

HISTORY OF C

C Programming

By

Mrs. R.Waheetha, MCA, M.Phil

Head,

Department of Computer Science

Holy Cross Home Science College

Thoothukudi

- C was evolved from ALGOL, BCPL and B by Dennis Ritchie at the Bell Laboratories in 1972.
- C is highly portable.
- This means that C programs written for one computer can be run on another with little or no modification.
- Portability is important if we plane to use a new computer with a different operating system.
- C language is well structured in terms of function modules or blocks.
- This modular structure makes program debugging, testing and maintenance easier.

CHARACTER SET

- The characters that can be used to form words, numbers and expressions form C set.
- The characters in C are grouped into the following categories:
 1. Letters
 2. Digits
 3. Special characters
 4. White spaces

KEYWORDS AND IDENTIFIERS

- All keywords have fixed meanings and these meanings cannot be changed.
- All keywords must be written in lowercase.
- Identifiers refer to the names of variables, functions and arrays.
- These are user-defined names and consist of a sequence of letters and digits, with a letter as first character.



- **Rules for Identifiers**

- 1. First character must be an alphabet (or underscore).
- 2. Must consist of only letters, digits or underscore.
- 3. Only first 31 characters are significant.
- 4. Cannot use a keyword.
- 5. Must not contain white space.

CONSTANTS

- Constants in C refer to fixed values that do not change during the execution of a program.
- **Integer Constants**
 - An integer constant refers to a sequence of digits. There are three types of integers, namely, decimal integer, octal integer and hexadecimal integer.
 - **Decimal integers** consist of a set of digits, 0 through 9 preceded by an optional – or + sign.
 - Valid examples of decimal integer constants are:
 $123 - 321 0 654321 + 78$
 - An octal integer constant consists of any combination of digits from the set 0 through 7, with a leading 0.
 - Some examples of octal integer are:
 $037 0 0435 0551$

- A sequence of digits preceded by 0x or 0X is considered as **hexadecimal integer**.
- They may also include alphabets A through F or a through f.
- The letter A through F represents the numbers 10 through 15
- Following are the examples of valid hex integers:

A23D, ADE, 125F

- **Real constants**

- Numbers containing fractional parts like 17.548 are called real (or floating point) constants are:

0.0083 – 0.75 435.36 + 247.0

- A real number may also be expressed in exponential (or scientific) notation.
- This notation floating point form.

0.65e4 12e-2 1.5e+5 3.18E3 -1.2E-1

• **Single Character Constants**

- A single character constant (or simply character constant) contains a single character enclosed within a pair of single quote marks.

“5”, “X”, “ “, “ ”, “ ”

• **String Constants**

- A string constant is sequence of characters enclosed in double quotes.
- The characters may be letters, numbers, special characters and blank space. Examples are:

“Hello!”, “1987”, “WELL DONE”, “?...!”, “5+3”

• **Backslash character Constants**

- C supports some special backslashes character constants that are used in output functions each one of them represents one character, although they consist of two characters.
- These characters combinations are known as escape sequences.

- **Backslash character Constants**

Constant

Meaning

“\f”

Form feed

“\n”

New line

“\t”

Horizontal tab

“\v”

Vertical tab

“\r”

Carriage Return

“\b”

Back space

● **VARIABLES**

- A variable is a data name that may be used to store a data value.
- Variable take different values at the time of execution.

● **DATA TYPES**

- ANSI C supports three classes of data types:
 - 1. Primary (or fundamental) data types
 - 2. Derived data types
 - 3. User-defined data types

- **Primary data types in C**

- Table Size and Range of Basic Data Types on 16- bit Machines

Data type Range of values

- Char -128 to 127
- Int -32,768 to 32,767
- Float 3.4e-38 to 3e+e38
- Double 1.7-308 to 1.7e+308

● Integer Types

- Integers are whole numbers with a range of values supported by a particular machine.
- Integers occupy one word of storage C has three classes of integer storage, namely short int, int, and long int, in both signed and unsigned forms.
- Short int represents small integer values and requires half the amount of storage as a regular int number uses.
- We declare long and unsigned integers to increase the range of values.

● Floating Point Types

- Floating point (or real) numbers are stored in 32 bits (on all 16 bit and 32 bit machines), with 6 digits of precision.
- Floating point numbers are defined in C by the keyword float.
- When the accuracy provided by a float number is not sufficient, the type double can be used to define the number.

● **Void Types**

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

● **Character Types**

- A single character can be defined as a character (char) type data.
- Characters are usually stored in 8 bits (one byte) to internal storage.
- The qualifier signed or unsigned may be explicitly applied to char.
- While unsigned chars have values between 0 and 255, signed chars have values from – 128 to 127

DECLARATION OF VARIABLES

- After designing suitable variable names, we must declare them to the compiler.
- Declaration does two things:
 1. It tells the compiler what the variable name is.
 2. It specifies what type of data the variable will hold.
- Primary Type Declaration
 - A variable can be used to store a value of any data type.
 - The syntax
 - $$\text{Data-type } v_1, v_2, \dots, v_n ;$$

v_1, v_2, \dots, v_n are the names of variables.
 - For example,
 - `int count ;`
 - `int number, total ;`
 - `double ratio ;`

User-Defined Type Declaration

- C supports a feature known as “type definition” that allows users to define an identifier that would represent an existing data type.
- The user-defined data type identifier can later be used to declare variables.
- It takes the general form:
 - typedef type identifier:
- Some examples
 - typedef int units ;
 - typedef float marks ;
- Another “identifier” is a user-defined data type is enumerated data type provided by
- ANSI standard.
- It is defined as follows:
 - enum identifier {value1, value2,...valuen};

- The “identifier” is a user-defined enumerated data type which can be used to declare variables that can have one of the values enclosed within the braces (known as enumeration constants).
- After this definition, we can declare variables to be of this „new“ type as below:
- `enum identifier v1, v2,...vn;`
- The enumerated variables `v1,v2,...vn` can only have one of the values `value1, value2,...valuen`.
- An example:
 - `enum day {Monday, Tuesday, Sunday} ;`
 - `enum day week_st, week_end ;`
 - `week_st = Monday ;`
 - `if (week_st == Tuesday)`
`week_end = Saturday ;`

DECLARATION OF STORAGE CLASS

- Variables in C can have not only data type but also storage class that provides information about their location and visibility.
- The storage class decides the portion of the program within which the variables are recognized.
- C provides a variety of storage class specifies that can be used to declare explicitly the scope and lifetime of variables.
- There are four storage class specifies (auto, register, static, and extern).
- Static and external (extern) variables are automatically initialized to zero.
- Automatic (auto) variables contain undefined values (known as „garbage“) unless they are initialized explicitly.

Storage classes and their Meaning

Storage

- auto
- Static
- extern
- Register

Class Meaning

Local variable known only to the function in which it is declared. Default is auto.

Local variable which exists and retains its value even after the control is transferred to the calling function.

Global variable known to all functions in the file.

Local variable which is stored in the register

ASSIGNING VALUES TO VARIABLES

- **ASSIGNING VALUES TO VARIABLES**

- Assignment Statement

- Values can be assigned to variables using the assignment operator = as follows:

Variable_name = constant;

- Balance = 75.84 ;
- Yes = „x“ ;
- Another way of giving values to variables is to input data through keyboard using the scanf function.

Scanf(“control string”, &variable1,&variable2,...);

- The control sting contains the format of data being received.
- The ampersand symbol & before each variable name is an operator that specifies the variable name’s address.

DEFINING SYMBOLIC CONSTANTS

- We often use certain unique constants in a program.
- These constants may appear repeatedly in a number of places in the program.
- One example of such a constant is 3.142, representing the value of the mathematical constant “pi”.
- A constant is defined as follows:

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

- Valid examples
- #define STRENGTH 100
- # define PASS_MARK 50
- #define p1 3.14159
- Symbolic names are sometimes called constant identifiers.

- The following rules apply to a #define statement which define a symbolic constant:

1. Symbolic names have the same form as variable names.

(Symbolic names are written in CAPITALS to visually distinguish them from the normal variable names, which are written in lowercase letters.

This is only a convention, not a rule.

2. No blank space between the hash sign “#” and the word.

3. „#” must be the first character in the line.

4. A blank space is required between # define and symbolic name.

5. #define statements must not end with a semicolon.

6. After definition, the symbolic name should not be assigned any other value within the program by using an assignment statement.

For example, `PASS_MARK = 35;` is illegal.

7. Symbolic names are NOT declared for data types. Its data type depends on the type of constant.

8. #define statements may appear anywhere in the program but before it is referenced in the program (the usual practice is to place them in the beginning of the program).

• DECLARING A VARIABLE AS CONSTANT

- We may like value of certain variables to remain constant during the execution of a program.
- Example:
 - `Const int class_size = 40;`
 - Const is a new data type qualifier defined by ANSI standard.
 - This tells the compiler that the value of the int variable `class_size` must not be modified by the program.

Simple “c” programs

1. *Area of a square*

```
#include <stdio.h>
#include <conio.h>
void main ( )
{
int a, s ;
printf (“Enter s value \n”);
scanf (“ % d”,&s);
printf (“ S = % d\ n ” ,s);
a = s * s;
printf (“Area of square = % d\n”, a);
getch();
}
```

- **Area of a triangle**

- $a = \frac{1}{2} bh$

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

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

```
Void main ( )
```

```
{
```

```
int b, h ;
```

```
float a ;
```

```
print f (“Enter b, h value \n”) ;
```

```
scan f (“%d% d” , &b, &h) ;
```

```
print f (“ b = %d h = %d / n” , b, h) ;
```

```
a = (0.5) * b * h ;
```

```
printf (“Area of a triangle = %f \n” , a );
```

```
getch();
```

```
}
```

OPERATORS AND EXPRESSIONS

- An operator is symbol used to manipulate
 - C operators can be classified into a number of categories. They include:
 1. Arithmetic operators
 2. Relational operators
 3. Logical operators
 4. Assignment operators
 5. Increment and decrement operators
 6. Conditional operators
 7. Bitwise operators
 8. Special operators

ARITHMETIC OPERATORS

- C provided all the basic arithmetic operates.

Operator

Meaning

+

Addition or unary plus

-

Subtraction or unary minus

x

Multiplication

/

Division

%

Modulo division

- Integer division truncates any fractional part.
- The modulo division operation produces the remainder of an integer division.

● Integer Arithmetic

- When both the operands in a single arithmetic expression such as $a+b$ are integers, the expression is called an integer expression, and the operation is called integer arithmetic.
- Integer arithmetic always yields an integer value.
- A and b are integers, $a = 14$ and $b = 4$
- $a - b = 10$
- $a + b = 18$
- $a \times b = 56$
- $a / b = 3$ (decimal part truncated)
- $a \% b = 2$ (remainder of division)

● Real Arithmetic

- An arithmetic operation involving only real operands is called real arithmetic.
- A real operand may assume values either in decimal or exponential notation.
- The operator $\%$ cannot be used with real operands.

● Mixed-mode Arithmetic

- When one of the operands is real and the other is integer, the expression is called a mixed-mode arithmetic expression.
- If either operand is of the real type the result is always a real number.
- $15/10.0 = 1.5$

RELATIONAL OPERATORS

- We often compare two quantities and depending on their relation, take certain decisions.
- An expression such as $A < b$ $1 < 20$
- An expression containing a relational operator is termed as a relational expression.
- The value of a relational expression is either True or false.
- C supports six relational operators in all.
- These operators and their meanings are shown in Table

Operator

Meaning

<

is less than

<=

is less than or equal to

>

is greater than

>=

is greater than or equal to

==

is equal to

!=

is not equal to

LOGICAL OPERATORS

- In addition to the relational operators, C has the following three logical operators.
 - `&&` meaning logical AND
 - `||` meaning logical OR
 - `!` meaning logical NOT
- The logical operators `&&` and `||` are used when we want to test more than one condition and make decisions.
- An expression of this kind, which combines two or more relational expressions, is termed as a logical expression or a compound relational expression.
- Some examples of the usage of logical expressions are:
 - `if (age > 55 && salary < 1000)`
 - `if (number < 0 || number > 100)`

ASSIGNMENT OPERATORS

- Assignment operators are used to assign the result of an expression to a variable.

$v \text{ op} = \text{exp};$

- Where V is a variable, exp is an expression and op is a C binary arithmetic operator.
- The operator $\text{op} =$ is known as the shorthand assignment operator.

An example

$x += y + 1;$

This is same as the statement

$x = x + (y + 1);$

INCREMENT AND DECREMENT OPERATORS

- C allows two very useful operators increment and decrement operators:

++ and --

- The operator ++ adds 1 to the operand, while – subtracts 1.
- Both are unary operators and takes the following form:
 - ++m; or m++;
 - m; or m--;
 - ++m; is equivalent to m = m+1; (or m + = 1;)
 - m; is equivalent to m = m-1; (or m -= 1;)

BITWISE OPERATORS

- C has special operators known as bitwise operators for manipulation of data at bit level.
- These operators are used for testing the bits, or shifting them right or left.
- Bitwise operators may not be applied to float or double.

Operator

&

|

^

<<

>>

Meaning

bitwise AND

bitwise OR

bitwise exclusive OR

shift left

shift right

● **SPECIAL OPERATORS**

- C supports some special operators of interest such as comma operator, size of operator, pointer operators (& and *) and member selection operators (. and →).
- **The comma Operator**
 - The comma operator can be used to link the related expressions together.
 - The sizeof is a compile time operator and, when used with an operand, it returns the number of bytes the operand occupies.
 - The operand may be a variable, a constant or a data type qualifier.

ARITHMETIC EXPRESSIONS

- An arithmetic expression is a combination of variables, constants, and operators arranged as per the syntax of the language.

- **EVALUATION OF EXPRESSIONS**

- Expressions are evaluated using an assignment statement of the form:

variable = expression;

- Variable is any valid C variable name.
- When the statement is encountered, the expression is evaluated first and the result then replaces the previous value of the variable on the left-hand side.
- Examples:
 - $x = a * b - c;$
 - $z = a - b / c + d;$


PRECEDENCE OF ARITHMETIC OPERATORS

- An arithmetic expression without parentheses will be evaluated from left to right using the rules of precedence of operators.
- There are two distinct priority levels of arithmetic operators in C:
 - High priority * / %
 - Low priority + -
- The basic evaluation procedure includes „two“ left-to-right passes through the expression.
- During the first pass, the high priority operators are applied as they are encountered.
- During the second pass, the low priority operators are applied as they are encountered.
- Example:
 - $x = a - b / 3 + c * 2 - 1$
When $a = 9$, $b = 12$, and $c = 3$, the statement becomes
 $x = 9 - 12 / 3 + 3 * 2 - 1$ Answer is 10
- However, the order of evaluation can be changed by introducing parentheses into an expression.
- Consider the same expression with parentheses as shown below:
 - $9 - 12 / (3 + 3) * (2 - 1)$
- Whenever, parentheses are used, the expressions within parentheses assume highest priority, if two or more sets of parentheses appear one after another as shown above, the expression contained in the left-most set is evaluated first and the right-most in the last.
- Answer is 7



- **Rules of Precedence and Associability**

- Precedence rules decides the order in which different operators are applied.
- Associability rule decides the order in which multiple occurrences of the same level operator are applied.



MANAGING INPUT AND OUTPUT OPERATIONS

FORMATTED OUTPUT

- The printf statement provides certain features that can be effectively exploited to control the alignment and spacing of print-outs on the terminals.

- The general form of printf statement is:

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

Control string consists of three types of items:

1. Characters that will be printed on the screen as they appear.
 2. Format specifications that define the output format for display of each item.
 3. Escape sequence characters such as \n, \t, and \b.
- The control string indicates how many arguments follow and what their types are.
 - The arguments arg1, arg2,, argn are the variables whose values are formatted and printed according to the specifications of the control string.
 - The arguments should match in number, order and type with the format specifications.

Examples

```
printf("Welcome to c class");  
printf("");  
printf("\n");  
printf("%d",x);  
printf("A=%f\n B= %f",a,b);  
printf("Welcome to c class);
```

- A simple format specification has the following form

- %wd type specifier

Where w specifies the minimum field width

Eg:

```
printf("%5d\n",m)
```

- %wc

The character will be right justified in fields of w column

Eg:

```
printf("%5s\n",name);
```


FORMATTED INPUT

- Formatted input refers to an input data that has been arranged in a particular format.
- The general form of scanf is
 Scanf (“control string”, arg1, arg2,.....argn);
- The control string specifies the field format in which the data is to be entered and the arguments arg1, arg2,....., argn specify the address of locations where the data is stored.
- Control string and arguments are separated by commas.
- Control string (also known as format string) contains field specifications, which direct the interpretation of input data.
- It may include:
 - Field (or format) specifications, consisting of the conversion character %, a data type character (or type specifier), and an optional number, specifying the field width.
 - Blanks, tabs and newlines are ignored.
 - The data type character indicates the type of data that is to be assigned to the variable associated with the corresponding argument.
 - The field width specifier is optional.

- Eg:

- `scanf("%d",&a);`
- `scanf("%d%d",&a,&b);`
- `scanf("%f%d",&a,&b);`
- `scanf("%c",&s);`
- `scanf("%s",&name);`

● Commonly used scanf Format Codes

- | ● Code | Meaning |
|--|----------------------------------|
| ● %c | Read a single character |
| ● %d | Read a decimal integer |
| ● %e | Read a floating point value |
| ● %f | Read a floating point value |
| ● %u | Read an unsigned decimal integer |
| ● %s | Read a string |
| ● The following letters may be used as prefix for contain conversion characters. | |
| ● h | for short integers |
| ● % ld | for long integers |
| ● I for long integers or double | |
| ● L for long double | |

Reading a Character

- Reading a single character can be done by using the function `getchar()`
 - `Variable_name = getchar()`
Variable_name is a valid name.

Eg:

```
char name;  
name = getchar();
```

Writing a Character

- Function `putchar()` is used for writing a characters one at a time
 - `Putchar(variable_name);`
 - Where `variable_name` is a type `char` variable.
 - Eg
 - `ans = 'y';`
 - `putchar(ans);`
 - `Putchar(\n)`

UNIT II

DECISION MAKING AND BRANCHING

C Programming

By

Mrs. R.Waheetha, MCA, M.Phil
Head,
Department of Computer Science
Holy Cross Home Science College
Thoothukudi

- C language possesses such decision-making capabilities by supporting the following statements:

1. If statement
2. Switch statement
3. Conditional operator statement
4. Goto statement

DECISION MAKING WITH IF STATEMENT

- The if statement is a powerful decision-making statement and is used to control the flow of execution of statements.
- It is basically a two-way decision statement and is used in conjunction with an expression.
- It takes the following form
if (test expression)
- The if statement may be implemented in different forms depending on the complexity of conditions to be tested.

Types of if

1. Simple if statement
2. if.....else statement
3. Nested if....else statement
4. else if ladder.

SIMPLE IF STATEMENT

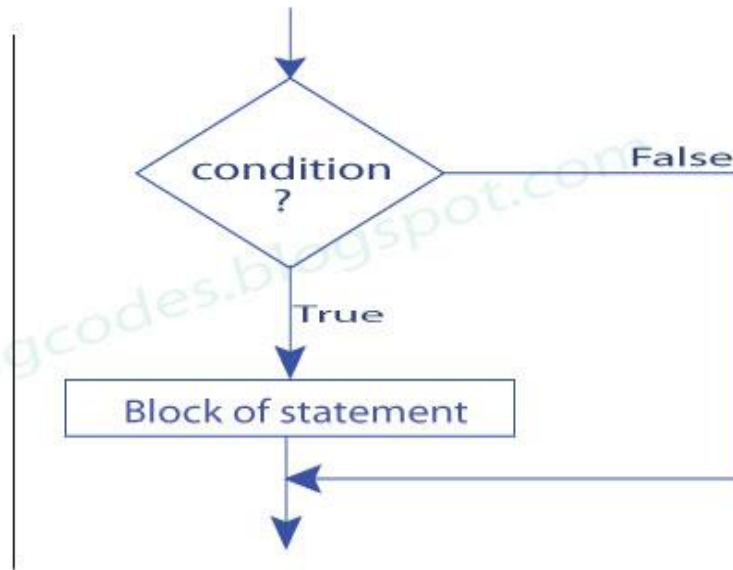
- The general form of a simple if statement is

```
if (test expression)
{
    Statement-block;
}
Statement-x;
```

- The “statement-block” may be a single statement or a group of statements.
- If the test expression is true, the statement-block will be executed; otherwise the statement-block skipped and the execution will jump to the statement-x.
- Remember, when the condition is true both the statement-block and the statement-x are executed in sequence.

- Flow chart

```
if(condition)
{
    block of statement
}
```



- Eg

```
discount =0;
```

```
if (amount >2500)
```

```
    discount = 10/100;
```

```
amount= ampunt+discount;
```

THE IF ... ELSE STATEMENT

- The if..... else statement is an extension of the simple if statement.
- The general form is

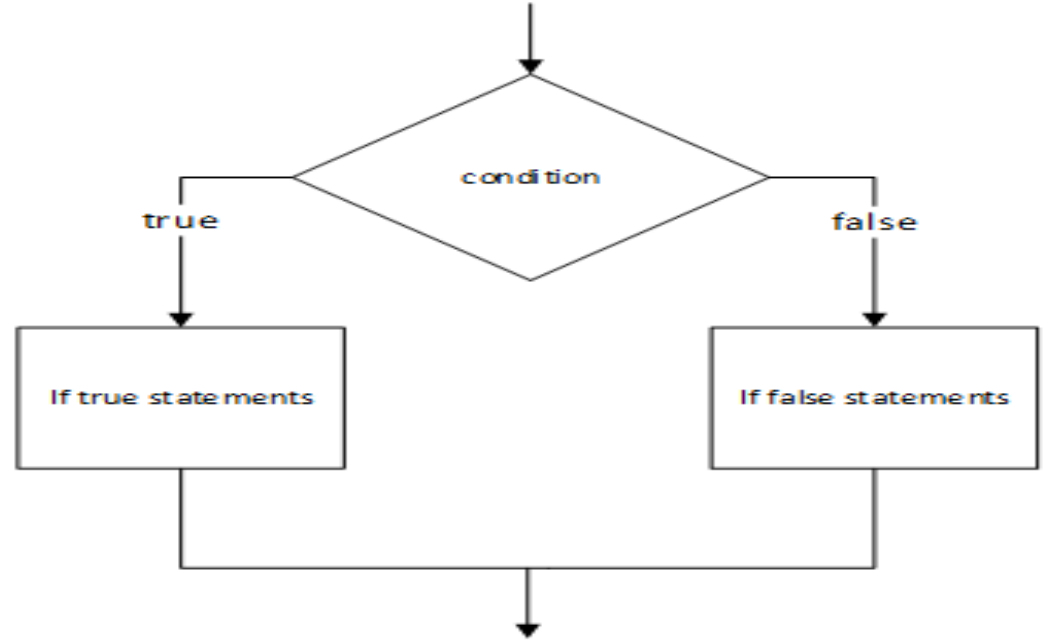
```
if(test condition)
{
    true-block;
}
else
{
    false-block;
}
```

- if the test expression is true, then the true-block statement(s), immediately following the if statements are executed;
- otherwise, the false-block statement(s), immediately following the if statements are executed; otherwise, the false-block statement(s) are executed.
- In either case, either true-block or false-block will be executed, not both.

- Flowchart

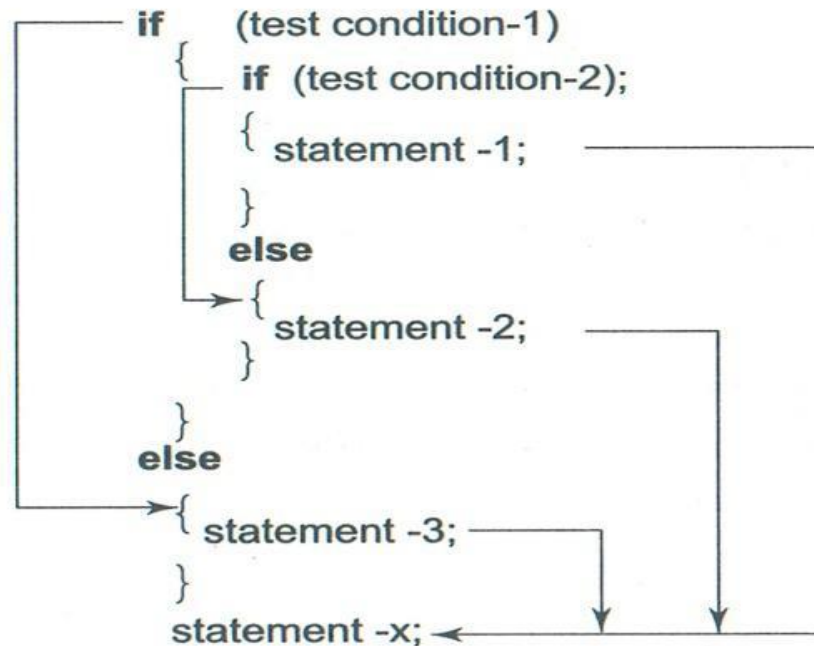
- Eg

```
int num=19;  
if(num<10)  
{  
    printf("The value is less than 10");  
}  
else  
{  
    printf("The value is greater than 10");  
}
```



NESTING OF IF...ELSE STATEMENTS

- When a series of decisions are involved, we may have to use more than one if...else.



- If condition-1 is true, the condition-2 will be checked. If it is true, statement-1 will be executed; otherwise, statement-2 will be executed. If condition-1 is false, statement-3 will be executed.

- Example

- `if (age < 18)`

```
{  
    printf("You are Minor.\n");  
    printf("Not Eligible to Work");  
}  
else  
{  
    if (age >= 18 && age <= 60 )  
    {  
        printf("You are Eligible to Work \n");  
        printf("Please fill in your details and apply\n");  
    }  
    else  
    {  
        printf("You are too old to work as per the Government rules\n");  
        printf("Please Collect your pension! \n");  
    }  
}
```

THE ELSE IF LADDER

- A multipath decision is a chain of ifs in which the statement associated with each else is an if.

```
if ( condition 1)  
    statement-1;
```

```
else if ( condition 2)  
    statement-2;
```

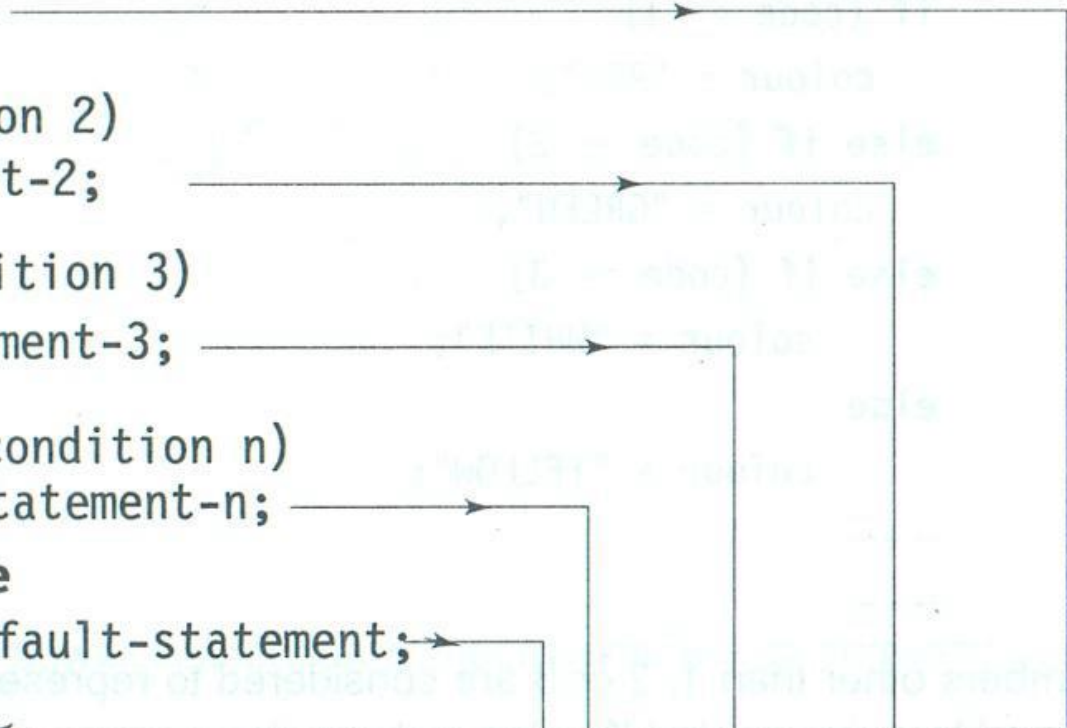
```
else if ( condition 3)  
    statement-3;
```

```
else if ( condition n)  
    statement-n;
```

```
else
```

```
    default-statement;
```

```
statement-x;
```



- This construct is known as the else if ladder.
- The conditions are evaluated from the top (of the ladder), downwards.
- As soon as a true condition is found, the statement associated with it is executed and the control is transferred to the statement-x (skipping the rest of the ladder).
- When all the n conditions become false, then the final else containing the default statement will be executed.

• Example

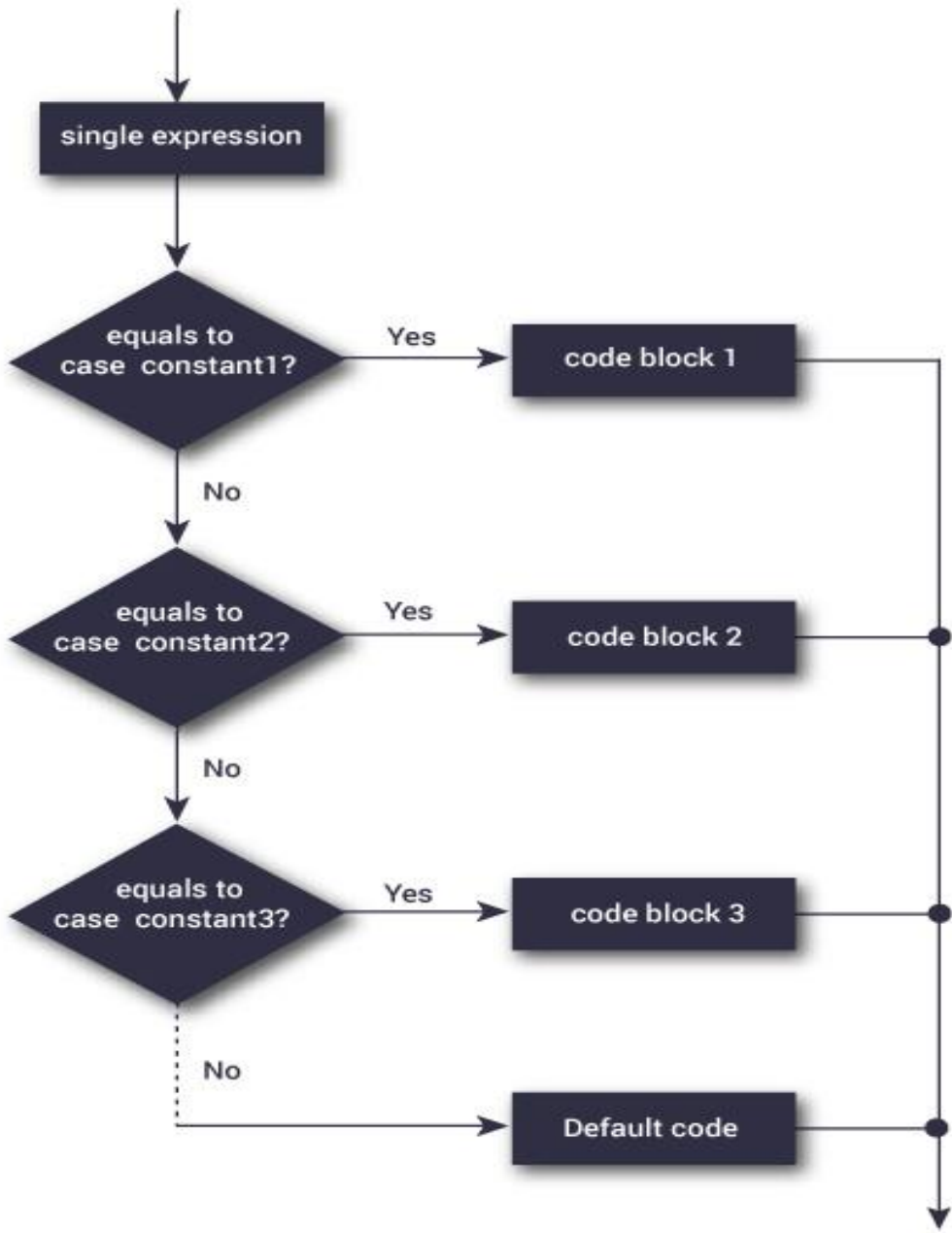
```
printf("Enter an integer between 1 and 6");  
scanf("%d",& x);  
if(x==1)    printf("\n The number is one ");  
else if(x==2) printf("\n The number is two ");  
else if(x==3) printf("\n The number is three ");  
else if(x==4) printf("\n The number is four ");  
else if(x==5) printf("\n The number is five ");  
else if(x==6) printf("\n The number is six ");  
else printf("\n You didn't follow the rule");
```

THE SWITCH STATEMENT

- C has a built-in multiway decision statement known as a switch.
- The switch statement tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statements associated with that case is executed.
- The general form of the switch statement is as shown below:

```
Switch (expression)
{
    Case value-1 :
        Block-1
        Break;
    Case value-2 :
        block-2
        break;
    .....
    .....
    Default:
        Default-block
        break;
}
Statement-x
```

- The expression is an integer expression or characters.
- Value-1 value-2.... Are constants or constant expressions (evaluable to an integral constant) and are known as case labels.
- Each of these values should be unique within a switch statement. Block-1. Block-2.... Are statement lists and may contain zero or more statements.
- There is no need to put braces around these blocks.
- Note that case labels end with a colon (:)
- When the switch is executed, the value of the expression is successfully compared against the values value-1, value-2,... if a case is found whose value matches with the value of the expression, then the block of statement, transferring the control to the statement-x following the switch.
- The default is an optional case.
- When present, it will be executed if the value of the expression does not match with any of the case values.
- If not present, no action takes place if all matches fail and the control goes to the statement-x.



```
#include <stdio.h>
int main()
{
    int num=2;
    switch(num+2)
    {
    case 4:
        printf("Case1: Value is: %d", num);
        break;
    case 6:
        printf("Case2: Value is: %d", num);
        break;
    case 8:
        printf("Case3: Value is: %d", num);
        break;
    default:
        printf("Default: Value is: %d", num);
    }
    getch();
}
```

```
#include <stdio.h>
int main ()
{
char grade = 'B';
switch(grade)
{
case 'A' :
printf("Excellent!\n" );
break;
case 'B' :
case 'C' :
printf("Well done\n" );
break;
case 'D' :
printf("You passed\n" );
break;
case 'F' :
printf("Better try again\n" );
break;
default :
printf("Invalid grade\n" );
}
printf("Your grade is %c\n", grade );
getch();
}
```


THE ? : OPERATOR

- The C language has an operator, useful for making two-way decisions.
- This operator is a combination of ? and :, and takes three operands.
- This operator is popularly known as the conditional operator.
- The general form of use of the conditional operator is as follows:
conditional expression ? expression1 : expression2
- The conditional expression is evaluated first. If the result is non-zero, expression1 is evaluated and is returned as the value of the conditional expression.
- Otherwise, expression 2 is evaluated and its value is returned.

Eg:

```
flag = (x < 0) ? 0 : 1;
```

THE GOTO STATEMENT

- C supports the goto statement to branch unconditionally from one point to another in the program there may be occasions when the use of go to might be desirable.
- The goto requires a label in order to identify the place where the branch is to be made.
- A label is any valid variable name, and must be followed by a colon.
- The label is placed immediately before the statement where the control is to be transferred.
- The general forms of goto and label statements are shown below:

```
goto label;
```

Arrays

UNIT III

C Programming

By

Mrs. R.Waheetha, MCA, M.Phil
Head,

Department of Computer Science
Holy Cross Home Science College
Thoothukudi

Unit – III: ARRAYS

- One-dimensional arrays – Declaration of One-dimensional arrays – Initialization of One-dimensional arrays - Two-dimensional arrays – Initialization of Two-dimensional arrays – Multi-dimensional arrays. Character Arrays and Strings:- Declaring and Initializing String Variables – Reading Strings from Terminal – Writing Strings to Screen – String Handling Functions. (10L)

- An array is a set of homogeneous (same data type) elements.
- declaration
- data type name of array[size] ;
- eg.

<code>float x [10] ;</code>	1 d float array
<code>int a [5] ;</code>	1-d integer array
<code>int s [5] [5] ;</code>	2-d integer array (matrix)
<code>char name [24] [20] ;</code>	character
<code>char address [20] ;</code>	array

One Dimensional Array

- A list of items with one variable name using only one subscript is called single subscripted or one dimensional array.
- Single Subscripted variable x_i is expressed as
 $x[1], x[2], x[3], \dots, X[n]$
- Subscript can begin with number 0 $x[0]$
- Example
 - `int num[5];`
 - The computer reserves five locations

Declaration of One Dimensional Array

- Arrays must be declared before they are used.
- The compiler will allocate space for them in memory
- General form of declaration is
 - Type variable name[size]
- Type will specify the data t in the type of the element in the array.
- Size indicates the maximum number of elements that can be stored in the array
- `int height[50];`
- `float a[5];`
- The size should be either a numeric constant or a symbolic constant
- C treats character strings as array of characters
 - `char name[10];`

INITIALIZATION OF ONE-DIMENSIONAL ARRAYS

- Array can be initialized at either of the following stages:
 - At compile time
 - At run time
- Compile Time Initialization
- We can initialize the elements of arrays in the same way as the ordinary variables when they are declared.
- The general form of initialization of arrays is:
`type array-name[size]={list of values};`
- The values in the list are separated by commas.
- For example, the statement
- `int number [3] = { 0,0,0 };`
- Will declare the variable number as an array of size 3 and will assign zero to each element.
`float total[5] = {0,0,15,75,-10};`
- An array can be explicitly initialized at run time.
- This approach is usually applied for initializing large arrays.

Two dimensional Array

- C allows to define table of items by using two dimensional array.

- It is declared as follows

```
type array_name[rowsize][columnsize];
```

- Two dimensional arrays are stored in memory as given below

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

- Each dimension of the array is indexed from zero to its maximum size minus one.

Initializing two dimensional array

- It can be initialized following their declaration with a list of values enclosed in braces.
- Eg
 - `int t[2][3] = {0,0,0,1,1,1};`
 - This initializes the elements of the first row to zero and the second row to one;

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

main()

```
{
    int a[2][2], b[2][2], result[2][2], i, j;
    clrscr();
    printf("Enter elements of 1st matrix\n");
    for ( i = 0; i < 2; ++i)
    for ( j = 0; j < 2; ++j)
    {
        scanf("%d", &a[i][j]);
    }
    printf("Enter vlaue of 2nd matrix\n");
    for ( i = 0; i < 2; ++i)
    for ( j = 0; j < 2; ++j)
    {
        scanf("%d", &b[i][j]);
    }
    for ( i = 0; i < 2; ++i)
    for ( j = 0; j < 2; ++j)
    {
        result[i][j] = a[i][j] + b[i][j];
    }
    printf("\nSum Of Matrix:")
    for ( i = 0; i < 2; ++i)
    for ( j = 0; j < 2; ++j)
    {
        printf("%d\t", result[i][j]);
    }
    getch();
    return 0; }
```

MULTI-DIMENSIONAL ARRAYS

- C allows arrays of three or more dimensions.
- The exact limit is determined by the compiler.
- The general form of a multi-dimensional array is
 - `type array_name[s1] [s2] [s3]....[sm];`
- Where *s* is the size of the *i*th dimension.
- Some example are:
- `int survey [3] [5] [4] [12];`
- `float table [5] [4] [3];`
- `survey` is a three-dimensional array declared to contain 180 integer type elements.
- Similarly `table` is a four-dimensional array containing 300 elements of floating-point type.
- The array `survey` may represent a survey data of rainfall during the last three years from January to December in five cities.

CHARACTER ARRAYS AND STRINGS

- A string is a sequence of characters that is treated as a single data item.
- Any group of characters (excepted double quote sign) defined between double quotation marks is a string constant.
- Example:

“Man is obviously made to think.”

- Character strings are often used to build meaningful and readable programs.
- The common operations performed on character strings include:
 - Reading and writing strings.
 - Combining strings together.
 - Copying one string to another.
 - Comparing strings for equality.
 - Extracting a portion of a string.

DECLARING AND INITIALIZING STRING VARIABLES

- C does not support strings as a data type.
- However, it allows us to represent strings as character arrays.

```
char stringname[size];
```
- The size determines the number of characters in the string_name.
- Some examples are:

```
char city[10];  
char name[30];
```
- Character arrays may be initialized when they are declared.

```
char city [9] = " NEW YORK ";  
char city [9]={,,N", ,,E", ,,W", ,, ", ,,Y", ,,0", ,,R", ,,K", ,,/0"};
```
- The familiar input function scanf can be used %s format specification to read in a string of characters.
- Example:

```
char address[10]  
scanf("%s", address);
```


- The problem with the scanf function is that it terminates its input on the first white space it finds.
- A white space includes blanks, tabs, carriage returns, form feeds, and new lines.
- The scanf function automatically terminates the string that is read with a null character and therefore the character array should be large enough to hold the input string plus the null character.

gets()

- Another and more convenient method of reading a string of text containing whitespaces is to use the library function gets available in the `<stdio.h>` header file.
 - This is a simple function with one string parameter and called as under;
- gets(str);
- str is variable declared properly. It reads characters into str from the keyboard until a new-line character is encountered and then appends a null character to the string.

```
char line [80];  
gets (line);  
printf (“%s”, line);
```

WRITING STRINGS TO SCREEN

- The printf function with % format is used to print strings to the screen.
- For example:

```
Printf(“%s”, name);
```

- Can be used to display the entire contents of the array name.
- Using putchar and puts Function We can use this putchar() function repeatedly to output a string of characters stored in an array using a loop.
- Example.

```
char name[6] = “PARIS”  
for (i=0, i<5; i++)
```

```
    putchar (name) [i];
```

```
    putchar (”,\n”);
```

- more convenient way of printing string values is to use the function puts declared in the header file (stdio.h).
- This is a one parameter function and invoked as under:

```
puts ( str );
```

- Where str is a variable containing a string value.
- This prints the value of the string variable str and then moves the cursor to the beginning of the next line on the screen.
- For example, the program segment.

```
char line [80];  
gets (line);  
puts (line);
```

- Reads a line of text from the keyboard and displays it on the screen.

STRING-HANDLING FUNCTIONS

- The C library supports a large number of string-handling functions that can be used to
- carry out many of the string manipulations the most commonly used string handling
- functions.
- **Function Action**
- `strcat()` concatenates two strings
- `strcmp()` compares two strings
- `strcpy()` copies one string over another
- `strlen()` finds the length of a string

strcpy()

- Its function works like a string assignment operator.
- It takes the form

```
strcpy(string1,string2);
```

- It assigns the content of string2 to string1.
- string1,string must be character array variable.
- Example

```
char s1[20],s2[20];  
strcpy(s1,"Hello");  
strcpy(s2,s1);
```

- Here Hello is copied to s2;

Strcat()

- The strcat function joins two strings together. It takes the following form

```
strcat(string1, string2);
```

- String1 and string2 are character arrays.
- When the function strcat is executed, string2 is appended to string1.
- It does so removing the null character at the end of string1 and placing string2 from there.
- The string at string2 remains unchanged.

Example

```
#include <stdio.h>
#include <string.h>
int main ()
{
char str1[50], str2[50];
//destination string
strcpy(str1, "This is my initial string");
//source string
strcpy(str2, ", add this");
//concatenating the string str2 to the string str1
strcat(str1, str2);
//displaying destination string
printf("String after concatenation: %s", str1);
return(0);
}
```

Output:

String after concatenation: This is my initial string, add this

Strcmp()

- The strcmp function compares two strings identified by the arguments and has a value 0 if they are equal.
- If they are not, it has the numeric difference between the first non matching characters in the strings.
- It takes the form:

`Strcmp(string1, string2);`

- **string1** – This is the first string to be compared.
- **string2** – This is the second string to be compared.
- This function return values that are as follows –
- if Return value < 0 then it indicates string1 is less than string2.
- if Return value > 0 then it indicates string2 is less than string1.
- if Return value $= 0$ then it indicates string1 is equal to string2.

```
#include <stdio.h>
#include <string.h>
int main()
{
char str1[] = "abcd", str2[] = "abCd", str3[] = "abcd";
int result;
// comparing strings str1 and str2
result = strcmp(str1, str2);
printf("strcmp(str1, str2) = %d\n", result);
// comparing strings str1 and str3
result = strcmp(str1, str3);
printf("strcmp(str1, str3) = %d\n", result);
return 0;
}
```

Output

- `strcmp(str1, str2) = 32`
- `strcmp(str1, str3) = 0`
- The first unmatched character between string `str1` and `str2` is third character. The ASCII value of 'c' is 99 and the ASCII value of 'C' is 67.
- Hence, when strings `str1` and `str2` are compared, the return value is 32.

strlen()

- This function counts and returns the number of characters in a string.
- It takes the form

```
n = strlen(string);
```
- Where n is an integer variable, which receives the value of the length of the string.
- The argument may be a string constant.
- The counting ends at the first null character.

Example

```
#include <stdio.h>
#include <string.h>
int main ()
{
    char str[50];
    int len;
    strcpy(str, "This is tutorialspoint.");
    len = strlen(str);
    printf("Length of |%s| is |%d|\n", str, len); return(0);
}
```

Output

- Length of |This is tutorialspoint.| is |23|

USER – DEFINED FUNCTIONS

C Programming

By

Mrs. R.Waheetha, MCA, M.Phil
Head,

Department of Computer Science
Holy Cross Home Science College
Thoothukudi

Unit – IV: FUNCTIONS

User-Defined functions:- Need for User-defined functions – Definition of functions – Return Values and their Types – Function Calls – Function Declaration – Category of functions – No Arguments and No return values – Arguments but No return Values – Arguments with return values – No arguments but a return a value – Recursion – Passing Arrays to functions – Passing Strings to functions – The Scope, Visibility and lifetime of a variables. Structures and Unions:- Defining a Structure – Declaring Structure Variables – Accessing Structure Members – Structure Initialization – Arrays of structures – Unions. (14L)

● **INDRODUCTION**

- C functions can be classified into two categories, namely, library functions and user defined functions.
- main is an example of user-defined functions.
- printf and scanf belong to the category of library functions.

NEED FOR USER-DEFINED FUNCTIONS

- main is a specially recognized function in C.
- Every program must have a main function to indicate where the program has to begin its execution.
- If a program is divided into functional parts, then each part may be independently coded and later combined into a single unit.
- In C, such subprograms are referred to as „functions“.
- In order to make use of a user-defined function, we need to establish three elements that are related to functions.
 1. Function definition.
 2. Function call.
 3. Function declaration.
- The function definition is an independent program module that is specially written to implement the requirements of the function.
- In order to use this function we need to invoke it at a required place in the program.
- This is known as the function call.
- The program (or a function) that calls the function is referred to as the calling program or calling function.
- The calling program should declare any function (like declaration of a variable) that is to be used later in the program. This is known as the function declaration or function prototype.

• **DEFINITION OF FUNCTIONS**

- A function definition, also known as function implementation shall include the following elements:
 1. Function name;
 2. Function type;
 3. List of parameters;
 4. Local variable declarations;
 5. Function statements; and
 6. A return statement.
- All the six elements are grouped into two parts, namely,
 - function header (First three elements); and
 - function body (second three elements).

- A general format of a function definition to implement these two parts is given below:

```
function_type function_name(parameter list)
{
    local variable declaration;
    executable statement1;
    executable statement2;
    . . . . .
    . . . . .
    return statement;
}
```

- The first line `function_type function_name(parameter list)` is known as the function header and the statements within the opening and closing braces constitute the function body, which is a compound statement.

- **Function Header**

- The function header consists of three parts: the function type (also known as return type), the function name and the formal parameter list. Note that a semicolon is not used at the end of the function header.

- **Name and Type**

- The function type specifies the type of value (like float or double) that the function is expected to return to the program calling the function.
- If the return type is not explicitly specified, C will assume that it is an integer type.
- If the function is not returning anything, then we need to specify the return type as void.
- Remember, void is one of the fundamental data types in C.
- It is a good programming practice to code explicitly the return type, even when it is an integer.
- The value returned is the output produced by the functions.
- The function name is any valid C identifier and therefore must follow the same rules of formation as other variable names in C.
- The name should be appropriate to the task performed by the function.
- However, care, must be exercised to avoid duplicating library routine names or operating system commands.

● **Formal Parameter List**

- The parameter list declares the variables that will receive the data sent by the calling program.
- They serve as input data to the function to carry out the specified task.
- Since they represent actual input values, they are often referred to as formal parameters.
- These parameters can also be used to send values to the calling programs.

● **Function Body**

- The function body contains the declarations and statements necessary for performing the required task.
- The body enclosed in braces, contains three parts, in the order given below
 1. Local declarations that specify the variables needed by the function.
 2. Function statements that perform the task of the function.
 3. A return statement that returns the value evaluated by the function.
- If a function does not return any value (like the printline function), we can omit the return statement.
- However, note that its return type should be specified as void. Again, it is nice to have a return statement even for void functions.

● **FUNCTION**

- Function is a self contained program performing a task.
- It has all qualities of a program.
- It is executed / called by main (or) another function, by calling the function name.
- It is also known as sub-program.
- Syntax

return data type fn. name (optional arguments)

{

 Function body

}


- function definition = function heading + fn. body

```
#include <stdio.h>
#include <conio.h>
int addition(int num1, int
    num2)
{
int sum;
    sum = num1+num2;
return sum;
}
int main()
{
    int var1, var2,res;
clrscr();
    printf("\nEnter number 1: ");
    scanf("%d",&var1);
```

```
    printf("\nEnter number 2: ");
    scanf("%d",&var2);
    res = addition(var1, var2);
    printf ("\nOutput: %d", res);
return 0;
}
```

Output:
Enter number 1: 100
Enter number 2: 120
Output: 220

CATEGORIES OF FUNCTIONS

- 
1. Function with No argument and No return values
 2. Function with argument and No return values
 3. Function with argument and return values
 4. Function with no argument but return value
 5. Function that return multiple values

Functions

```
graph TD; Functions[Functions] --> WithArguments[With Arguments]; Functions --> WithoutArguments[Without Arguments];
```

With Arguments

declared and defined with parameter list

values for parameter passed during call

Eg :

```
// declaration  
int sum (int x, int y);  
//call  
sum(10, 20);
```

Without Arguments

No parameters included.

No value passed during function call

Eg :

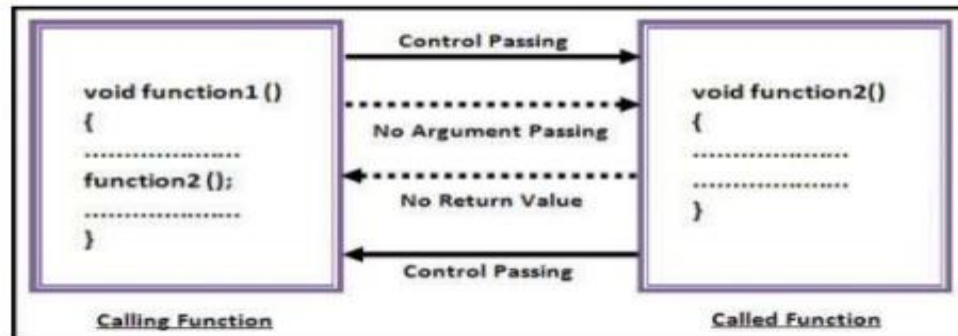
```
// declaration  
int display();  
// call  
display();
```

1. Function with No argument and No return values

- When a function has no arguments, it does not receive any data from the calling function.
- So there is no data transfer between calling and called function.

NO ARGUMENTS AND NO RETURN VALUES

- A function does not receive any data from the calling function. Similarly, it does not return any value.



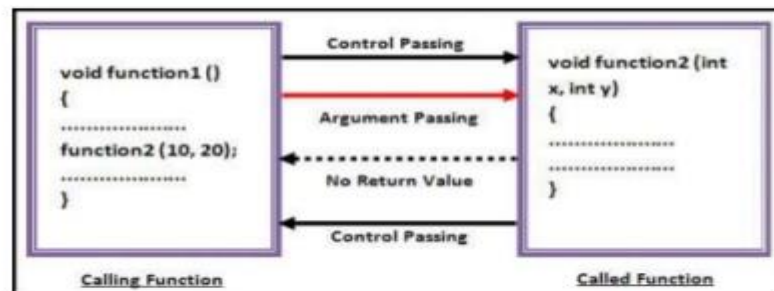

```
#include <stdio.h>
void area();
int main()
{
    area();
    return 0;
}
void area()
{
    int area, side;
    printf("Enter the side of square :");
    scanf("%d",&side);
    area = side * side;
    printf("Area of Square = %d",area);
}
```

2. Function with argument and No return values

- Here function will accept data from the calling function as there are arguments, however, since there is no return type nothing will be returned to the calling program.
- So it's a one-way type communication.

ARGUMENTS BUT NO RETURN VALUES

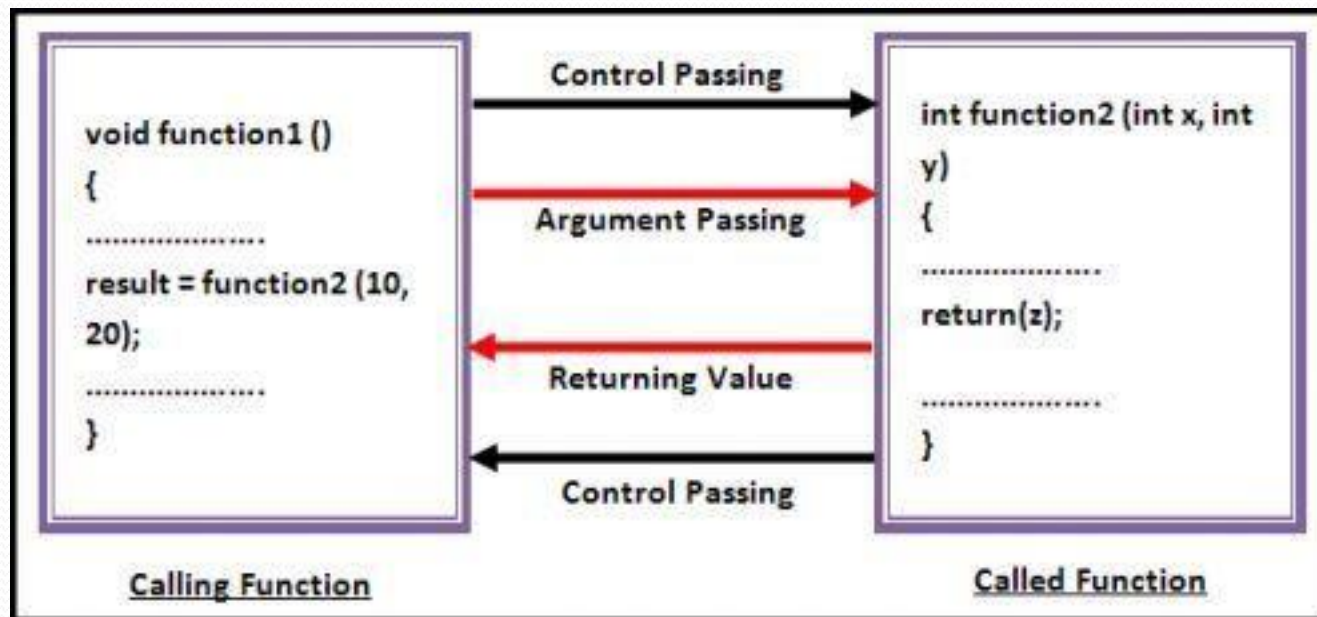
- The nature of data communication between the calling function and the called function with **arguments but no return values** is shown in the diagram



```
#include <stdio.h>
#include <conio.h>
void area( int side);
int main()
{
int side;
clrscr();
printf("Enter the side of square :");
scanf("%d",&side);
area(side);
return o;
}
void area(int side)
{
int area;
area = side * side;
printf("Area of Square = %d",area);
}
```

3. Function with argument and return values

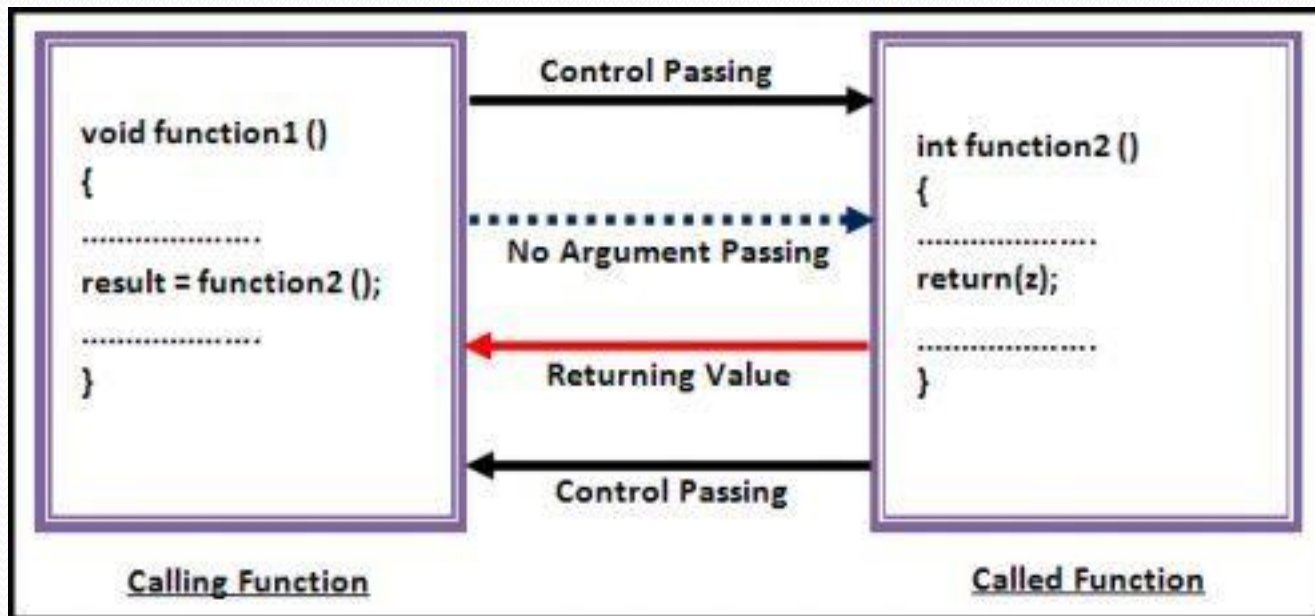
- This receives data from the calling function through arguments
- Data Type of the return value will depend upon the return type of function declaration.
- This type of user-defined function is called a fully dynamic function, and it provides maximum control to the end-user.
- Two-way data communication between function



```
#include<stdio.h>
int multiplication(int, int);
int main()
{
int a, b, multi;
printf("\n Please Enter two integer values \n");
scanf("%d %d",&a, &b);
multi = multiplication(a, b);
printf("\n Multiplication of %d and %d is = %d \n", a, b,multi);
return 0;
}
int multiplication(int a, int b)
{
int m;
m = a * b;
return (m);
}
```

4. Function with no argument but return value

- It will not take any argument but return a value to the calling function.
- The Data Type of the return value will depend upon the return type of function declaration.
- This type of function will return some value when we call the function from main()



```
#include<stdio.h>
int multiplication();
int main()
{
int m;
m = multiplication();
printf("\n Multiplication of a and b is = %d \n", Multi );
return o;
}
int multiplication()
{
int m, a = 20, b = 40;
m = a * b;
return (m;)
}
```

RECURSION

- A function calling itself is known as recursion.
- Recursion must have a terminating condition.
- Here statement with in body of the function calls the same function and same times it is called as circular definition.
- In other words recursion is the process of defining something in form of itself.

```
void recurse() {  
    ... ..  
    recurse();  
    ... ..  
}  
  
int main() {  
    ... ..  
    recurse();  
    ... ..  
}
```

The diagram illustrates the flow of execution. A blue arrow points from the `recurse();` line inside the `recurse()` function back to the start of the `recurse()` function, labeled "recursive call". Another blue arrow points from the `recurse();` line inside the `main()` function to the start of the `recurse()` function, labeled "function call".

```
return 5 * factorial(4) = 120  
└─ return 4 * factorial(3) = 24  
    └─ return 3 * factorial(2) = 6  
        └─ return 2 * factorial(1) = 2  
            └─ return 1 * factorial(0) = 1
```

javaTpoint.com

$1 * 2 * 3 * 4 * 5 = 120$

Fig: Recursion


```
#include <stdio.h>
#include <conio.h>
long int factorial(int n);
int main()
{
    int num;
    long int fact=0;
    printf("Enter an integer number: ");
    scanf("%d",&num);
    fact=factorial(num);
    printf("Factorial of %d is = %ld",num,fact);
    getch();
    return 0;
}
long int factorial(int n)
{
    if(n==1) return 1;
    return(n*factorial(n-1));
}
```

PASSING ARRAYS TO FUNCTIONS

- **One-Dimensional Arrays**

- Like the value of simple variables, it is possible to pass the value of an array to a function
- It is sufficient to list the name of the array, without any subscripts, and the size of the array as arguments.
- Example
 - `Float large(float a[], int s)`
 - The function `large` is defined to take two arguments, the array name and size of the array to specify the no.of elements in array.

```
#include<stdio.h>
#include<conio.h>
int add(int x[], int n);
void main()
{
    int x[20],n,i,s;
    clrscr();
    printf("\n Enter the value of n: ");
    scanf("%d",&n);
    printf("\n\nEnter the value of x\n");
    for(i=0;i<n;i++)
        scanf("%d",&x[i]);
    s = add(x,n);
    printf("\n\n The sum of n number = %d",s);
    getch();
}
int add(a[20],int n)
{
    int i,s1=0;
    for(i=0;i<n;i++)
        s1=s1+x[i];
    return(s1);
}
```

Three Rules to Pass an Array to a Function

1. The function must be called by passing only the name of the array.
2. In the function definition, the formal parameter must be an array type; the size of the array does not need to be specified.
3. The function prototype must show that the argument is an array.

• Two-Dimensional Array

- We can also pass multi-dimensional array to functions.
- The rules are
 - The function must be called by passing only the array name
 - We must indicate that the array has two-dimensions by including two set of brackets
 - The size of the second dimension must be specified.
 - The prototype declaration should be similar to the function header.

Passing string to function

- The strings are treated as character array and so the rules for passing strings to function are similar as passing array to function.
- Rules
 - The string to be passed must be declared as a formal argument of the function when it is defined.

```
void dis(char name[])  
{  
}
```

- The function prototype must show that the argument is a string.

```
void dis(char name[]);
```

- A call to the function must have a string array name without subscripts as its actual argument

```
dis(name); where name is string array in the calling  
function
```

SCOPE, VISIBILITY AND LIFE TIME OF VARIABLES

- Storage class relevant to functions are
 1. Automatic variables.
 2. External variables.
 3. Static variables.
 4. Register variables.
- The scope of variable determines over what region of the program a variable is actually available for use („active“).
- Longevity refers to the period during which a variable retains a given value during execution of a program („alive“).
- So longevity has a direct effect on the utility of a given variable.
- The visibility refers to the accessibility of a variable from the memory.
- The variables may also be broadly categorized, depending on the place of their declaration, as internal (local) or external (global).
- Internal variables are those which are declared within a particular function, while external variables are declared outside or any function.

Automatic variables

- Automatic variables are declared inside a function in which they are to be utilized.
- They are created when the function is called and destroyed automatically when the function is exited, hence the name automatic.
- Automatic variables are therefore private (or local) to the function in which they are declared.
- Because of this property, automatic variables are also referred to as local or internal variables.

```
main( )  
{  
int number;  
-----  
}
```

We may also use the keyword auto to declare automatic variables explicitly.

```
main( )  
{  
auto int number;  
-----  
}
```


- Variables that are both alive and active throughout the entire program are known as external variables.
- They are also known as global variables.
- Unlike local variables, global variables can be accessed by any function in the program. External variables are declared outside a function.
- For example, the external declaration of integer number and float length might appear as.
- `int number;`
- `float length = 7.5;`
- Manonmaniam Sundaranar University, Directorate of Distance & Continuing Education, Tirunelveli.
- 62
- `main()`
- `{`
- `-----`
- `}`
- `function1()`
- `}`
- `-----`
- `}`
- `function2()`
- `{`
- `-----`
- `}`

Static Variables

- As the name suggests, the value of static variables persists until the end of the program.
- A variable can be declared static using the keyword static like
- **static int x;**
- static float y;
- A static variable may be either an internal type or an external type depending on the place of declaration.
- Internal static variables are those which are declared inside a function.
- The scope of internal static variables extend up to the function in which they are defined.
- Therefore, internal static variables are similar to auto variables, except that they remain in existence (alive) throughout the remainder of the program.
- Therefore, internal static variables can be used to retain values between function calls.
- An external static variable is declared outside of all functions and is available to all the functions in that program.
- The difference between a static external variable and a simple external variable is that the static external variable is available only within the file where it is defined while the simple external variable can be accessed by other files.

Register variables

- We can tell the compiler that a variable should be kept in one of the machine's registers, instead of keeping it in the memory (where normal variables are stored).
- Since a register access is much faster than a memory access, keeping the frequently accessed variables (e.g., loop control variables) in the register will lead to faster execution of programs.
- This is done as follows:
- `register int count;`

Storage class	Where declared	Visibility (Active)	Lifetime (Alive)
None	Before all functions in a file (may be initialized)	Entire file plus other files where variable is declared with extern	Entire program (Global)
extern	Before all functions in a file (cannot be initialized) extern and the file where originally declared as global.	Entire file plus other files where variable is declared	Global
static	Before all functions in a file	Only in that file	Global
None or auto	inside a function (or a Block)	Only in that function or block	Until end of function of block
register	inside a function or block	Only in that function block	Until end of function or block
static	Inside a function	Only in that function	Global

STRUCTURES AND UNIONS

Structures

- Structure is a convenient tool for handling a group of logically related data items.
- It is a collection of heterogenous data items.
- Each data item is known as structure member.
- Defining a sturcture

```
struct student
{
    char name[25];
    int rno, m1,m2,m3,m4,total;
    float avg;
};
```

- Struct declares a structure to hold the details of name,rno,m1 etc.
- These fields are called structure elements or members.
- Each member belong to the datatype called student.
- Student is the name of the structure and is called as structure tag.

- The general format of structure is

- ```
struct tag name
{
 data type member1;
 data type member1;

}

```

1. The template is terminated with a semicolon
2. Each member is declared independently.
3. The tag can be used to declare structure variables of its type.

# Declaring Structure variable

- A structure variable declaration is similar to declaration of any other data type.
- It includes the following elements
  - The keyword struct
  - The structure tag name
  - List of variable name separated by commas
  - A terminating semicolon.
  - Example

```
struct student s1, s2;
```



# Accessing Structure Members

- We can access and assign values to the members of a structure in a number of ways.
- They should be linked to structure variables in order to make them meaningful members.
- The link between a member and a variable is established using the member operator '.' which is also known as dot operator or period operator.
- Example

```
 struct student
 {
 char name[25];
 int rno, m1,m2,m3,m4,total;
 float avg;
 };
 main()
 {
 struct student s1;

 s1.rno= 12344;
 s1.m1=68;
 s1.m2=87;
 s1.m3=83;
 s1.m4 = 60;
 s1.total=s1.m1+s1.m2+s1.m3+s1.m4;
 s1.avg=s1.total/4;
 }
```

# Structure Initialization

- Structure can be initialized timed at compile time.

- Example

```
struct stud
{
 char name[20];
 int rno,age;
}
main()
{
 struct stud s1 = {"Kala", 53421, 18};
}
```

- Initialization of structure must have following statements
  - The keyword struct
  - The structure tag name
  - The name of the variable to be declared
  - The assignment operator =
  - A set of values for the members of the structure variables seperated by commas and enclosed in braces
  - A terminating semicolon.

# Rules for Initializing structure

- We cannot initialize individual member inside the structure template.
- The order of the values in braces must match the order of members in the structure definition.
- It is permitted to have partial initialization.
- The unassigned members will be assigned default values as follows
  - Zero for integer and floating point
  - '\0' for character and string

# Array of Structure

- Structures are used to describe a number of related variables.
- We can declare an array of structure.
- Each element of array represent a structure variable.
- An array of structures is stored inside the memory in the same way as a multi-dimensional array.
- Example
  - `Struct stud s[10];`
  - Here array s consist details of 10 students.

```
struct student
```

```
{
 char name[25];
 int rno, m1,m2,m3,m4,total;
 float avg;
};
main()
{
 int i, n;
 struct student s1[10];

 clrscr();
 printf("\n Enter th no.of
students")'
 scanf("%d",&n);
 printf("\n Enter the details of
students like name,m1,m2,m3,m4");
for(i=0;i<n;i++)
{
 scanf("%s",&s1[i].name);

 scanf("%d%d%d%d",&s1[i].m1,&s1
[i].m2,&s1[i].m3,&s1[i].m4);
```

```
s1[i].total=s[i].m1+s1[i].m2+s1[i].m
3+s1[i].m4;
 s1[i].avg=s1[i].total/4;
 }
for(i=0;i<n;i++)
{
 printf("\nName = %s",s1[i].name);
 printf("\n M1=%d",&s1[i].m1);
 printf("\n M2=%d",&s1[i].m2);
 printf("\n M3=%d",&s1[i].m3);
 printf("\n M4=%d",&s1[i].m4);
 printf("\n Total =%d",&s1[i].total);
 printf("\n Average=%f",&s1[i].avg
 }
 getch();
}
```

# UNIONS

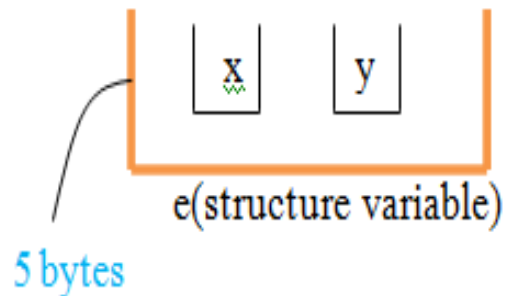
- Unions follow the same syntax as structures.
- There is major distinction between them in terms of storage.
- In structures, each member has its own storage location, whereas all the members of a union use the same location.
- This implies that, although a union may contain many members of different types, it can handle only one member at a time.
- Example

```
union item
{
 int m;
 float x;
 char c;
} code;
```

- This declares a variable code of type union item.
- The union contains three members, each with a different data type.
- However, we can use only one of them at a time.

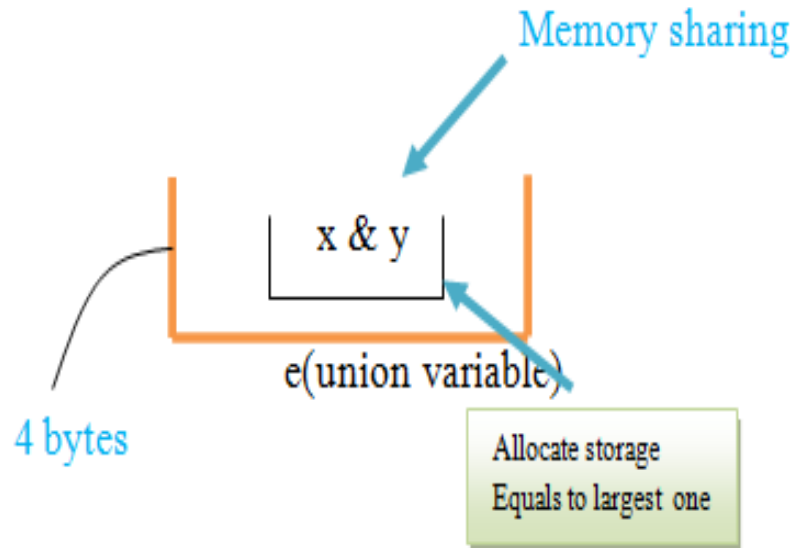
## Structure

```
struct Emp
{
 char x; //size 1 byte
 float y; //size 4 byte
}e;
```



## Unions

```
union Emp
{
 char x;
 float y;
}e;
```



- The compiler allocates a piece of storage that is large enough to hold the largest variable type in the union.
- To access a union member union variable member eg.
- code.m
- code.x
- code.c
- Are all valid member variables.
- During accessing, we should make sure that we are accessing the member whose value is currently stored.
- For example, the statements such as
- code.m = 379;
- code.x = 7859.36;
- printf(“%d”, code.m) ;
- Would produce erroneous output(which is machine dependent.)
- Union may be used in all places where a structure is allowed.
- Unions may be initialized when the variable is declared.
- But it can be initialized only with a value of the type as the first union member.
- union item abc = { 100 } ; is valid.



# POINTERS

C Programming

By

Mrs. R.Waheetha, MCA, M.Phil  
Head,

Department of Computer Science  
Holy Cross Home Science College  
Thoothukudi

# Unit – V: POINTERS AND FILES

- Pointers:- Understanding pointers – Accessing the Address of a Variable – Declaring Pointer Variables – Accessing a variable through its pointer – Pointer Expressions –Pointers as function arguments. File Management in C:- Defining and Opening a file – Closing a File – Input/output Operations on files – Error Handling during I/O Operations. (12L)

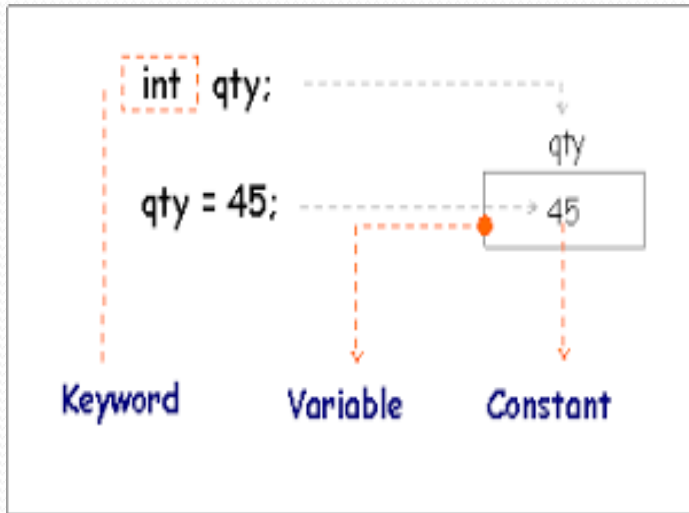
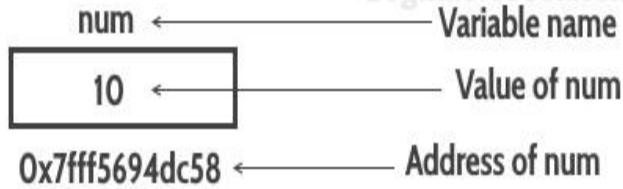
- Pointer is derived data type in C pointers contain memory addresses as their values.
- Pointers can be used to access and manipulate data stored in the memory.
- Benefits of pointers
  - Pointers are more efficient in handling arrays and data tables.
  - It can be used to return multiple values from a function via function arguments.
  - It permit reference to function.
  - The use of pointer arrays to character strings which helps in saving data storage space.
  - It supports dynamic memory management.
  - It is an efficient tool for manipulating structure, list, stacks etc.
  - It reduce the length and complexity of program.
  - It increase the execution speed .

# Understanding Pointers

- Whenever we declare a variable, the system allocates, somewhere in the memory, an appropriate location to hold the value of the variables.
- Consider the following statement  

```
int quantity = 179;
```
- This statement instructs the system to find a location for the integer variable quantity and puts the value 179 in that location.
- Let us assume that the system has chosen the address location 5000 for quantity.
- During execution of the program, the system always associates the name quantity with the address 5000.
- Since memory addresses are simply numbers, they can be assigned to some variables that can be stored in memory, like any other variable.
- Such variables that hold memory addresses are called pointer variables.
- A pointer variable is, therefore, nothing but a variable that contains an address, which is a location of another variable in memory.

Beginnersbook.com



```
char str[6] = "Hello";
```

|         |      |      |      |      |      |      |
|---------|------|------|------|------|------|------|
| index   | 0    | 1    | 2    | 3    | 4    | 5    |
| value   | H    | e    | l    | l    | o    | \0   |
| address | 1000 | 1001 | 1002 | 1003 | 1004 | 1005 |

variable ptr  
 value 1000  
 address 8000

| Variable  | Address | Memory locations |
|-----------|---------|------------------|
|           | 0       |                  |
|           | 2       |                  |
|           | ....    |                  |
| p         | 60000   |                  |
|           | ....    |                  |
| k         | 65534   |                  |
| variables | address | value            |

# Accessing the address of a Variable

- The actual location of a variable in the memory is system dependent and the address of a variable is not known immediately.
- But it can be determined with the help of the operator &.
- Example
  - `P = &qty`
  - Will assign the address to the variable p
- The operator & is called as ‘address of’.
- The &operator can be used only with a simple variable or an array element.

# Declaring Pointer Variables

- Pointer variables contain addresses that belong to separate data type.
- They must be declared as pointers before we use it.
- The declaration of pointer is as follows
  - `data_type *pointer_name;`
- This tells the compiler three things about the pointer variable `pointer_name`
  - The asterisk(\*) tells that the variable `pointer_name` is a pointer variable.
  - `Pointer_name` needs a memory location.
  - `Pointer_name` points to a variable of type `data_type`.
  - Example
    - `int *p;`
    - `Float *a;`

# Accessing a variable through its Pointer

- We can access the value of the pointer variable using an unary operator \* (asterisk).
- It is known as indirection operator.
- Another name for indirection operator is dereferencing operator.
- Example

```
int qty, *p,n;
qty = 179;
p = &qty;
n = *p;
```

- Qty and n are integer variables and p is pointer variable .
- Second line assigns 179 to qty.
- Third line assigns the address of qty to p.
- The fourth line returns the value of the variable qty.
- '\*' can be remembered as 'value of address'



# Pointer Expression

- Pointer variables can be used in expression.
- If p1 and p2 are declared and initialized pointer then the following is valid.
  - $Y = *p1 * *p2;$
  - $Sum = sum + *p1;$
  - $*p1 = *p2 + 10;$
- C allows us to add integers or subtract integers from pointers.
- $P1 + 4$ ,  $p2 - 2$ ,  $p1 - p2$  are allowed.

# POINTERS AS FUNCTION ARGUMENTS

- When we pass addresses to a function, the parameters receiving the addresses should be pointers.
- The process of calling a function using pointers to pass the addresses of variables is known as “call by reference”
- The function `exchange()` receives the addresses of the variables `x` and `y` and exchanges their contents.

- Program

```
void exchange (int *, int *); /* prototype */
main()
{
int x,y;
x = 100;
y = 200;
printf (Before exchange : x = %d y = %d\n\n", x,x);
exchange(&x, &y);
printf("After exchange : x = %d y = %d\n\n", x,y);
}
exchange (int *a, int *b)
{
int t;
t = *a; /*Assign the value at address a to t */
*a = *b; /* put b into a */
b = t; / put t into b */
}
```

- Output

- Before exchange: x = 100 y = 200
- After exchange: x = 200 y = 100

# **FILE MANAGEMENT IN C**

# INTRODUCTION

- A file is a place on the disk where a group of related data is stored.
- Basic file operations.
  - Naming a file,
  - Opening a file,
  - Reading data from a file,
  - Writing data to a file, and
  - Closing a file.

# DEFINING AND OPENING A FILE

- To store data in a file in the secondary memory, we must specify certain things about the file, to the operating system.

- They include: High Level I/O Functions

- **Function name**

- **Operation**

- fopen()

Creates a new file for use.

- fclose()

Opens an existing file for use.

- getc()

Close a file which has been opened for use.

- putc()

Reads a character from a file.

- fprintf()

Writes a character to a file.

- getw()

Writes a set of data values to a file.

- putw()

Reads a set of data values from a file.

- File include the following

1. Filename.
2. Date structure.
3. Purpose.

- Filename is a string of characters that make up a valid filename for the operating system.
- It contain two parts primary name an optional period with the extension.
- Examples:
  - input.data
  - Student.c
- Data structure of a file is defined as FILE in the library of standard I/O function definitions.
- FILE is a defined data type.
- General format declaring and opening a file:

```
FILE *fp;
fp = fopen("filename", "mode") ;
```
- The first statement declares the variable fp as a “pointer to the data type FILE”,.
- The second statement opens the file named filename and assigns an identifier to the FILE type pointer fp.
- This pointer, which contains all the information about the file is subsequently used as a communication link between the system and the program.

- Mode can be one of the following:
- r open the file for reading only.
- w open the file for writing only.
- a open the file for appending (or adding) data to it.

```
FILE *p1, *p2;
```

```
p1 = fopen("data", "r");
```

```
p2 = fopen("results", "w");
```

- Additional modes of operation
  - r+ The existing file is opened to the beginning for both reading and writing
  - w+ Same as w except both for reading and writing.
  - a+ Same as a except both for reading and writing.



# CLOSING A FILE

- A file must be closed as soon as all operations on it have been completed.

```
fclose(file_pointer);
```

```
FILE *P1, *P2;
```

```
p1 = FOPEN("INPUT", "w");
```

```
p2 = fopen("OUTPUT", "r");
```

```
fclose(p1);
```

```
fclose(p2);
```

# Input / Output operation on Files

- The simplest file I/O functions are `getc` and `putc`.
- These are analogous to `getchar` and `putchar` function.
- It handles one character at a time.
- A file is opened with `w` and file pointer `fp1`.
- The statement
  - `putc(c, fp1);`
- Writes the character contained in the character variable `c` to the file pointer `fp1`.
- `getc` is used to read a character from that file which is opened in read mode.
- `c = getc(fp1);`
- The file pointer moves by one character position for every operation of `getc` or `putc`.
- The `getc` will return an end-of-file marker EOF, when the end of the file has been reached.

# The getw and putw Functions

- The getw and putw are integer-oriented functions.
- They are similar to the getc and putc functions and are used to read and write integer values.
- These functions would be useful when we deal with only integer data
- The general forms of getw and putw are:

`putw(integer,fp);`

`getw(fp) ;`

- The function `fprintf` and `fscanf` perform I/O operations that are similar to `printf` and `scanf`.

- The general form of `fprintf` is

```
fprintf(fp, "control string", list);
```

- Where `fp` is a file pointer associated with a file that is opened.
- The control string contains output specifications for the items in the list.
- The list may include variables, constants and string.

- Example

```
fscanf(f1, "%s%d%f", name, age, ht);
```

- The general form of `fscanf` is

```
fscanf(fp, "control string", list);
```

- This statement helps in reading the items in the list from the file specified by `fp`, according to the specifications contained in the control string.

- Example

```
fscanf(f2, "%s%d", item, qty);
```

- `fscanf` returns the number of items that are successfully read.
- When the end of file is reached, it returns the value EOF.

# ERROR HANDLING DURING I/O OPERATIONS

- It is possible that an error may occur during I/O operations on a file.
- Typical error situations include:
  1. Trying to read beyond the end-of-file mark.
  2. Device overflow.
  3. Trying to use a file that has not been opened.
  4. Trying to perform an operation on a file, when the file is opened for another type of operation.
  5. Opening a file with an invalid filename.
  6. Attempting to write to a write-protected file.
- We have two status-inquiry library functions;
- `feof` and `ferror` that can help us detect I/O errors in the files.

- The feof function can be used to test an end of file condition.
- It takes a FILE pointer as its only argument and returns a nonzero integer value if all of the data from the specified file that has just been opened for reading, then the statement

```
if(feof(fp))
```

```
 printf("End of data.\n");
```

- The ferror function reports the status of the file indicated.
- It also takes a FILE pointer as its argument and returns a nonzero integer if an error has been detected up to that point, during processing.
- It returns zero otherwise.
- The statement

```
if(ferror(fp) != 0
```

```
 printf("An error has occurred.\n"))
```

- Would print the error message, if the reading is not successful.