\t Enter developer name : ");
		scanf("%s", tch.nm);
		printf("\n\n Developer ID : %d", tch.id);//Garbage
		printf("\n\n Developed By : %s", tch.nm);
		getch();
}
```

Output :

```
		Enter developer id : 101

		Enter developer name : technowell

Developer ID : 25972

Developed By : technowell_
```

String Handling in C :

String :

A string is a collection of characters. Strings are always enlosed in double quotes as "string_constant".

Strings are used in string handling operations such as,

- Counting the length of a string.
- Comparing two strings.
- Copying one string to another.
- Converting lower case string to upper case.
- Converting upper case string to lower case.
- Joining two strings.
- Reversing string.

Declaration :

The string can be declared as follow :

Syntax:

```
            char string_nm[size];
```

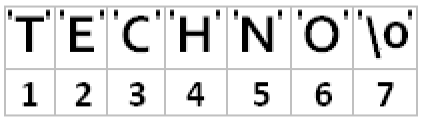
Example:

```
            char name[50];
```

String Structure :

When compiler assigns string to character array then it automatically supplies **null character ('\0')** at the end of string. Thus, size of string = original length of string + 1.

```
            char name[7];
            name = "TECHNO"
```



'T'	'E'	'C'	'H'	'N'	'O'	'\o'
1	2	3	4	5	6	7

Read Strings :

To read a string, we can use scanf() function with format specifier %s.

```
            char name[50];
            scanf("%s",name);
```

The above format allows to accept only string which does not have any blank space, tab, new line, form feed, carriage return.

Write Strings :

To write a string, we can use printf() function with format specifier %s.

```
            char name[50];
            scanf("%s",name);
            printf("%s",name);
```

String handling functions :

'string.h' is a header file which includes the declarations, functions, constants of string handling utilities. These string functions are widely used today by many programmers to deal with string operations.

Some of the standard member functions of string.h header files are,

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
void main()
{
            char str[50];
            clrscr();
            printf("\n\t Enter your name : ");
            gets(str);
            printf("\nLower case of string: %s",strlwr(str));
            printf("\nUpper case of string: %s",strupr(str));
```

```
                printf("\nReverse of string: %s",strrev(str));
                printf("\nLength of String: %d",strlen(str));
                getch();
}
```

Output :

```
Enter your name : Technoexam
Lower case of string: technoexam
Upper case of string: TECHNOEXAM
Reverse of string: MAXEONHCET
Length of String: 10_
```

Header File in C :

Header file contains different predefined functions, which are required to run the program. All header files should be included explicitly before main ( ) function.

It allows programmers to seperate functions of a program into reusable code or file. It contains declarations of variables, subroutines. If we want to declare identifiers in more than one source code file then we can declare such identifiers in header file. Header file has extension like '*.h'. The prototypes of library functions are gathered together into various categories and stored in header files.

E.g. All prototypes of standard input/output functions are stored in header file 'stdio.h' while console input/output functions are stored in 'conio.h'.

The header files can be defined or declared in two ways as

Method 1 : #include "header_file-name"
Method 2 : #include <header_file-name>

Method 1 is used to link header files in current directory as well as specified directories using specific path. The path must be upto 127 characters. This is limit of path declaration. Method 2 is used to link header files in specified directories only.

Standard Header Files :

Followings are the some commonly used header files which plays a vital role in C programming :

Assert.h	Ctype.h	Math.h
Process.h	Stdio.h	Stdlib.h
String.h	Time.h	Graphics.h