UNIT – 3
Fundamentals of C

FYBTECH

Subject : **Programming and Problem Solving**

Unit3 Contents

- **Introduction to C:** Fundamentals of C-Programming, Data types, Constants, Variables, Operators, Expression, Pre-processor directives. Data Input and Output.
- **Control Structures:** Structure of C program, Coding conventions, Decision making, Control Structures- Iterative, break and continue statements. Array- Single and Multidimensional arrays. Strings.

History and Features of C

- Dennis Ritchie is the creator of C
  - Created at Bell Laboratories
- Portable Language
  - C language is machine independent. Source Code written using C can be compiled on any machine(i.e. platform independant)
- Structured Language
  - problem is solved using a divide and conquer approach

Program Structure

A sample C Program

```
#include<stdio.h>
int main()
{
    int a; //variable declaration
    --other statements
}
```

Header Files

- The files that are specified in the include section is called as header file
- These are precompiled files that has some functions defined in them
- The functions can be called in the program by supplying parameters
- Header file is given an extension .h
- C Source file is given an extension .c

What is the purpose of header file in C?

- A header file is a file with extension .h which contains C function declarations and macro definitions to be shared between several source files.
- Header file is used in the program by including it, with the C preprocessing directive ' #include '.

Main Function

- This is the entry point of a program
- When a file is executed, the start point is the main function
- From main function the flow goes as per the programmers choice.
- There may or may not be other functions written by user in a program
- Main function is compulsory for any c program

Writing the first program

```
#include<stdio.h>  ←———— header file
int main()    ←———— main function
{
    printf(“Hello”);
    return 0;
}
```

- This program prints Hello on the screen when we execute it

Running a C Program

- Type a program
- Save it
- Compile the program – This will generate an exe file (executable)
- Run the program (Actually the exe created out of compilation will run and not the .c file)
- In different compiler we have different option for compiling and running. We give only the concepts.

Comments

- Comments are used to document programs and improve readability
- In C a comment will start with /* and ends with */

Syntax: /* Comments */

```
//This is a single line comment
/* This is a multi line
  comment in C */
/***************************************************************
* This style of commenting is used for functions
***************************************************************/
```

- Only C style comments should be used

Indentation of Code

- **Indentation** is the practice by Software Engineers to use spaces or tabs consistently in every line of code to group lines together based on their scope for easy readability
- An indented code looks better and can be understood easily
- The code in any line should not exceed 80 columns

Compiler and Linker Errors

- If a program does not follow the syntax of a language then the compiler raises errors

Example: Missing semicolon

- When the linker is not able to find a piece of code the linker errors are generated

Example: Variable or function referenced, but not defined anywhere in code

Data Types

- Data types determine the following:
  - Type of data stored
  - Number of bytes it occupies in memory
  - Range of data
  - Operations that can be performed on the data
- C supports the following data types:
  - int – for storing whole numbers
  - char – for storing character values, represents a single character
  - float – for storing fractional values
  - double – for storing fractional values

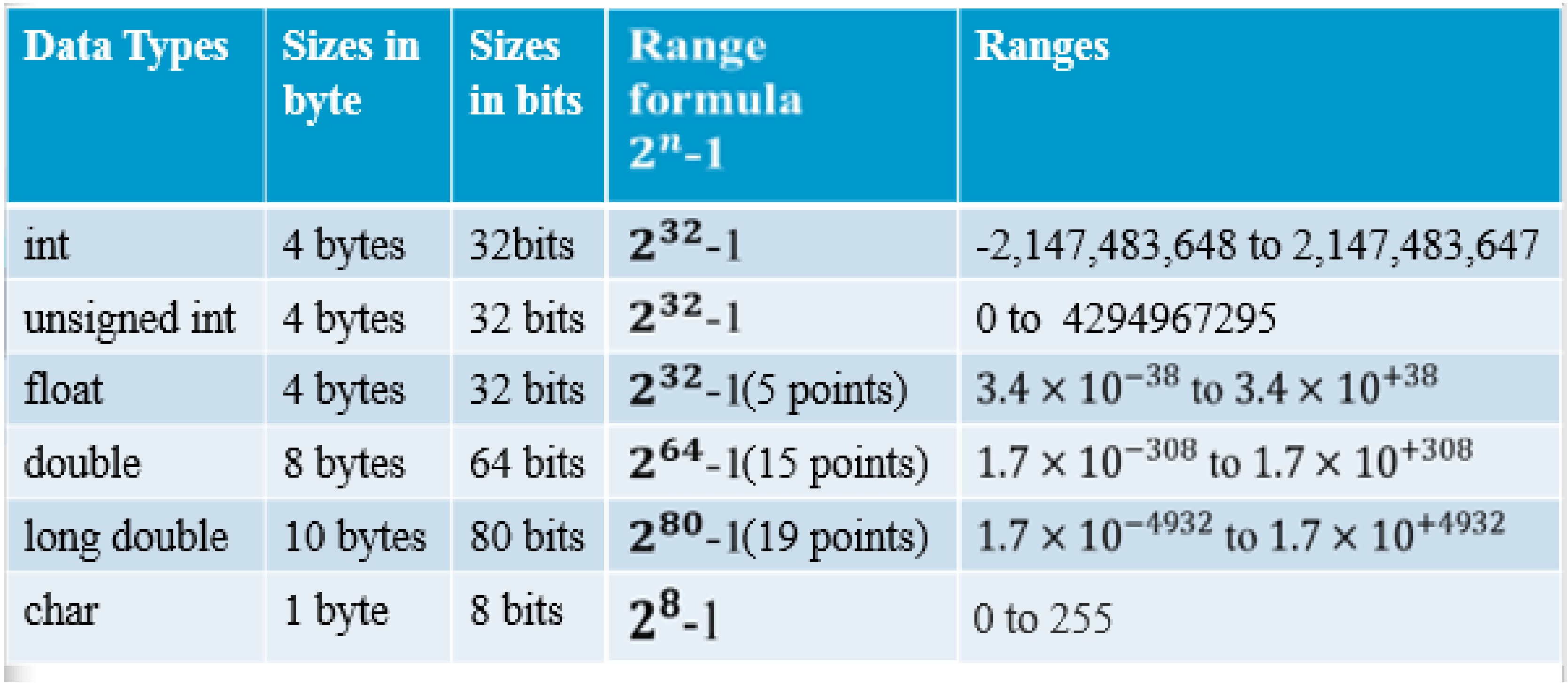
Note: float can store up to 6 digits of precision

double can store up to 12 digits of precision

Data Type

- C supports the following modifiers along with data types:
  - **short**
  - **long**
  - **signed**
  - **unsigned**

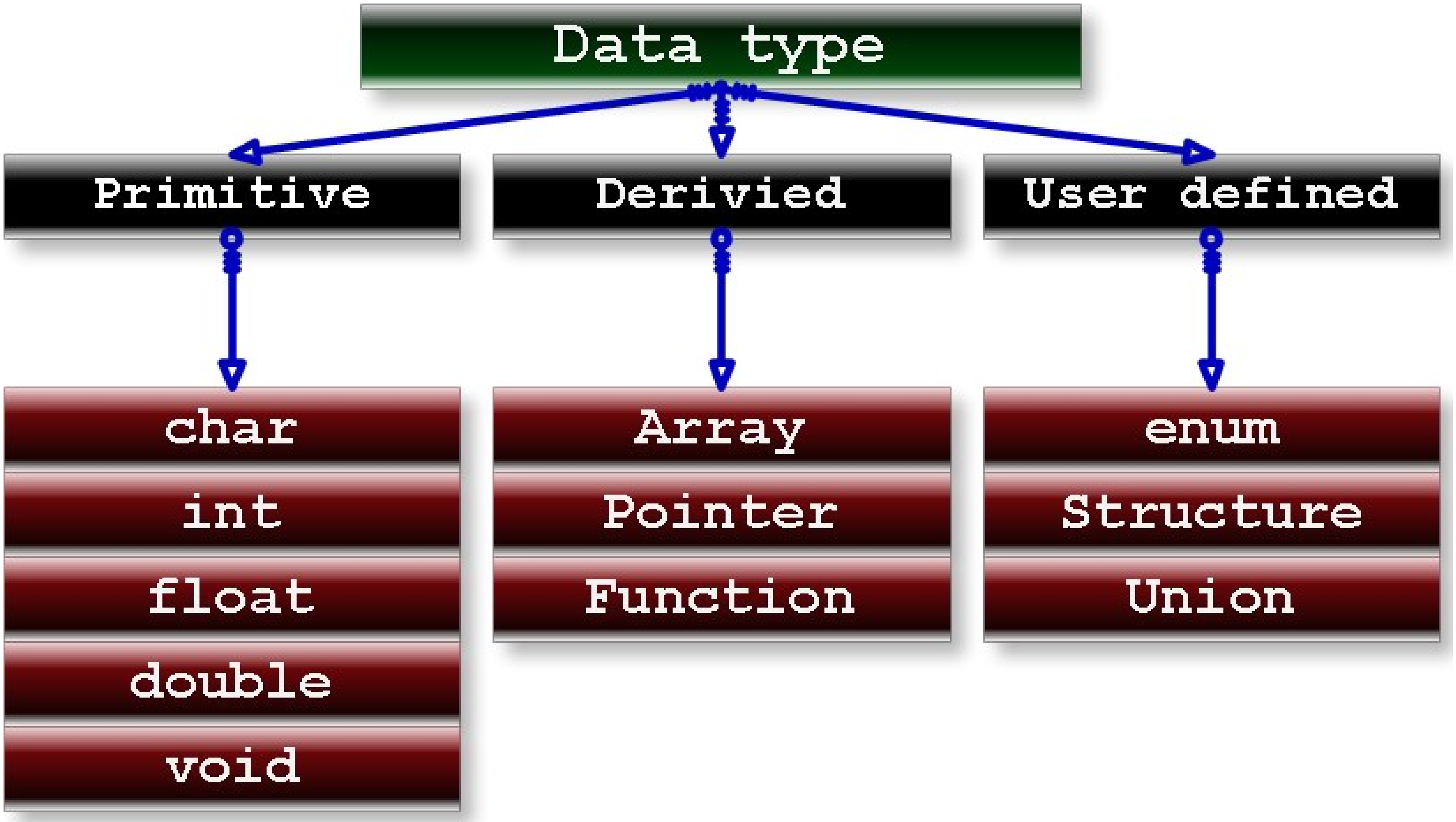
Range of Data Types



Data Types	Sizes in byte	Sizes in bits	Range formula 2^n-1	Ranges
int	4 bytes	32bits	2^{32}-1	-2,147,483,648 to 2,147,483,647
unsigned int	4 bytes	32 bits	2^{32}-1	0 to 4294967295
float	4 bytes	32 bits	2^{32}-1(5 points)	3.4×10^{-38} to $3.4 \times 10^{+38}$
double	8 bytes	64 bits	2^{64}-1(15 points)	1.7×10^{-308} to $1.7 \times 10^{+308}$
long double	10 bytes	80 bits	2^{80}-1(19 points)	1.7×10^{-4932} to $1.7 \times 10^{+4932}$
char	1 byte	8 bits	2^{8}-1	0 to 255

Note: Number of bytes and range given to each data type is platform dependent

Data Types in C



Constants

- A Constant is a value that never changes during program execution.

- Constants are given name and are referred by the given name.

Ex. PI=3.142

Variables

Hold the data in your program

- A variable is an entity that can change during program execution

Rules for Variable Names

- The first character in variable name must be an alphabet
- No other special character except underscore is allowed
- No blanks or commas are allowed
- Variable names are case sensitive
- Keywords cannot be used as variable names

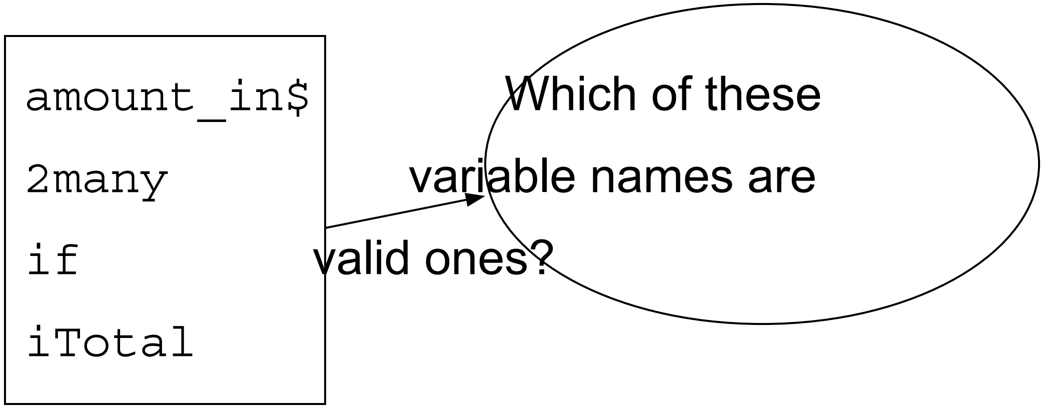
Syntax : datatype varname;

Variables (cont’d)

Ex :

int a;   //-----variable declaration

int a=10; //----var definition



```
amount_in$
2many
if
iTotal
```

Which of these variable names are valid ones?

Variables

- Variables are data that will keep on changing
- Declaration

  ```
  <<Data type>> <<variable name>>;
  int a;
  ```
- Definition

  ```
  <<varname>>=<<value>>;
  a=10;
  ```
- Usage

  ```
  <<varname>>
  a=a+1;    //increments the value of a by 1
  ```

Declaration of Variables

Syntax:

data-type [variable-name list]

data-type variable-name1 , variable-name2, … ;

Example:

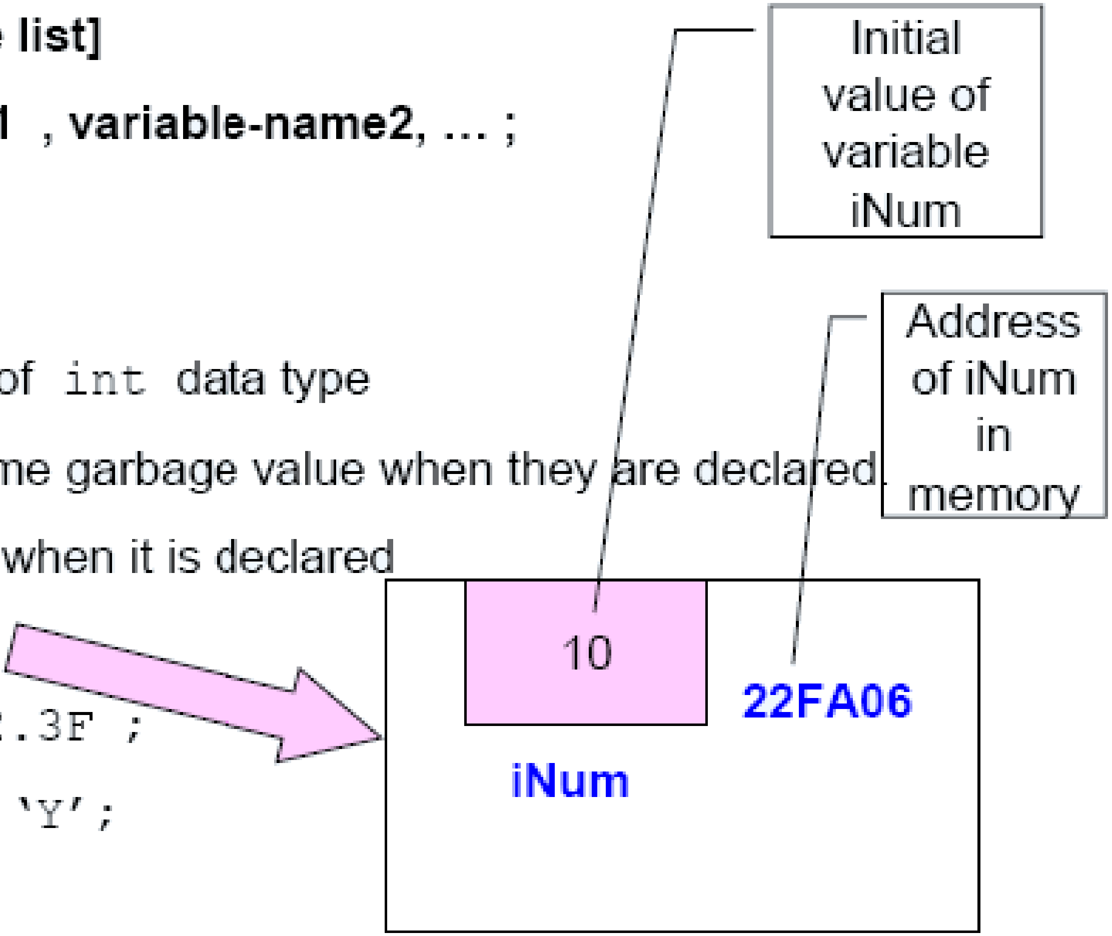
```
int iNum;
```

declares a variable of `int` data type

The variables will contain some garbage value when they are declared.

A variable can be **initialized** when it is declared

```
int iNum = 10;
float fData = 2.3F ;
char cChoice = 'Y';
```



Keywords

- **Keywords** are predefined, reserved words used in programming that have special meanings to the compiler.
- **Keywords** are part of the syntax and they cannot be used as an identifier.

Standard ANSI C Keywords

auto
break
case
char
const
continue
default
do
double
else
enum
extern
float
for
goto
volatile

if
int
long
register
return
short
signed
sizeof
static
struct
switch
typedef
union
unsigned
void
while

Operators In C

- Assignment operator **(=)**
- Arithmetic operators **(+, -, *, /, %)**
- Relational operators **(>, >=, <, <=, == , !=)**
- Logical operators **(!, &&, | |)**
- Address operator **(&)**
- Increment and Decrement operators **(++, --)**
- Compound Assignment Operators **(=, +=, -=, /=, *=, %=)**
- **sizeof** operator

Use of Modulus (%) Operator

- Used to find the remainder after integer division
- The operands that are supplied to this operator should always be integers
- The operator returns an integer value
- Using ‘float’ for any of the operands will result in a compiler error

Example:

Remainder = Number % 4 ;

If the variable ‘Number’ has a value 21, then the resultant value in ‘Remainder’ will be 1

Precedence of Arithmetic Operators

Operator Priority

* , / and % Highest

+ and - Lowest

The expression that is written within parenthesis is given highest priority

Evaluate the following expression:

Using a = 5, b = 3, c = 8 and d = 7

b + c / 2 – (d * 4) % a

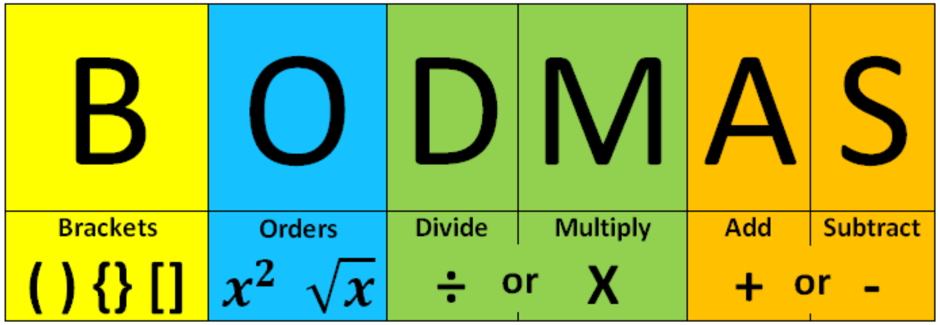
b + c / 2 – 28 % a

b + 4 – 28 % a

b + 4 – 3

7 – 3

4



B	O	D	M	A	S
Brackets	Orders	Divide	Multiply	Add	Subtract
( ) {} []	x^2 $\sqrt{x}$	÷ or	X	+ or	-

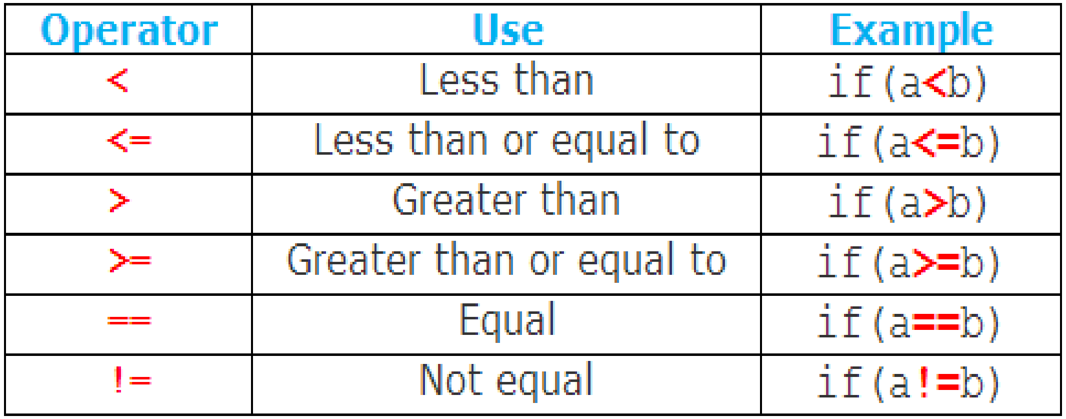
Relational Operators

- Used to compare two values and also called as **comparison operators**
- Expressions that contain relational operators are called as **relational expressions**
- A relational operator returns either zero or a non-zero value
- If the expression is true then it returns a non-zero value (>0)
- If the expression is false then it returns zero

•**Example:**

1500 > 700 returns 1 (true)

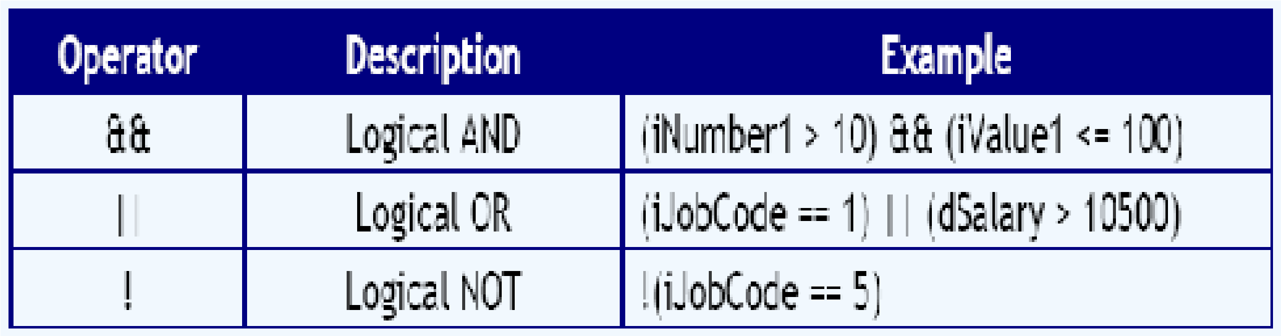
1500 < 700 returns 0 (false)



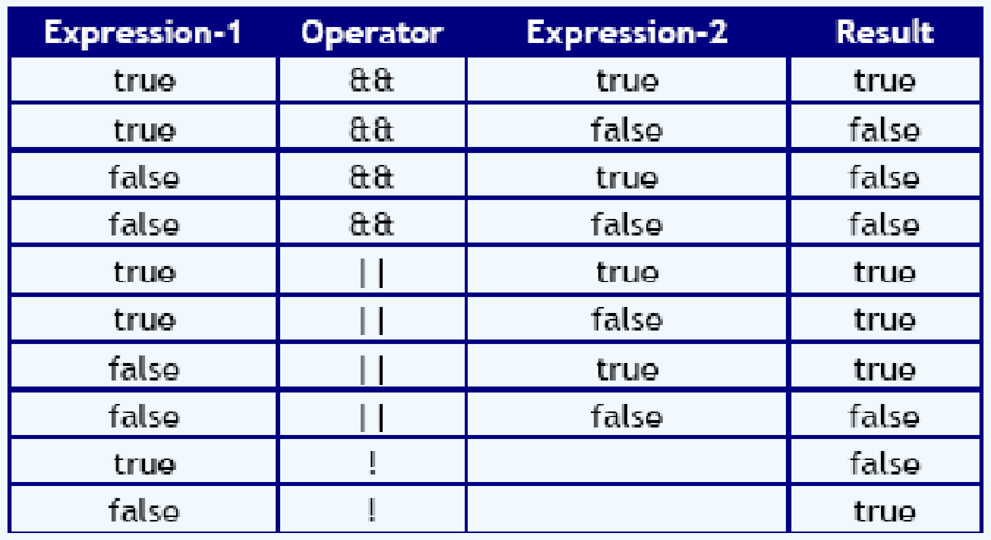
Operator	Use	Example
<	Less than	if(a<b)
<=	Less than or equal to	if(a<=b)
>	Greater than	if(a>b)
>=	Greater than or equal to	if(a>=b)
==	Equal	if(a==b)
!=	Not equal	if(a!=b)

Logical Operators

- Used to combine two or more relational expressions
- An expression involving logical operators is called as a **logical expression**



Operator	Description	Example
&&	Logical AND	(iNumber1 > 10) && (iValue1 <= 100)
||	Logical OR	(iJobCode == 1) || (dSalary > 10500)
!	Logical NOT	!(iJobCode == 5)



Expression-1	Operator	Expression-2	Result
true	&&	true	true
true	&&	false	false
false	&&	true	false
false	&&	false	false
true	||	true	true
true	||	false	true
false	||	true	true
false	||	false	false
true	!		false
false	!		true

Increment and Decrement Operators

- Operators ++ and -- are called as increment and decrement operators
- These operators increment or decrement the variable’s value by 1
- They are also called as *unary operators* because they have only one operand to operate
- If the operator is used before the operand, it is *prefix* and if the operator is used after the operand, it is *postfix*

Example:

++ Value and -- Value is called as *prefix*

Value++ and Value-- is called as *postfix*

Difference Between Prefix and Postfix

- Prefix operator first increments / decrements and then makes the assignment

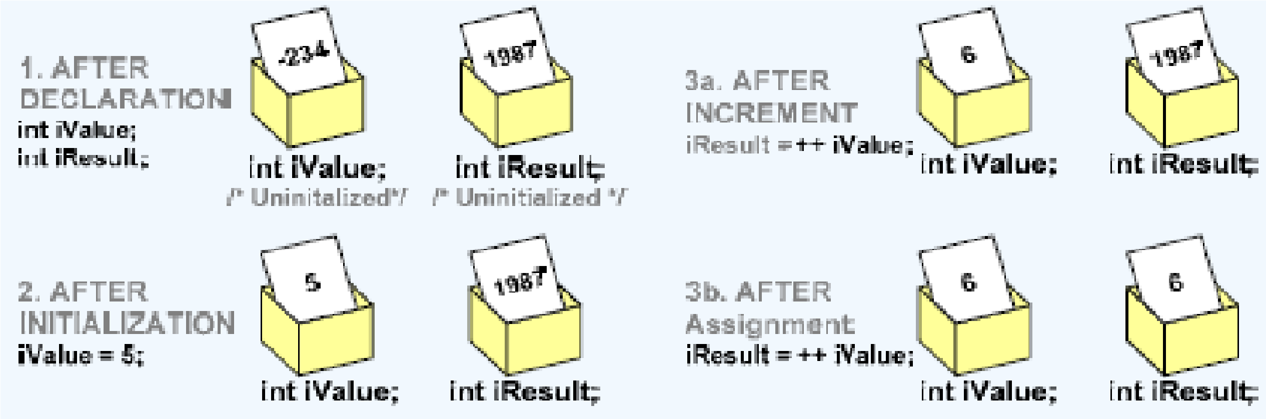
Example:

int iValue;

int iResult;

iValue = 5;

iResult = ++iValue;



- Post fix operator makes the assignment and then increments/decrements the

Value

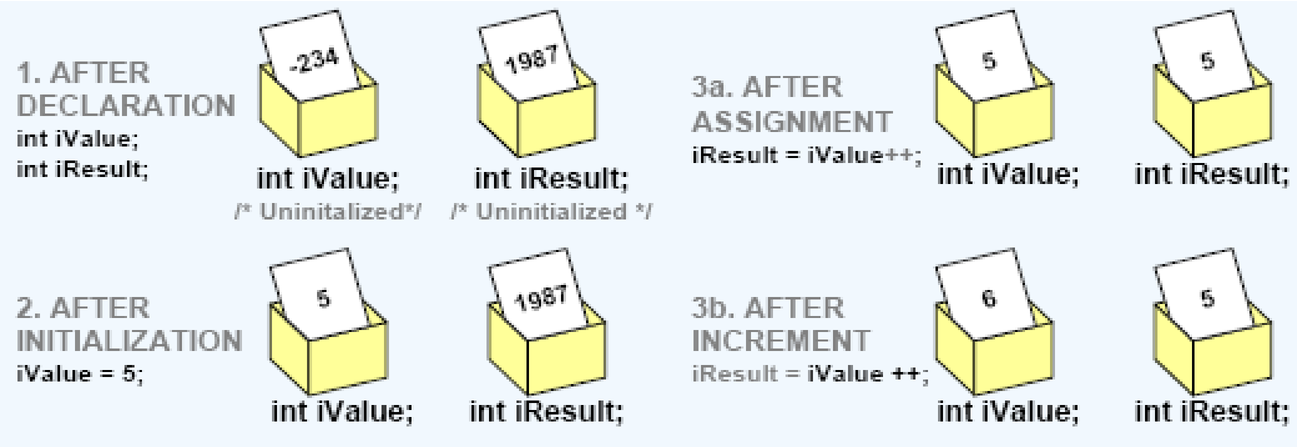
Example:

int iValue;

int iResult;

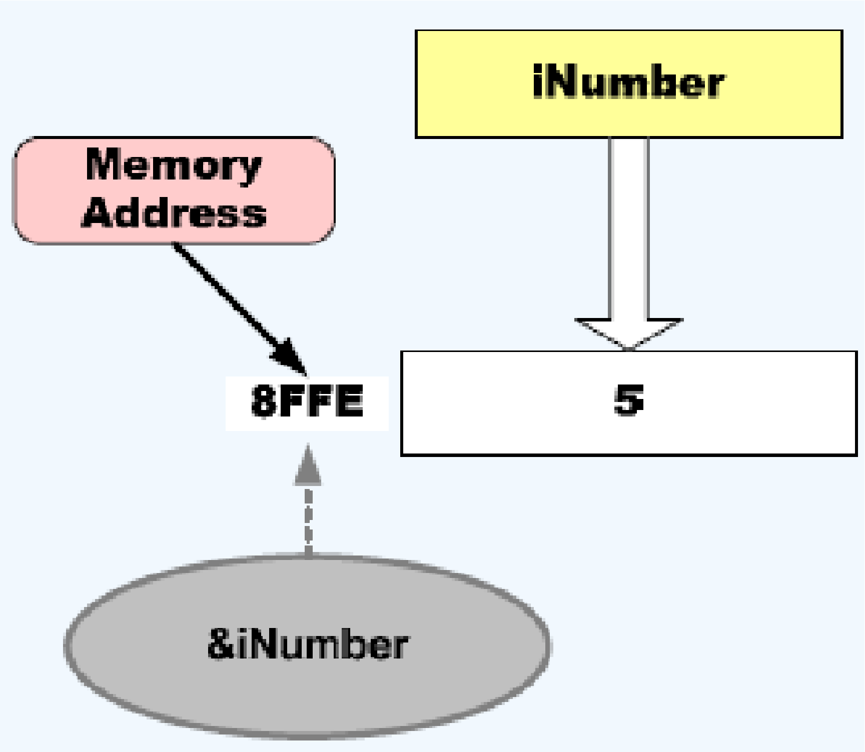
iValue = 5;

iResult = iValue++;



Address of Operator

- Ampersand (&) is the “address of” operator
- It is used to fetch the memory address of a variable



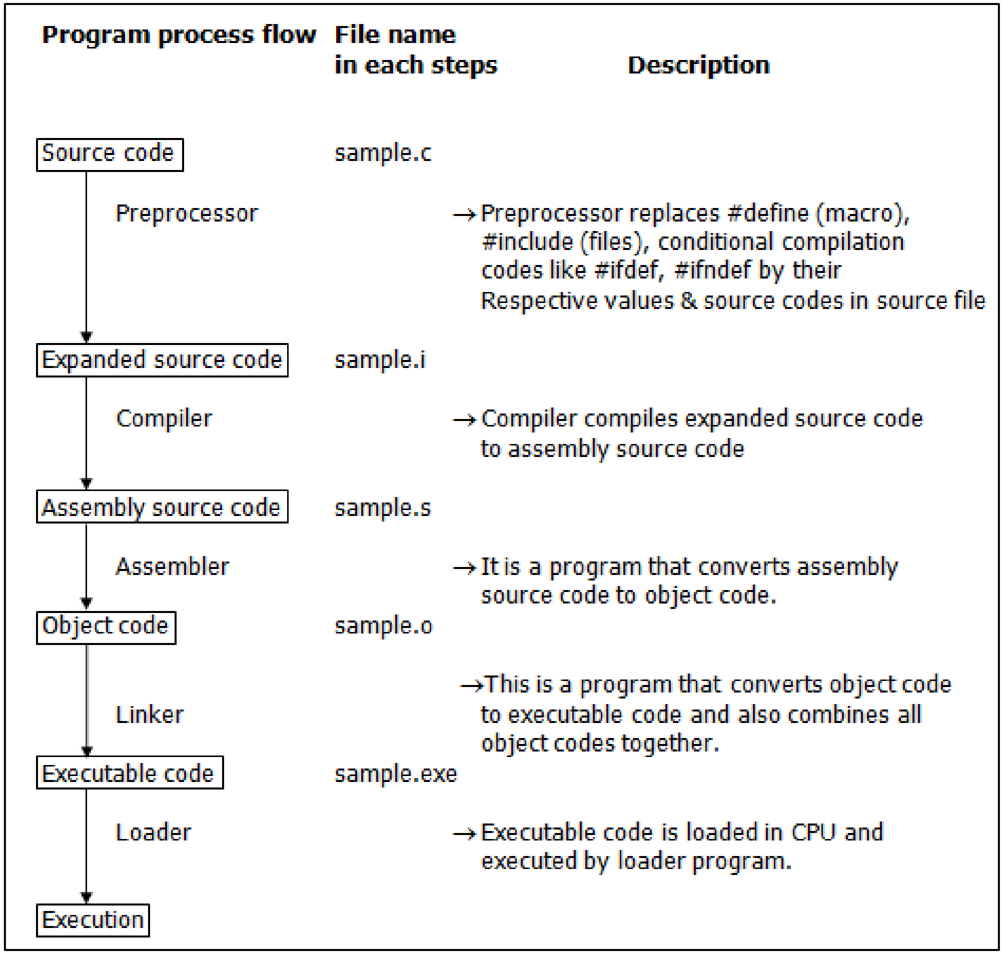
Preprocessor Directives

- Before a C program is compiled in a compiler, source code is processed by a program called preprocessor i.e preprocessor executes before a program is compiled . This process is called preprocessing. The preprocessor works on the source code.
- Commands used in preprocessor are called preprocessor directives and they begin with **“#”** symbol.

Preprocessor	**Syntax/Description**
Macro	**Syntax:** #define This macro defines constant value and can be any of the basic data types.
Header file inclusion	**Syntax:** #include <file_name> The source code of the file “file_name” is included in the main program at the specified place.
Conditional compilation	**Syntax:** #ifdef, #endif, #if, #else, #ifndef Set of commands are included or excluded in source program before compilation with respect to the condition.
Other directives	**Syntax:** #undef, #pragma #undef is used to undefine a defined macro variable. #Pragma is used to call a function before and after

Preprocessor Directives

- A program in C language involves into different processes.



Program process flow	File name in each steps	Description
Source code	sample.c	
Preprocessor		→ Preprocessor replaces #define (macro), #include (files), conditional compilation codes like #ifdef, #ifndef by their Respective values & source codes in source file
Expanded source code	sample.i	
Compiler		→ Compiler compiles expanded source code to assembly source code
Assembly source code	sample.s	
Assembler		→ It is a program that converts assembly source code to object code.
Object code	sample.o	
Linker		→This is a program that converts object code to executable code and also combines all object codes together.
Executable code	sample.exe	
Loader		→ Executable code is loaded in CPU and executed by loader program.
Execution		

Formatted Input: scanf( )

Read input from screen (standard input)

Syntax

```
scanf(“format Specifier-list”,&variable-1, &variable-2,…………………..);
```

Example

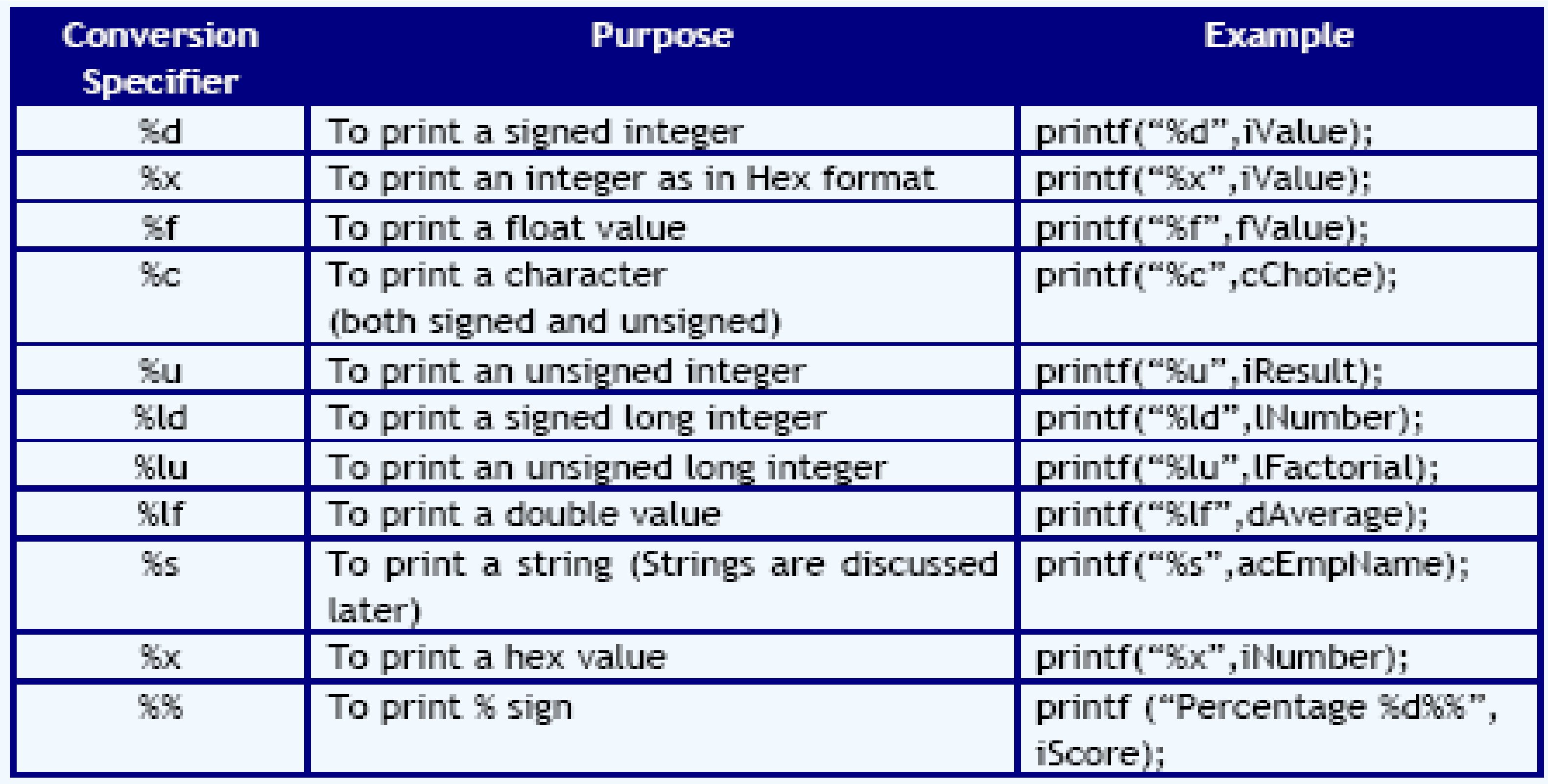
```
void main()
{
 int a,b;
printf(“Enter a number”);
scanf(“%d%d”,&a,&b);
printf(“ No is :%d”,a);
}
```

Formatted Output Using printf

Writes onto screen (standard output)

Syntax

printf(“Conversion Specifier-list”,variable-1, variable-2,…………………);



Conversion Specifier	Purpose	Example
%d	To print a signed integer	printf(“%d”,iValue);
%x	To print an integer as in Hex format	printf(“%x”,iValue);
%f	To print a float value	printf(“%f”,fValue);
%c	To print a character (both signed and unsigned)	printf(“%c”,cChoice);
%u	To print an unsigned integer	printf(“%u”,iResult);
%ld	To print a signed long integer	printf(“%ld”,lNumber);
%lu	To print an unsigned long integer	printf(“%lu”,lFactorial);
%lf	To print a double value	printf(“%lf”,dAverage);
%s	To print a string (Strings are discussed later)	printf(“%s”,acEmpName);
%x	To print a hex value	printf(“%x”,iNumber);
%%	To print % sign	printf (“Percentage %d%%”, iScore);

Formatted Output Using printf (cont'd)

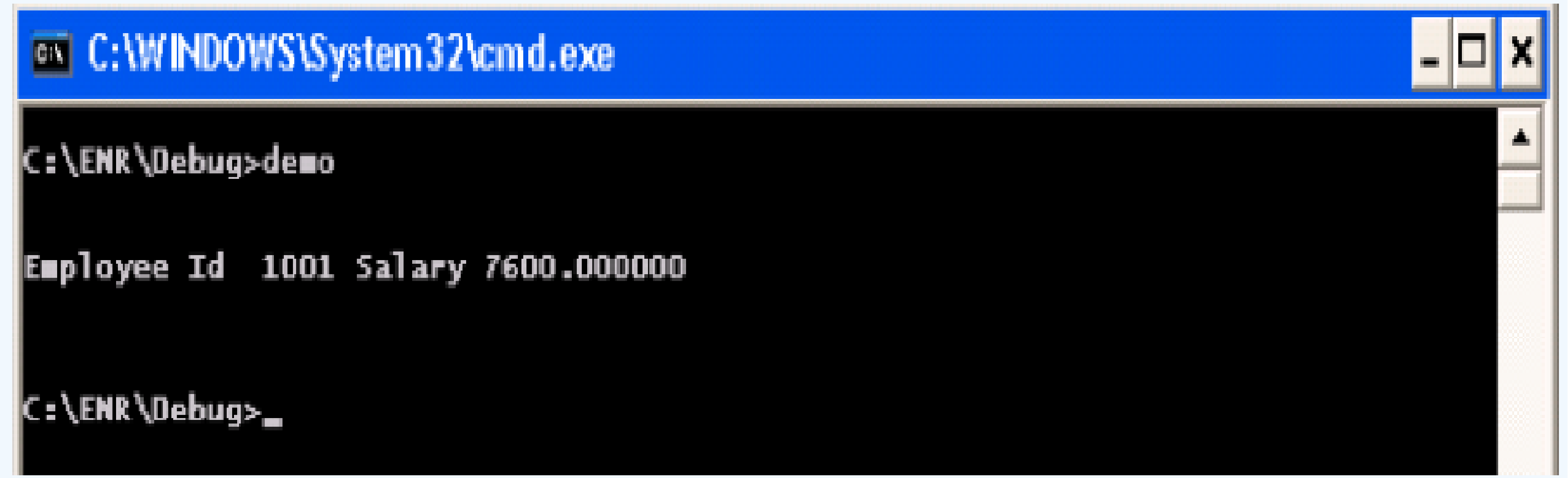
Example:

int EmployeeId = 1001;

double Salary = 7600.00;

printf("Employee Id %d" , EmployeeId);

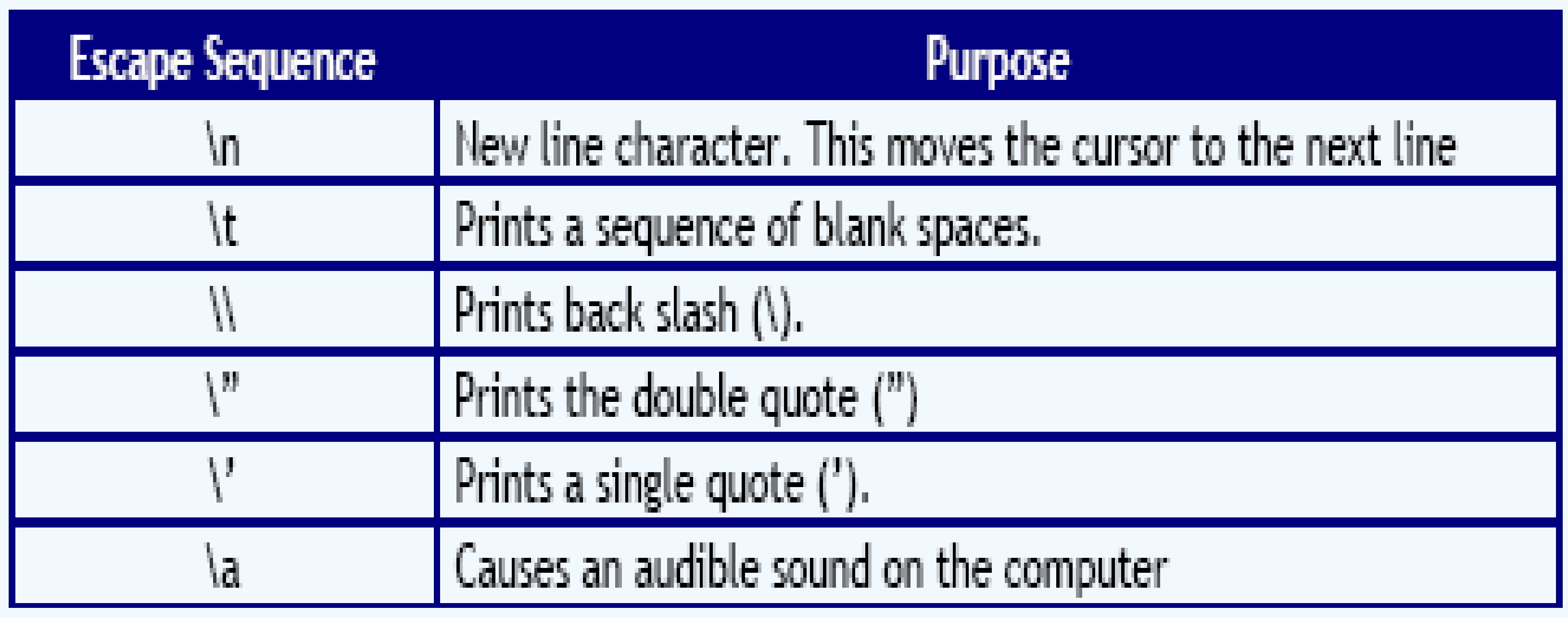
printf("Salary %lf", Salary);



```
C:\ENR\Debug>demo

Employee Id  1001 Salary 7600.000000


C:\ENR\Debug>_
```

Formatted Output Using printf (cont'd)

- An escape sequence is interpreted to have a special meaning in the screen output
- All the escape sequences must be preceded by a back slash (\)
- Escape sequences are non printable characters
- Escape sequences are generally used with 'printf' function

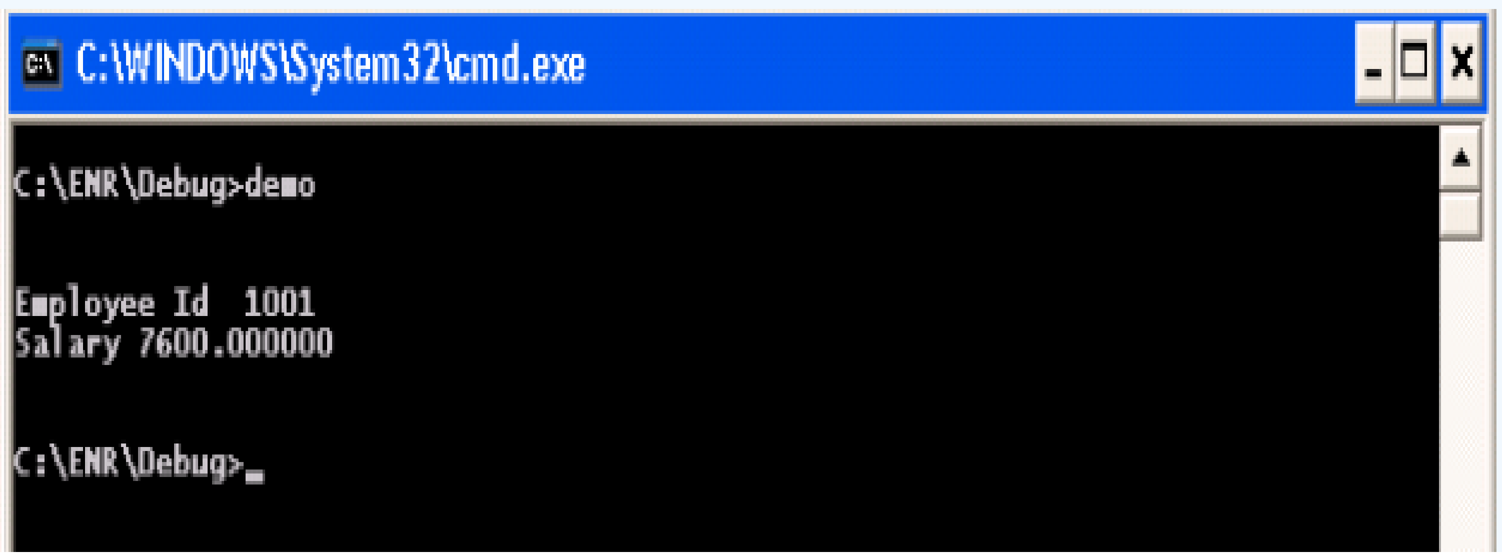


Escape Sequence	Purpose
\n	New line character. This moves the cursor to the next line
\t	Prints a sequence of blank spaces.
\\	Prints back slash (\).
\"	Prints the double quote (")
\'	Prints a single quote (').
\a	Causes an audible sound on the computer

Formatted Output Using printf (cont'd)

Example:

```
int EmployeeId = 1001;
double Salary = 7600.00;
printf("Employee Id %d\n", EmployeeId);
printf("Salary %lf\n", Salary);
```



```
C:\ENR\Debug>demo


Employee Id  1001
Salary 7600.000000


C:\ENR\Debug>_
```

Declaring and Using Character Variables

Syntax:

char variablename1, variablename2,….;

Example:

char Alphabet;

char Status, Number ;

- The value can be assigned by enclosing it in a single quote (‘ ‘)

Example: Alphabet = ‘W’; Status = ‘y’;

- A character variable can be assigned with a numeric value by directly assigning a number

Example:

Number = 77;

- The above statement can also be written as Number = ‘M’;

The ASCII Character Set

- Character data is represented in a computer by using standardized numeric codes which have been developed.
- The most widely accepted code is called the **American Standard Code for Information Interchange** (**ASCII**).
- The ASCII code associates an integer value for each symbol in the character set, such as letters, digits, punctuation marks, special characters, and control characters.
- ASCII value for capital A is 65, B is 66,………., Z is 90.
- ASCII value for small a is 97, b is 98,…………, z is 122.

Printing a character on screen

- For printing characters, **‘%c’** conversion specifier is used in ‘printf’

Example:

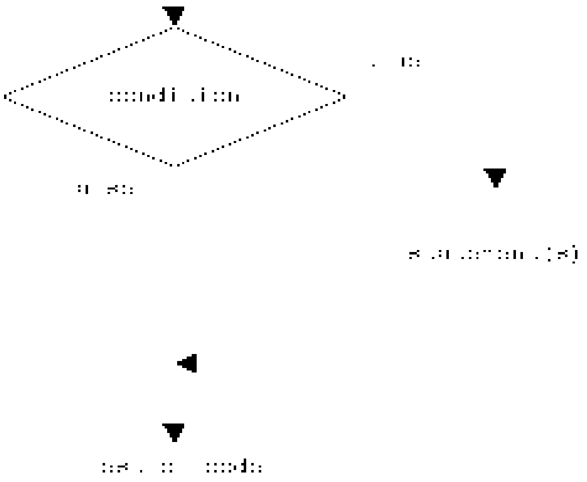
```
char Alphabet = ‘N’;

printf (“%c”, Alphabet);
```

The above code is same as:

```
char Alphabet = 78;

printf (“%c”, Alphabet);
```

Control Structures

Selectional Control Structures



- There are two selectional control structures
  - If statement
  - Switch statement

Simple if Statement

- In a simple ‘if’ statement, a condition is tested
- If the condition is true, a set of statements are executed
- If the condition is false, the statements are not executed and the program
- control goes to the next statement that immediately follows if block

Example

```
if (Duration >= 3)
 {  RateOfInterest = 6.0;  }
```

If-else Statement

else Statement

- In simple ‘if’ statement, when the condition is true, a set of statements are executed. But when it is false, there is no alternate set of statements
- The statement ‘else’ provides the same

Syntax:

```
if (testExpression)
{
// codes inside the body of if
}
else
{
// codes inside the body of else
}
```

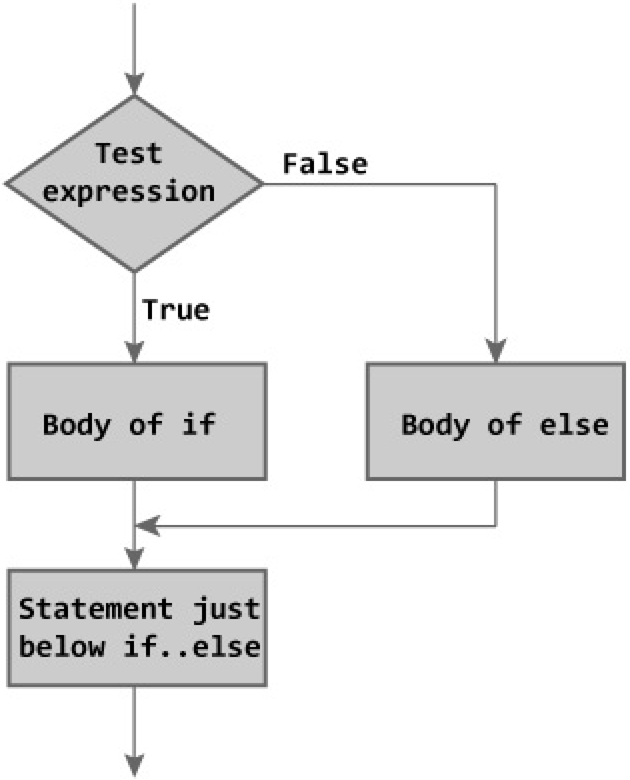


Figure: Flowchart of if...else Statement

If-else Statement (Cont’d)

else Statement

Example:

```
if (Duration >= 3) {
RateOfInterest = 6.0;
}
else  {
RateOfInterest = 5.5;
}
```

else if Statement

- The ‘else if’ statement is to check for a sequence of conditions
- When one condition is false, it checks for the next condition and so on
- When all the conditions are false the ‘else’ block is executed
- The statements in that conditional block are executed and the other ‘if’ statements are skipped

Nested if Statement

Syntax:

```
    if (condition-1)
      {
      Statement 1;
        if (condition-2)
            {
            Statement 2;
            }
            else
            {
                Statement n;
            }
      }
      else {
      Statement x;
      }
Next Statement;
```

Nested if Statement

- An ‘if’ statement embedded within another ‘if’ statement is called as nested ‘if’
- Example:

```
if (iDuration > 6 )
{
      if (dPrincipalAmount > 25000)
       {
      printf(“Your percentage of incentive is 4%”);
      }
      else
      {
      printf(“Your percentage of incentive is 2%”);
      }}
else {
printf(“No incentive”);
}
```

Example

```
/* Program to check whether an integer entered by the user is odd or even */
#include <stdio.h>
int main()
{
    int number;
    printf("Enter an integer: ");
    scanf("%d",&number);
    /* True if remainder is 0*/
     if( number%2 == 0 )
         printf("%d is an even integer.",number);
    else
         printf("%d is an odd integer.",number);
    return 0;
}
```

switch case Statement

- The ‘switch’ statement is a selection control structure that selects a choice from the set of available choices.
- It is very similar to ‘if’ statement.
- But ‘switch’ statement cannot replace ‘if’ statement in all situations.

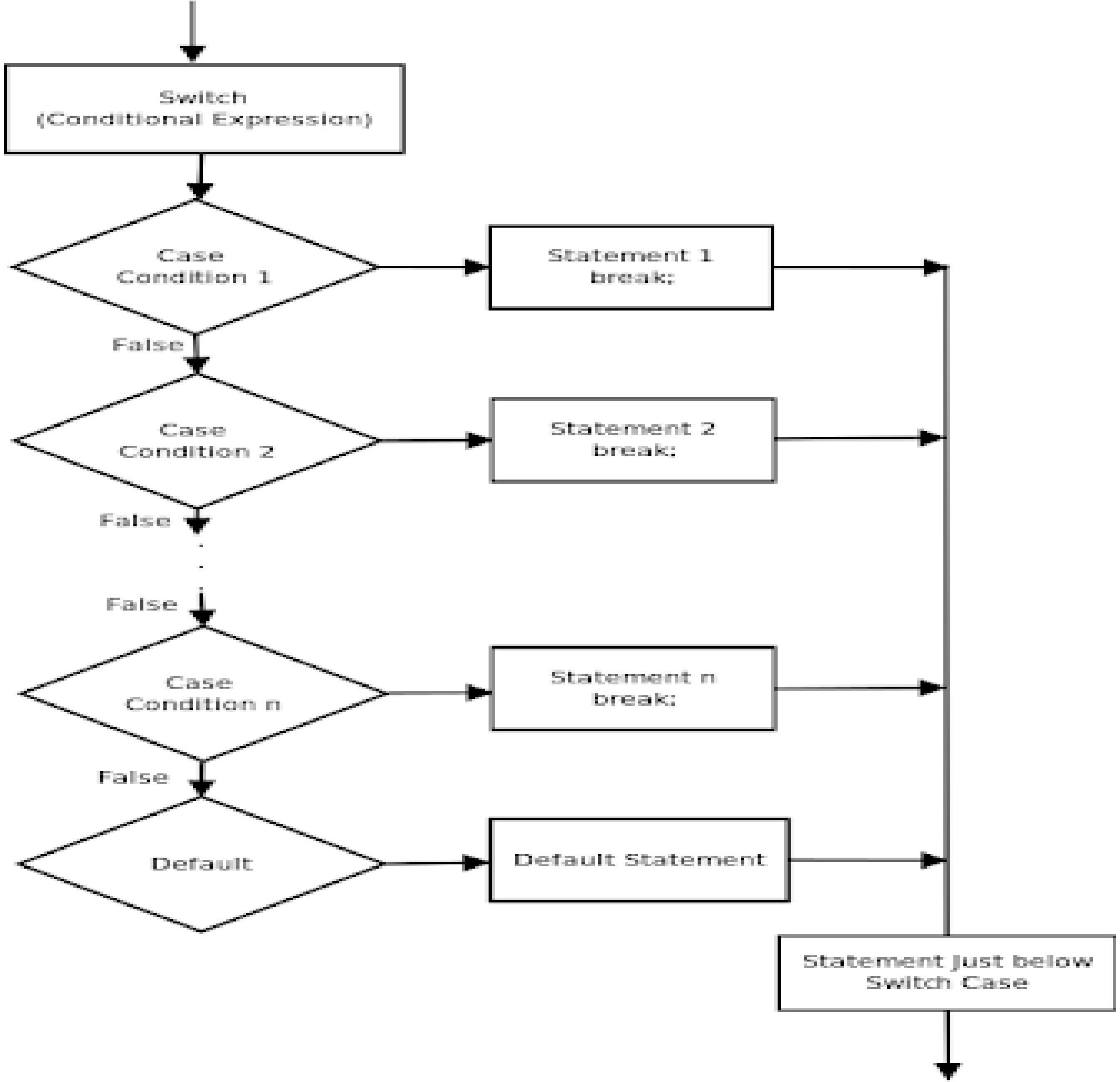
Syntax:

```
int n;
switch (n)
{
case 1: // code to be executed if n = 1;
        break;
case 2: // code to be executed if n = 2;
        break;
default: // code to be executed if n doesn't match any cases
}
```

switch case Example

```
/*Following is a simple program to demonstrate syntax of switch.
#include <stdio.h>
void main()
{
  int x =2;
  switch (x)
  {
     case 1: printf("Choice is 1");
           break;
     case 2: printf("Choice is 2");
            break;
     case 3: printf("Choice is 3");
           break;
     default: printf("Choice other than 1, 2 and 3");


  }
}
```

Output: Choice is 2


Switch
(Conditional Expression)
Case
Condition 1
Statement 1
break;
False
Case
Condition 2
Statement 2
break;
False
False
Case
Condition n
Statement n
break;
False
Default
Default Statement
Statement just below
Switch Case

Iterational Control Structures

- Iterational (repetitive) control structures are used to repeat certain statements for a specified number of times
- The statements are executed as long as the condition is true
- These kind of control structures are also called as **loop control structures**
- Three kinds of loop control structures are:
  - while
  - do while
  - for

while Loop Control Structure

- A ‘while’ loop is used to repeat certain statements as long as the condition is true
- When the condition becomes false, the ‘while’ loop is quitted
- This loop control structure is called as an **entry-controlled** loop because, only when the condition is true, are the statements executed

Syntax:

```
while (condition)
{
     Set of statements;
}
Next Statement;
```

Example:

```
unsigned int Count = 1;
while (Count <= 3)
{
printf(“%d\n”,Count);
```

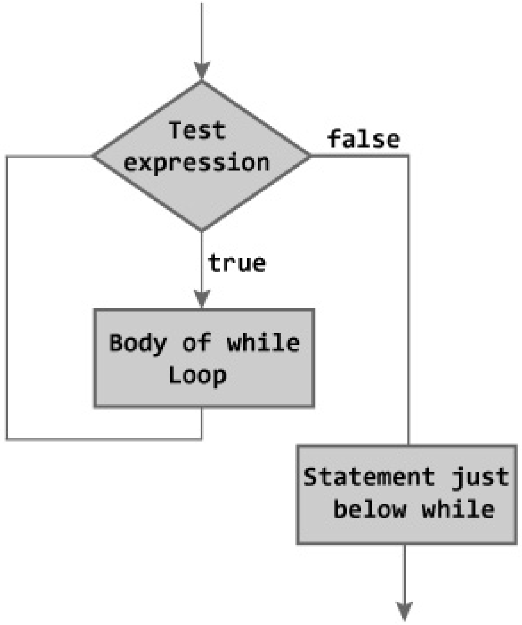


Figure: Flowchart of while Loop

What is the output of the following code?

Example 1

```
unsigned int Count=3;
while (Count<=5)
{
printf(“%u\n”,Count);
Count++;
}
```

Example 2

```
int Count = 0;
while(Count<=10)
{
     printf(“\n
   MITWPU”);
     Count++;
}
```

do while Loop Control Structure

- The ‘do while’ loop is very similar to ‘while’ loop. In ‘do while’ loop, the condition is tested at the end of the loop.
- Because of this, even when the condition is false, the body of the loop is executed at least once.
- This is an **exit-controlled** loop.

Syntax:

```
do
    {
Set of statement(s);
    } while (condition);
Next Statement;
```

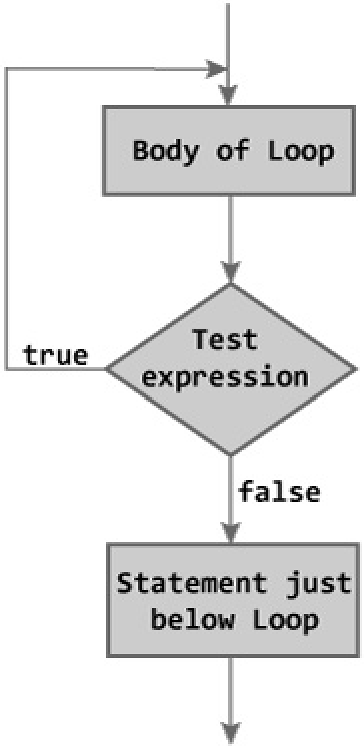
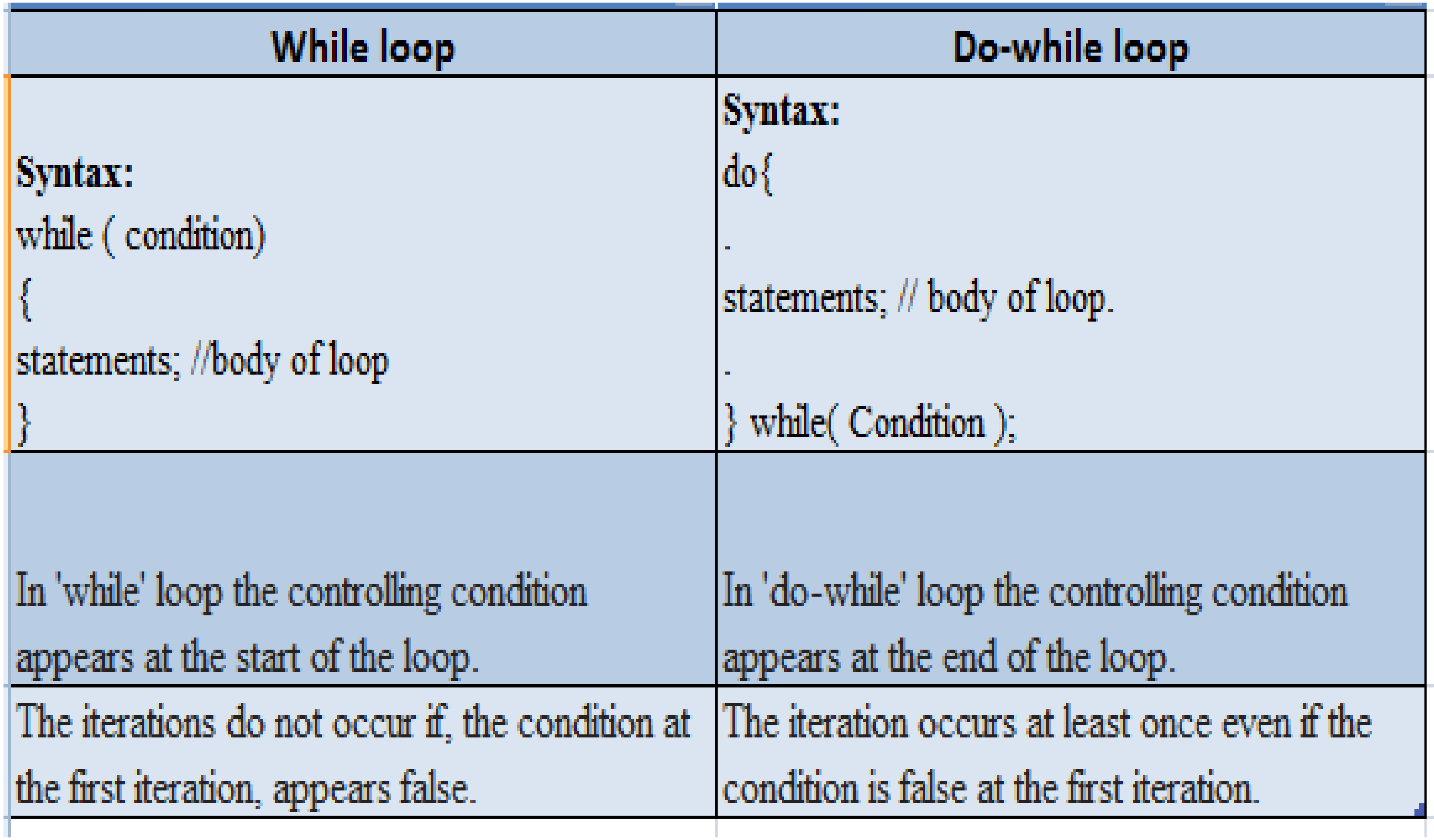


Figure: Flowchart of do...while Loop

do while Loop Control Structure Example

```
int Number, Sum = 0;
do {
   printf(“Enter a number. Type 0(zero) to end
  the input ”);
   scanf(“%d”,&Number);
   Sum = Sum + Number;
   } while (Number != 0);
```

Difference between while and do while loops



While loop	Do-while loop

Syntax:
while (condition)
{
statements; //body of loop
}
	Syntax:
	do{
	.
	statements; // body of loop.
	.
	} while(Condition);

In 'while' loop the controlling condition appears at the start of the loop.	In 'do-while' loop the controlling condition appears at the end of the loop.
The iterations do not occur if, the condition at the first iteration, appears false.	The iteration occurs at least once even if the condition is false at the first iteration.

for Loop Control Structure

- The ‘for’ loops are similar to the other loop control structures
- The ‘for’ loops are generally used when certain statements have to be executed a specific number of times
- Advantage of for loops:
  - All the three parts of a loop (**initialization**, **condition** , **increment**) can be given in a single statement
  - Because of this, there is no chance of user missing out initialization or increment steps which is the common programming error in ‘while’ and ‘do while’ loops

Syntax:

```
for (Initialization; Termination-Condition; Increment-Step)
{
Set of statement(s);
}
Next Statement;
```

for Loop Control Structure (cont’d)

Example:

```
int Count;
for (int Count = 1; Count <= 5; Count++)
 {
printf(“%d\n”,Count);
}
```

Output:

```
1
2
3
4
5
```

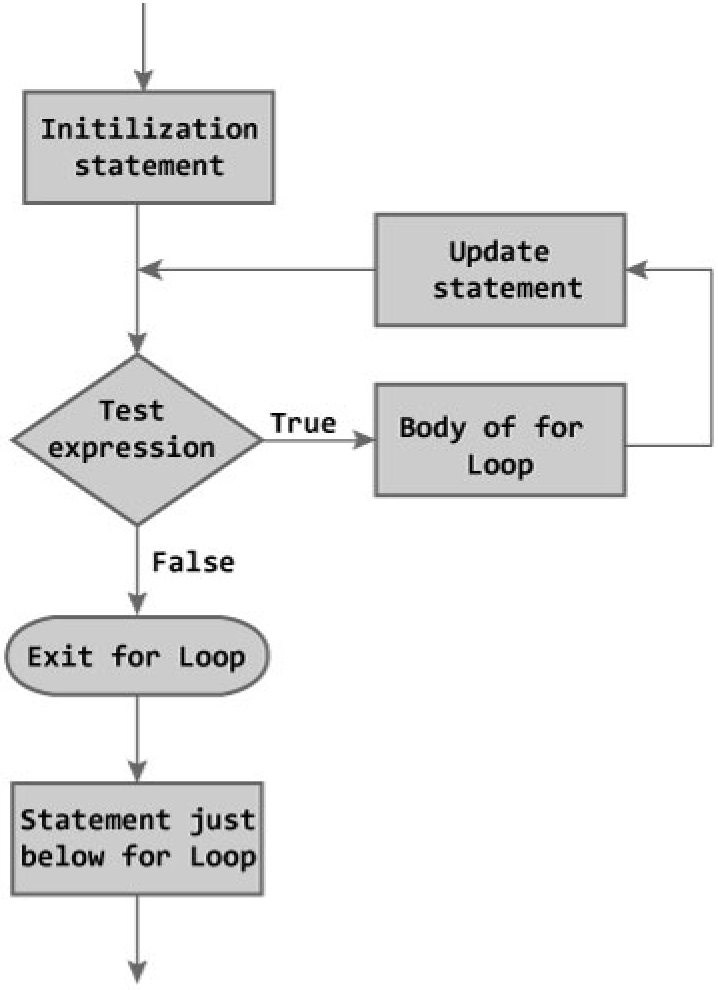


Figure: Flowchart of for Loop

What is the output of the following code?

```
int Num;
int Counter;
int Product;
for(Counter=1; Counter<= 3; Counter++)
{
    Product = Product * Counter;
}
printf("%d", Product);
```

The output is a junk value -- WHY???

What is the output of the following code?

```
for(int Count=0;Count<10;Count++);
{
    printf("%d\n",Count);
}
```

Have U observed this?

The output 10

for and while loops

Rewrite it using **while** statement

Given

```
int sum,Ctr=0;

for(int Sum=0,Ctr=0;Ctr<10;Ctr=Ctr+1)

{

    scanf(“%d”,&Num);

    Sum=Sum+Num;

}

printf(“%d”,Sum);
```

```
Sum=0,Ctr=0;

while(Ctr<10)

{

    scanf(“%d”,&Num);

    Sum=Sum+Num;

    Ctr=Ctr+1;

}

printf(″%d″,Sum);
```

Nested Loops

- A loop with in another loop is called as nested loop.

Example:

```
   while (flag==1)
{
   for (Count=1;Count<=10;Count++)
   {
      statements;
   }
}
```

Quitting the Loops – **break** Statement

The break statement is used to:

- Force the termination of a loop.
- When a break statement is encountered in a loop, the loop terminates immediately and the execution resumes the next statement following the loop.

Note:

- Break statement can be used in an if statement only when the if statement is written in a loop
- Just an if statement with break leads to compilation error in C

What is the output of the following code?

```
int Counter1=0;
int Counter2;
while(Counter1 < 3) {
for (Counter2 = 0; Counter2 < 5; Counter2++) {
printf("%d\t",Counter2);
if (Counter2 == 2){
break;
}
}
printf("\n");
Counter1 += 1;
}
```

0 1 2 is printed 3 times

Continuing the Loops - **continue** Statement

- 'continue' statement forces the next iteration of the loop to take place and skips the code between continue statement and the end of the loop
- In case of **for** loop, continue makes the execution of the increment portion of the statement and then evaluates the conditional part.
- In case of **while** and **do-while** loops, continue makes the conditional statement to be executed.

Example:

```
int count;
for( Count = 0 ; Count < 10; Count++) {
if (Count == 4) {
continue;
}
printf("%d\n", Count);
}
```

The above code displays numbers from 1 to 9 except 4.

Terminating the program using **exit()** function

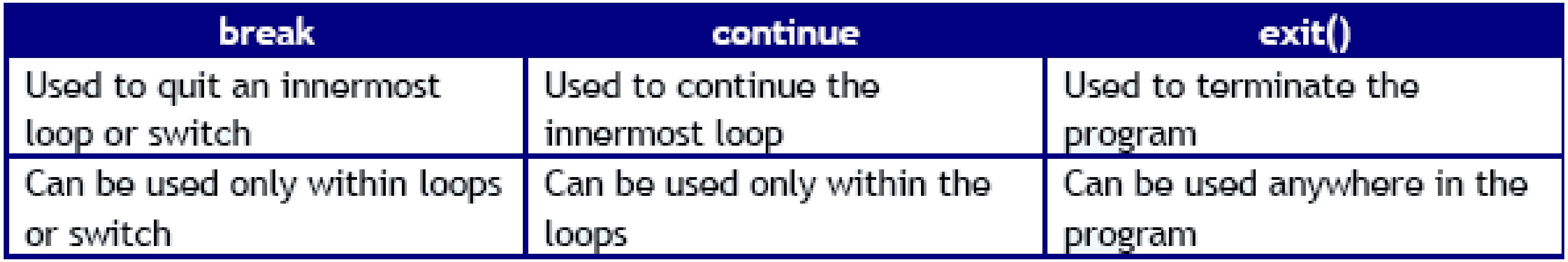
- The function ‘exit’ is used to quit the program.
- Terminates the program and returns the status code to the operating system.
- This function is defined in ‘**stdlib.h**’ header file.

Syntax:

```
exit(int status);

exit(0);
```

- The status code zero indicates that the program completed successfully.
- If there is a failure any other code has to be returned.

Comparison of break, continue and exit



break	continue	exit()
Used to quit an innermost loop or switch	Used to continue the innermost loop	Used to terminate the program
Can be used only within loops or switch	Can be used only within the loops	Can be used anywhere in the program

Practice Programs

if - else statement

1. Create a program that prints a Student is passed(if marks>=40) or failed.

2. Check the entered character whether it is vowel or consonant.

switch statement

1. Check the entered character whether it is vowel or consonant.

2. Write a program that calculates the area of circle, area of triangle, area of square using switch statement.

while loop control structure

1. Write a program that calculates the sum of the digits of a entered number and display the sum.

2. Write a program to print the output like:

```
*
     * *
    * * *
   * * * *
  * * * * *
```

Source code of printing “* “

```
void main()
{
        int i,j;
        int space=4;
//run loop (parent loop) till number of rows
        for(i=0;i< 5;i++)
        {
//loop for initially space, before star printing
for(j=0;j< space;j++)
                {
            printf(" ");
                }
for(j=0;j<=i;j++)
{
   printf("* ");
}
    printf("\n");
    space--;   // decrement one space after one
row
}
}
```

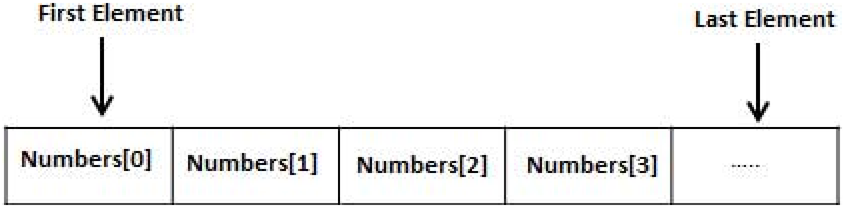
Arrays in C

- An array is a set of elements of same datatype.
- Each variable in an array is called an array element.
- All the elements are of same type, but may contain different values.
- The entire array is contiguously stored in memory.
- The position of each array element is known as array index or subscript.
- An integer array looks like this:

Declaring Arrays

- An array is a set of elements of same datatype.
- Each variable in an array is called an array element.
- All the elements are of same type, but may contain different values.
- In C, an array can be of any basic data type.
- Example: int a[6]; //single or one dimensional array

  float Salary[6];

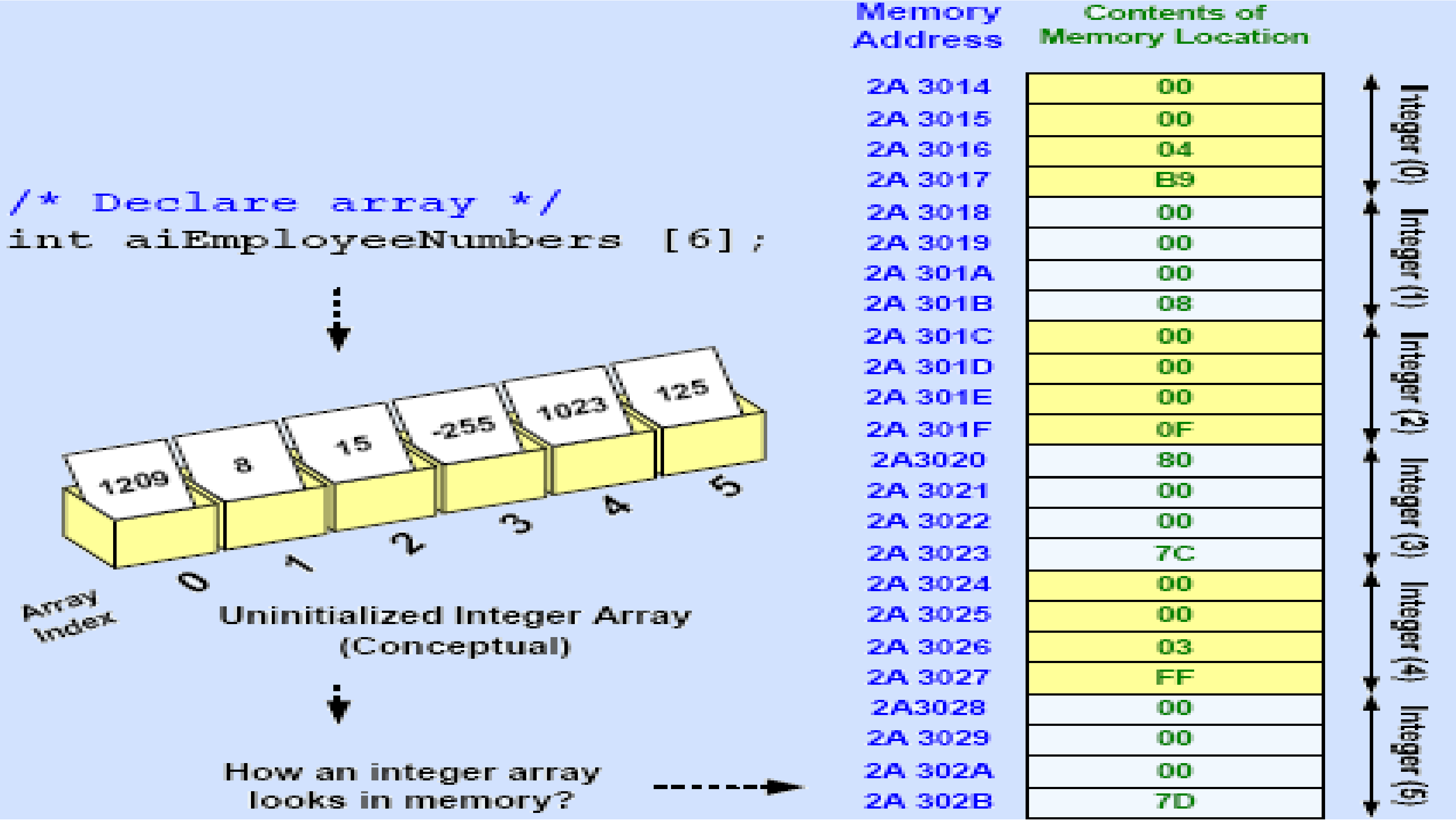


Numbers[0]	Numbers[1]	Numbers[2]	Numbers[3]	.....

Contd…

- The array index starts with zero.
- The valid array indexes for the above declared array is 0 to 5.
- When an array is declared without initializing it, the elements have unknown (garbage) values.

Memory Representation of an Integer Array



```
/* Declare array */
int aiEmployeeNumbers [6];
```

Uninitialized Integer Array (Conceptual)

How an integer array looks in memory?

Memory Address	Contents of Memory Location	
2A 3014	00	Integer (0)
2A 3015	00	
2A 3016	04	
2A 3017	B9	
2A 3018	00	Integer (1)
2A 3019	00	
2A 301A	00	
2A 301B	08	
2A 301C	00	Integer (2)
2A 301D	00	
2A 301E	00	
2A 301F	0F	
2A3020	80	Integer (3)
2A 3021	00	
2A 3022	00	
2A 3023	7C	
2A 3024	00	Integer (4)
2A 3025	00	
2A 3026	03	
2A 3027	FF	
2A3028	00	Integer (5)
2A 3029	00	
2A 302A	00	
2A 302B	7D	

Declaring and Initializing Arrays

- Arrays can be initialized as they are declared.
- Example:

  int aiEmployeeNumbers[ ] = {15090, 15091, 15092, 15093,15094, 15095};

- The size in the above case is optional and it is automatically computed.
- In the above example size of the array is 6 and it occupies 6 * 4 = 24 bytes.

Using Array Elements

- An array can be accessed by giving the respective index.
- Syntax:

  ArrayName[index]

- Example:

  float afSalaries[6]; /* Declare an array of six floats */

  afSalaries[0] = 12500.00; /* Assign a value to element at index 0 */

  afSalaries[1] = 15000.00; /* Assign a value to element at index 1 */

  ...

  afSalaries[5] = 25000.00; /* Assign a value to element at index 5 */

Contd…

- /* Print the salary at array index 0 */
- printf (“Salary at Index 0 is %f \n”, afSalaries[0]);
- Arrays can also be referenced as index[ArrayName].
- The statement 5[afSalaries] is a valid statement.

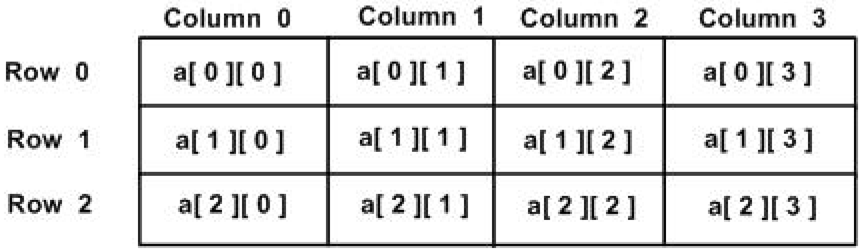
Arrays and Loops

```
#include<stdio.h>
int main()
{
#define ARRAYSIZE 10

int aiArray[ARRAYSIZE],iCount;   //int aiArray[10];

for (iCount = 0; iCount < ARRAYSIZE; iCount++)
 {          printf(“Enter %d element”,iCount);
      scanf(“%d”,&aiArray[iCount]);
}
      for (iCount = 0; iCount < ARRAYSIZE; iCount++)
{
      printf(“Element in %d position %d”,iCount,aiArray[iCount]);
}
return 0;
}
```

Multi dimensional Arrays:

- The array has two subscripts. One subscript denotes the row & the other the column. The declaration of two dimension arrays is as follows:

  - int arr[2][3];



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

- Syntax :

```
data_type array_name[row_size][column_size];

int m[10][20];
```

- Here m is declared as a matrix having 10 rows(numbered from 0 to 9) and 20 columns(numbered 0 through 19). The first element of the matrix is **m[0][0]** and the last row last column is **m[9][19]** .

Initialization Of Multidimensional Arrays

Like the one dimension arrays, 2 dimension arrays may be initialized by following their declaration with a list of initial values enclosed in braces.

- **Example:**

int table[2][3]={0,0,0,1,1,1};

OR

int table[2][3]={{0,0,0},{1,1,1}}

int first[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};

int second[3][4] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11}; //a clearer definition than the first

int third[][5] = {0,1,2,3,4}; /* third[] only has one index of 1 */

- int fourth[][6] = {0,1,2,3,4,5,6,7,8,9,10,11}; /* fourth[] has 2 indices - 0 or 1 */

Multidimensional Arrays

```
#include<stdio.h>
int main(){
int arr[2][2]; // Two-dimensional array declaration
int i, j;
for(i=0; i<2; i++) {
 for(j=0;j<2;j++) {
   printf("Enter value for arr[%d][%d]:", i, j);
   scanf("%d", &arr[i][j]);
}
}
printf("Two Dimensional array elements:\n"); //Displaying array elements
for(i=0; i<2; i++) {
  for(j=0;j<2;j++) {
printf("%d ", arr[i][j]);
}
printf("\n");
}
return 0;
```

Strings

- "A string is a series of characters in a group that occupy contiguous memory or String is an array of character."
- Example: "My Training"

  "Fundamentals of Data Structure"
- A string should always be enclosed with in double quotes (")
- In memory, a string ends with a null character '\0' and it occupies 1 byte of memory.
- Space should be allocated to store '\0' as part of the string
- C compiler automatically adds a NULL character '\0' to the character array created.

Declaration Of Strings

- C language does not directly support string as a data type. Hence, to display a String in C, need to make use of a character array.
- **char** variablename [Number_of_characters];
- Example: char acEmployeeName[20];

Here 20 implies that the maximum number of characters can be 19 and one position is reserved for '\0'

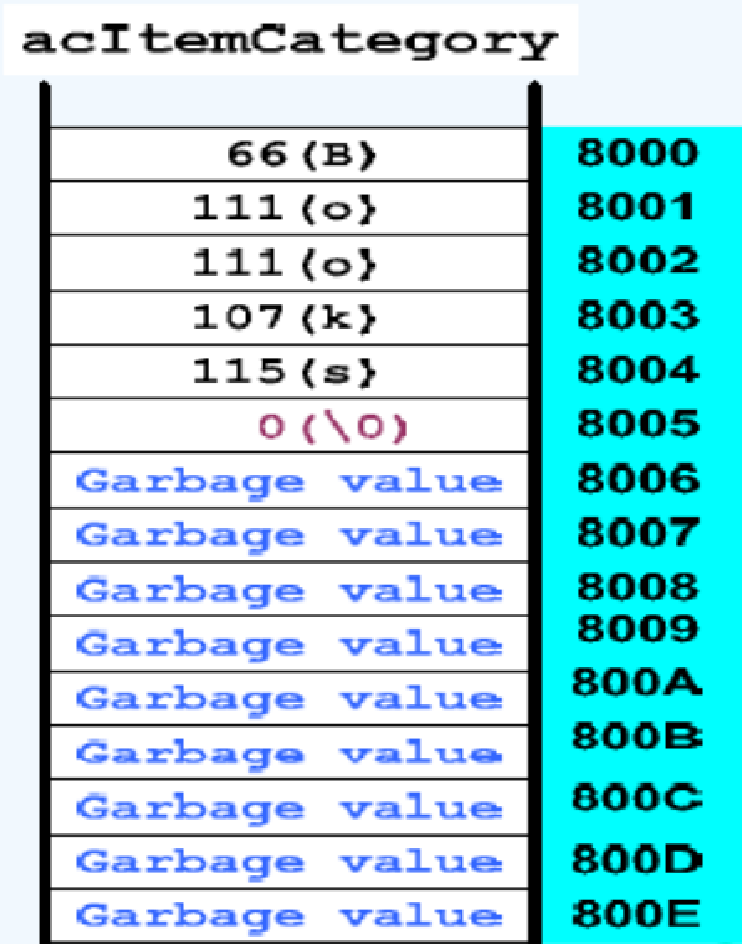
Since a character occupies one byte, the above array occupies 20 bytes (19 bytes for the employee name and one byte for '\0')

/* Declaring a string as a character array */

char acInfy[ ] = "MITWPU Pune";

Storage Of Strings In Memory

char acItemCategory[15]= “Books”;



acItemCategory

66(B)	8000
111(o)	8001
111(o)	8002
107(k)	8003
115(s)	8004
0(\0)	8005
Garbage value	8006
Garbage value	8007
Garbage value	8008
Garbage value	8009
Garbage value	800A
Garbage value	800B
Garbage value	800C
Garbage value	800D
Garbage value	800E

Static Initialization of Strings

- char acItemCategory[15]="Greeting Cards";
- In the above declaration, the size of the array is specified according to the number of characters in the string. One extra space for '\0'
- char acItemCategory[15] ="Ornaments" ;
- Here the size is more than the number of characters which is valid.
- char acItemCategory[ ]="Groceries";
- Here the size of the array is computed automatically which is 10. Total number of characters in the string is 9 and 1 for '\0';

Contd…

- char acItemCategory[3]=“Books”;
- Here the size specified is 3. But the number of characters in the string is 5. This is invalid.
- char acItemCategory[ ]={'s','t','a','t','i','o','n','a','r','y','\0'};
- Here the character constants are supplied to initialize the string.

Reading and Printing Strings - Example

```
/*Program to accept Item Category and display*/
#include<stdio.h>
int main()
{     char str[15]; //character array declaration
      printf("Enter the category code: ");
      scanf("%s",str);
      printf("The given Item Category is %s", str);
return 0;
}
```

String Handling Functions

- The following are the string built-in functions that are supported by C

  strlen() *strcpy()* *strcat()* *strcmp()*

- These functions are defined in **string.h** header file.
- All these functions take either a character pointer or a character array as an argument.

strlen() Function

- strlen() function is used to count the number of characters in the string. Counts all the characters excluding the null character ‘\0’
- Syntax:

```
size_t strlen (const char* s);
```

- The strlen() function is defined in <string.h> header file.

Contd…

- Example:

    strlen(“Programming Fundamentals”); /
returns 24/

    strlen(acItemCategory);

- returns the number of characters in the character array ‘acItemCategory’

Example: C strlen() function

```
#include <stdio.h>
#include <string.h>
int main() {
char a[20]="Program";
char b[20]={'P','r','o','g','r','a','m','\0'};
char c[20];
    printf("Enter string: ");
    gets(c); //scanf("%s",c);
printf("Length of string a = %d \n",strlen(a));

 //calculates the length of string before null charcter.

 printf("Length of string b = %d \n",strlen(b));
 printf("Length of string c = %d \n",strlen(c));
    return 0; }
```

strcpy() Function

- strcpy() function is used to copy one string to another
- Syntax:

```
        strcpy (Dest_String , Source_String);
```

- Here Dest_string should always be variable
- Source_String can be a variable or a string constant

Contd…

- The previous contents of Dest_String, if any, will be over written
- Example:

```
        char acCourseName[40];
strcpy(acCourseName , “C Programming”);
```

- Now acCourseName will get the value “C Programming”

Example: C strcpy()

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[10]= "awesome";
    char str2[10];
    char str3[10];
    strcpy(str2, str1); //function call
    strcpy(str3, "well");
    puts(str2);
    puts(str3);
    return 0;
}
```

strcat() Function

- strcat() function is used to concatenate (Combine) two strings
- Syntax:

  strcat(Dest_String_Variable , Source_String) ;

- In this, the Destination should be a variable and Source_String can either be a string constant or a variable.
- The contents of Dest_String is concatenated with Source_String contents and the resultant string is stored into Dest_String variable.

Contd…

- Example:
- char acTraineeFpCourse [50] = “The course is “;
- strcat(acTraineeFpCourse,”Oracle 8i”);
- The resultant string in acTraineeFPCourse will be
- “The course is Oracle 8i”

Example: C strcat() function

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[] = "This is", str2[] = "MIT-WPU";
    //concatenates str1 and str2 and resultant string is stored in str1.
    strcat(str1,str2);
    puts(str1);
    puts(str2);
    return 0;
}
```

strcmp() Function

- strcmp() function is used to compare two strings
- This is case sensitive
- Syntax:

  int strcmp(String1 , String2)

- Here both String1 and String2 can either be a variable or a string constant
- strcmp() function returns an integer value.

Contd…

- If strings are equal it returns zero
- If the first string is alphabetically greater than the second string then it returns a positive value.
- If the first string is alphabetically less than the second string then it returns a negative value.
- Example:

  strcmp(“My Work”, “My Job”) returns a positive value.

Contd…

- If strings are equal it returns zero
- If the first string is alphabetically greater than the second string then it returns a positive value.
- If the first string is alphabetically less than the second string then it returns a negative value.
- Example:
- strcmp(“My Work”, “My Job”) returns a positive value

Example: C strcmp() function

```
#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;

}
```