C Programming Fundamentals

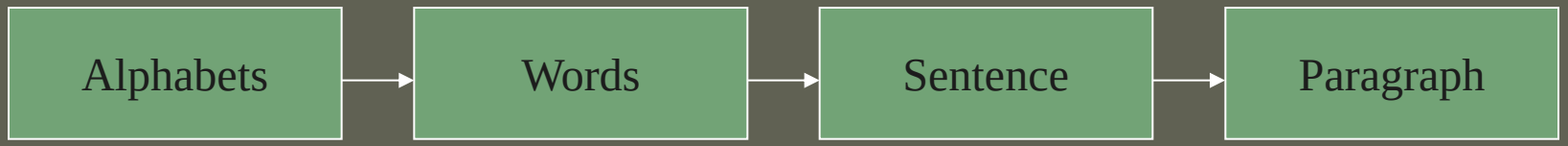
Module 1

CHARACTERISTICS OF C PROGRAM

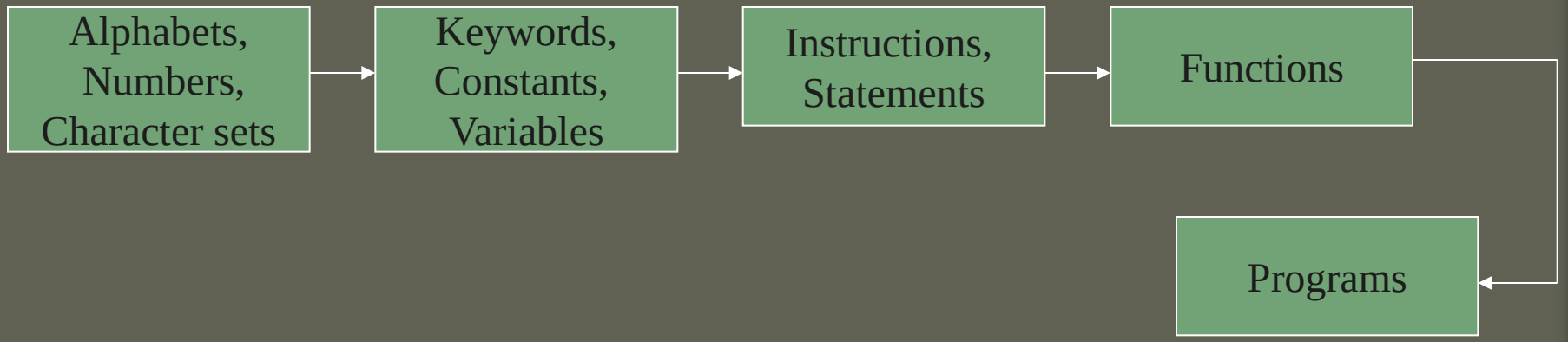
1. Structured Programming Language.
2. C uses Top Down Approach.
2. C has rich set of operators.
3. C has large number of Inbuilt (Library) functions.
4. Using C Program both system s/w and Application s/w can be developed.
5. C is a middle level language.
6. It is highly portable.
7. It is highly efficient.

GETTING STARTED WITH C

Learning English



Learning C



C CHARACTER SET

1. Alphabets - A-Z (Upper case) and a-z (Lower case).

2. Digits - 0 to 9.

3. Special Characters

   Eg:

   + , - , * , %, / , = , \(back slash),& (ampersand),

(Almost all the characters in the key board is supported by C)

C Tokens

1.Key Word (Reserved Words) - int, while, for, main

2.Identifiers - Names given to various program elements.
→ Variables, constants, Functions, arrays etc.,

3.Constants – Any Entity whose value remains the same throughout the program execution is a constant.

→Numerical constants

1.Interger constants - 2
2.Real constants - 50.6

→Character Constants

1.Single Character Constant - 'A'
2.String Constant - "Hai"

ESCAPE SEQUENCE

- Escape sequence or Back slash character constant or non-printable characters.

- The character such as single quotes, Double quotes and control characters are displayed using escape sequence.
- The character preceded with a Back slash is said to be a Escape sequence character.

Eg:

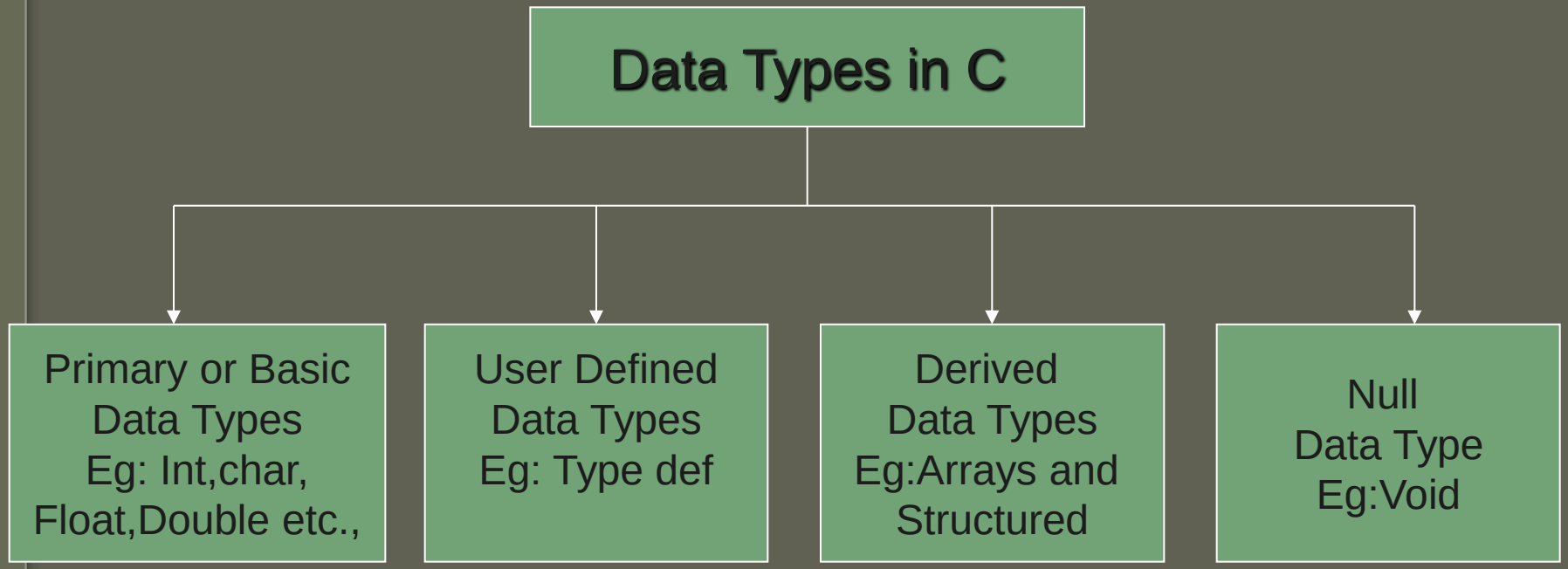
1.'\n' - New line character.
2.'\t' - Horizontal tab
3.'\v' - Vertical tab
4.'\0' - Null
5.'\\' - Back slash
6.'\?' - Question marks
7.'\'' - Single Quotes
8.'\"' - Double quotes

DATA TYPES

Two types of Statements in C

→ Declarative statement - used to allocate memory for variables depend on data types

→ Executable statement - used to do specific task



C DATA TYPES

Data Types	Memory space allocated	Range of values	Control Strings or Format Descriptor
Int	2 Bytes	-32,768 to 32,767	%d
Char	1Byte	-128 to 127	%c
Float	4 Bytes	3.4e-38 to 3.4 e+38	%f
Double	8 Bytes	1.7e-308 to 1.7e+308	%lf

Type Qualifiers:

Short, Long, Signed, unsigned

Qualifiers are used to increase or decrease the range

Of values for data types

VARIABLE DECLARATION

- Declaration tells the variable name to the compiler and specifies the type of data. Based on the data type the compiler will allocate appropriate memory space for those variables.

Syntax:

data-type v1,v2,….,vn;

Where

Data Type is one of the primary type

v1,v2,…,vn are varables

; → All statements in C pgm are to be terminated using semi colon

, → Variable separator (Delimiter)

ASSIGNING VALUES TO VARIABLES

Values can either be supplied as i/p through input statement or constant values can be assigned to the variables during declaration.

Initialization during Declaration:

```
int a=10;
char b='A'
float c=2.7
```

C - STRUCTURE

```
Documentation Section
Header File Section or Link Section
Global Variable Declaration
Main() Function
{
        Local Variable Declaration;
        Executable Statements;
        ..........................;
        ..........................;
}
Sub Program
{
   Statements;
}
```

Sections of C

Documentation Section:

→It contains a Single Comment Lines or a set of comment lines.

→ To specify the details of the program comments are being used

→ The statements with in comment will not be executed.

→ It increases the readability of the program

Documentation Section

Examples:

```
// Simple C Program
#include<stdio.h>
/*void main()
{
        Declarative statement;
        Executable statements;
}*/
```

// - Single Line comment

/* */ - Double Line comment

Sample Program

```
#include<stdio.h>
#include<conio.h>
main()
{
      int a,b,sum;
      printf("Enter the value of a:");
      scanf("%d",&a);
      printf("Enter the value of b:");
      scanf("%d",&b);
      sum=a+b;
      printf("Sum=%d",sum);
      getch();
}
```

Sections of C

Header File Section:

→ It is used to include the corresponding Header files based on the in-built functions used in the program

→ It is other wise said to be preprocessor section.

Eg:

1.#include<stdio.h> - Includes standard i/p inbuilt funtions.
2.#include<conio.h> - Console i/p header file
3.#include<string.h> - String library
4.#include<ctype.h> - Character testing header file

Header File section

Example:

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

To include the in-built functions in the main program the corresponding header files are to be included for proper execution.

Global Variable Section

- The variables which are declared out side the main( )Function is said to be a Global Variable.
- The values of Global variables can be accessed by all the functions in a given program

Local Variable Section

- The variables which are declared inside the function is said to be Local variable.
- The first statement with in any function should be a Local variable declaration statement.

Global Variable Section

Examples:

```
#include<stdio.h>
int a=10; // Global variable
void main( )
{
        int a=20; // Local Variable
        printf("%d",a);
        f1( );
}
f1( )
{
        printf("%d",a);
}
```

Output:

```
20
10
```

The Main( ) Section

- The Program execution starts in a c program only from main( ) function.
- All C Programs should have exactly one main( ) function.
- The { (open brace) denotes the start of the main( ) function. And } (close brace) denotes the end of main( ) function.
- All the statements written with in these braces form the Function body.

The Main Function

Examples:

```
#include<stdio.h>
void main( ) // Program execution starts here
{ // starting of main function

        Statements; // Body of main function

} //  Ending of main function
```

Memory Allocation for variables

Example:

```
main( )
{
        int a,b;
        float c;
        char ch;
}
```

Memory Allocation for variables

int a,b;

Float c;

a

b

1000-1001
(2 Bytes)

2000-2001
(2 Bytes)

3000-3003
(4 Bytes)

char ch;

4000
(1 Bytes)

OPERATORS

Operator & Operand

Types of Operators:

- → 1.Arithmetic Operators
- → 2.Relational Operators
- → 3.Assignment Operators
- → 4.Increment and Decrement Operators
- → 5.Conditional or Ternary Operators
- → 6.Bitwise Operators
- → 7.Special Operators

ARITHMETIC OPERATORS

Operators	Meaning	Example
+	Addition	2+9 = 11
-	Subtraction	9-2 = 7
/	Division	4 / 2 = 2
*	Multiplication	3 * 2 = 6
%	Modulo Division	4 % 2 = 0

Division operator returns Quotient and Modulo operator Returns Reminder.

Unary & Binary Operators

RELATIONAL OPERATORS

Operators	Meaning	Example	Returns
<	Is less than	2 < 9	1
<=	Less than or equal to	5<=2	0
>	Greater than	4>9	0
>=	Greater than or equal to	3>=1	1
==	Equal to	2==3	0
!=	Is not equal to	4!=4	0

Relational Operators Compares Right hand side value along with left hand side Value and returns either 1 or 0 depend on the operator used.

Example

Examples:

```
#include<stdio.h>
void main( )
{
        printf("condition   Return value");
        printf("\n5!=5: %d",5!=5);
        printf("\n5==5: %d",5==5);
        printf("\n5>50: %d",5>50);
        printf("\n5<50): %d",5<50);
}
```

Output:

```
Condition    Return value
  5!=5:              0
  5==5:              1
  5>50:              0
  5<50:              1
```

LOGICAL OPERATORS

Operators	Meaning	Example	Returns
&&	Logical AND	(9>2)&&(17>2) (9>2)&&(17<2)	1 0
||	Logial OR	(9>2)||(17==7) (9<2)||(17==7)	1 0
!	Logical NOT	29!=29 20!=29	0 1

Example

Examples:

```
#include<stdio.h>
void main( )
{
        int a=1;
        printf("condition   Return value");
        printf("\n1.%d",((2>5) && (5>2)));
        printf("\n2. %d", ((2>5) || (5>2)));
        printf("\n3. %d", !a);
}
```

Output:

```
Condition    Return value
  1.                0
  2.                1
  3.                0
```

Increment and Decrement Operators

Increment Operator → ++ (Increments the value of a variable by 1)

Decrement Operator → -- (Decrements the value of a variable by 1)

Eg: if a=2

1.a++ => a=a+1 = 2+1 = 3

2.a-- => a=a-1 = 2-1 = 1

Types of Inc and Dec Operators

1. Post Increment → a++
2. Post Decrement → a--
3. Pre Increment → ++a
4. Pre Decrement → --a

→Placing the operator after the variable is Post. In pos operation(Read/Write) is performed on the variable only then the value is incremented/Decremented by 1.

→Placing the operator before the variable is Pre. In pre the value is Incremented/Decremented by 1 onl then the operation(Read/Write) is performed.

Example

Examples:

```
#include<stdio.h>
void main( )
{
        int a=10;
        printf(“%d”,a);
        printf(“%d”,a++);
        printf(“%d”,a);
        printf(“%d”,++a);
        printf(“%d”,a);
        printf(“%d”,--a);
        printf(“%d”,a);
}
```

Output:

10 10 11 12 12 11 11

Assignment Operator

→Assignment operator is used to assign values in right hand side to the left hand side variable

Ex:

```
x = 10;
x = a + b;
x = y = z (Nested or Multiple assignment operator)
x = y = 2
```

Compound Assignment Operator

- **Combination of arithmetic and assignment operator is said to be Compound assignment operator.**

Operators	Example	Example
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Conditional or Ternary Operator

- It checks the condition and executes the statement depending on the condition.

Syntax	Condition ? exp1 : exp2
Description	"?" acts as Ternary Operator, It first evaluates the condition, if it is true then exp1 is evaluated else exp2 is evaluated (It is just similar to if – else statement)
Example	Int a=5,b=3; Big=(a>b) ? a : b Printf("Big is %d",big);

Conditional or Ternary Operator

- Write a ‘C’ Program to check whether the input number is ODD or EVEN number using “if” statement.
- Write a ‘C’ Program to check whether the input number is ODD or EVEN number without using “if” statement.

BITWISE OPERATOR

- It is used to manipulate the data at bit level. It operates only on integers

Operators	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Shift Left
>>	Shift Right
~	Compliment

SPECIAL OPERATOR

→ Special operators are used for special purposes.

Operators	Meaning
,	Comma Operators
Sizeof( )	Size of Operator(it returns the size of variable or data type)
& and *	Pointer operator(& - address of operator and * - Value at address operator
. and →	Member access operators (Structure)

OPERATOR PRECEDENCE

*Mathematical Evaluation of an Expression (BODMAS) rule is applied

*Evaluating expression using computer applies Rule of Precedence.

Rule of Precedence:

Precedence	Operators
High Precedence	( ) Paranthesis
	^ - Exponential / & * -Division & Multiplication + & - Add and Sub
Low Precedence	= (Assignment operator)

Associativity of Operators in C

When precedence of two operators are same then associatively of operator is considered for evaluation

OPERATOR	DESCRIPTION	ASSOCIATIVITY
( )	Parentheses (function call) (see Note 1)	left-to-right
[ ]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++ —	Postfix increment/decrement (see Note 2)	
++ —	Prefix increment/decrement	right-to-left
+ –	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(*type*)	Cast (convert value to temporary value of *type*)	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes on this implementation	
* / %	Multiplication/division/modulus	left-to-right
+ –	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right

|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= |=	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	
,	Comma (separate expressions)	left-to-right

Example

Evaluate:

3 – 9 / 3 + 77 * 3 -1

??????

Example

Evaluate:

3 – 9 / 3 + 77 * 3 -1

=> 3 – (9 / 3) + 77 * 3 -1

=> 3 – 3 + 77 * 3 -1

=> 3 – 3 + (77 * 3) -1

=> 3 – 3 + 231 – 1

=> (3 – 3) + 231 – 1 (Equal Precedence evaluate from left to right)

=> (0 + 231) -1

=> 231 – 1

=> 230

Example

Evaluate:

(8/2) – 6/3 + 7-2 * (4%3)+ 3 -1

??????

Statements in C

1.Input and Output statements

- 1.1 Formatted and unformatted i/p

2.Decision Making

- 2.1. if – Statement
- 2.2. if – else statement
- 2.3. if – else if – else (Multiple if stmt)
- 2.4. Nested if

3.Selective statement (Switch - case)

4.Looping Statement

- 4.1.While – loop
- 4.2.do – While Loop
- 4.3. for - Loop

Statements in C

5. Control Transfer Statements

5.1. Break (Block Terminator)

5.2. Continue (Skips an iteration)

Both Break and Continue are used in Loop and is executed based on the condition.

5.3. Exit( ) – used to terminate the program

5.4. Go to - Transfers the control from current statement to any of the statement in a program.

Input and Output statements

Unformatted I/O Statements.	Formatted I/O Statements.
1.getchar( ); 2.Putchar( ); 3.gets( ); 4.puts( ); 5.getch( );	1.printf( ); 2.scanf( );

Formatted input – output statement

Input statement:

Scanf( ) – used to read any type of values From the standard input device (Key Board).

Syntax	Scanf(" control string",&var1,&var2,..&varn);
Description	The control string must begin with a percentage sign % followed by conversion character. & - " Address of" Operator
Example	int a,b,c Scanf(" %d %d %d",&a,&b,&c);

Control Strings

Control String	Meaning
%c	Single character
%d	Integer
%s	Strings
%f	Float
%ld	Long int

Formatted input – output statement

Output statement:

printf( ) – used to print text or variables values in output screen.

From the standard input device (Key Board).

Syntax	1.printf(" Text"); The text specified in " " will be printed in output screen. 2.Printf(" control string",var1,var2,…varn); Specified values of the variables will be printed based on control string.
Example	int a=1 printf(" print the value of a:"); Printf("%d",a);

Unformatted I/O Statement

Single character i/p function:

1.getchar( ) – Single character i/p function.

2.getch( ) – Single character i/p function.

Syntax	Char variable = getchar( );
Description	The getchar( ) function reads a single character as i/p from the keyboard and assigns it to the character variable.
Example	char c; c=getchar( );//reads single character.

Unformatted I/O Statement

Single character o/p function:

1.putchar( ) – Single character o/p function.

Syntax	putchar(char variable);
Description	The putchar( ) function prints the value of character variable to output screen.
Example	char c='A'; OutPut: putchar(c); A

Example

```
#include<stdio.h>
#include<conio.h>
Void main( )
{
char c;
Printf(“Enter the character:”);
Scanf(“%c”,&c);// c = getchar( );
Printf(“%c”,c);// putchar( c );
Printf(“%d”,c);
```

Output:
Enter the character: A
A
65 // Ascii value of A is 65 and a is 97

String I/O function

1.gets( ) – Reads a string as input
2.puts( ) – prints a string in output screen.

Syntax	string variable = gets( );
Description	The gets( ) function reads a string i/p from the keyboard and assigns it to the string variable.

Example	char name[10] ;//String variable declaration
	name=gets( );//reads string character.

String I/O function

puts( ) – prints a string in output screen.

Syntax	1.puts(string variable); 2.Puts(“ Text ”);
Description	The puts( ) function prints a string to output screen. It can print either a string variable value or prints a text with in “ “

Example	char c[10] ;//String variable declaration
	c=gets( );//reads string character.
	Puts(“ Value of c is: “);
	Puts(c);//prints value of c
	Output:
	God
	Value of c is: God

Example

```
#include<stdio.h>
#include<conio.h>
Void main( )
{
char name[ 10 ];
Printf(“Enter the string:”);//puts(“Enter the
   string”);
Scanf(“%s”,&name);// name=gets( );
Printf(“%s”,name);// puts( name );
```

Output:
Enter the string: God
God

Character Testing Function

Function	Returns	Test
Isalpha(a)	1-if true 0- if false	Tests whether a is alphabet
Isdigit(a)	1-if true 0- if false	Tests whether a is digit
Isupper(a)	1-if true 0- if false	Tests whether a is upper case or not
Islower(a)	1-if true 0- if false	Tests whether a is Lower case or not
toupper(a)	1-if true 0- if false	Converts a to uppercase
tolower(a)	1-if true 0- if false	Converts a to lowercase

Example

```
#include<stdio.h>
#include<ctype.h>
Void main( )
{
Char ch;
Puts("Enter the char:");
ch=getchar( );
if(isalpha(ch)
   printf("alphabet");
if(isupper(ch)
   printf("upper case");
if(islower(ch)
   printf("Lower case");
}
```

output:

```
Enter the char:A
alphabet
uppercase
```

Example

```
#include<stdio.h>
#include<ctype.h>
Void main( )
{
Char ch;
Puts("Enter the char:");
ch=getchar( );
if(isupper(ch)
{
   printf("upper case");
   printf("%c",tolower(ch));
}
else
{
   printf("lower case");
   printf("%c",toupper(ch));
}
```

output:

```
Enter the char: A
uppercase
a

Enter the char: a
lower case
A
```

Converting Upper to Lower and Lower to upper without using inbuilt functions

```
#include<stdio.h>
#include<conio.h>
Void main( )
{
Char ch;
Printf("Enter the character:");
ch=getchar( );
if( (ch>='a') && (ch<='z'))
    ch=ch-32;
else if( (ch>='A') && (ch<='Z'))
    ch=ch+32;
else
    printf("\nEnter a valid alphabet");
Printf("%c",ch);
}
Output:
Enter the character:A
a
Enter the character:f
F
```

Conditional Statements

1. If – Statement without else

2. If – else Statement

3. If – else – if Statement
   (Multiple – if Statement)

4. Nested – if Statement

If - Statement

Syntax:

```
if ( Condition )
{
        Statements;
        ---------------;
}
```

Ex:

```
int a=1,b=2,c=3;
big=a;
if( b > big )
        big=b;
if( c > big )
        big=c;
printf("Big = %d", big);
```

If - else Statement

Syntax:

```
if ( Condition )
{
        Statements;
        ----------;
}
else
{
        Statements;
        ----------;
}
```

Ex:

```
int a=1,b=2;
if( a > b )
        printf(" a is big");
else
printf(" b is big");
```

If – else - if Statement

Syntax:

```
if ( Condition )
{
        Statements;
        ----------;
}
else if ( Condition )
{
        Statements;
        ----------;
}
....    ......
....    ......

else
{
        Statements;
        ----------;
}
```

Example

1. **Write a C Program to find Biggest of 3 Nos.**
2. **Write a C program to print Distinction, 1st class,2nd class and fail based on conditions**
   1. **avg > = 75 → Distinction**
   2. **(avg>=60)&&(avg<75) → 1st class**
   3. **(avg>=45)&&(avg<60) → 2nd class**
   4. **(avg<45) → Fail**

Nested - If Statement

Syntax:

```
if ( Condition )
{
        if ( Condition )

        {
                Statements;
                ---------------;
        }
        else
        {
                Statements;
                ---------------;
        }
}
else
{
        Statements;
        ---------------;
}
```

Multiple Selection with if

```
if (day == 0 ) {
    printf (“Sunday”) ;
}
if (day == 1 ) {
    printf (“Monday”) ;
}
if (day == 2) {
    printf (“Tuesday”) ;
}
if (day == 3) {
    printf (“Wednesday”)
     ;
}
```

(continued)

```
if (day == 4) {
    printf (“Thursday”) ;
}
if (day == 5) {
    printf (“Friday”) ;
}
if (day == 6) {
    printf (“Saturday”) ;
}
if ((day < 0) || (day > 6)) {
    printf(“Error - invalid day.\n”) ;
}
```

Multiple Selection with if-else

```
if (day == 0 ) {
    printf (“Sunday”) ;
} else if (day == 1 ) {
    printf (“Monday”) ;
} else if (day == 2) {
    printf (“Tuesday”) ;
} else if (day == 3) {
    printf (“Wednesday”) ;
} else if (day == 4) {
    printf (“Thursday”) ;
} else if (day == 5) {
    printf (“Friday”) ;
} else if (day = 6) {
    printf (“Saturday”) ;
} else {
    printf (“Error - invalid
      day.\n”) ;
}
```

This if-else structure is more efficient than the corresponding if structure. Why?

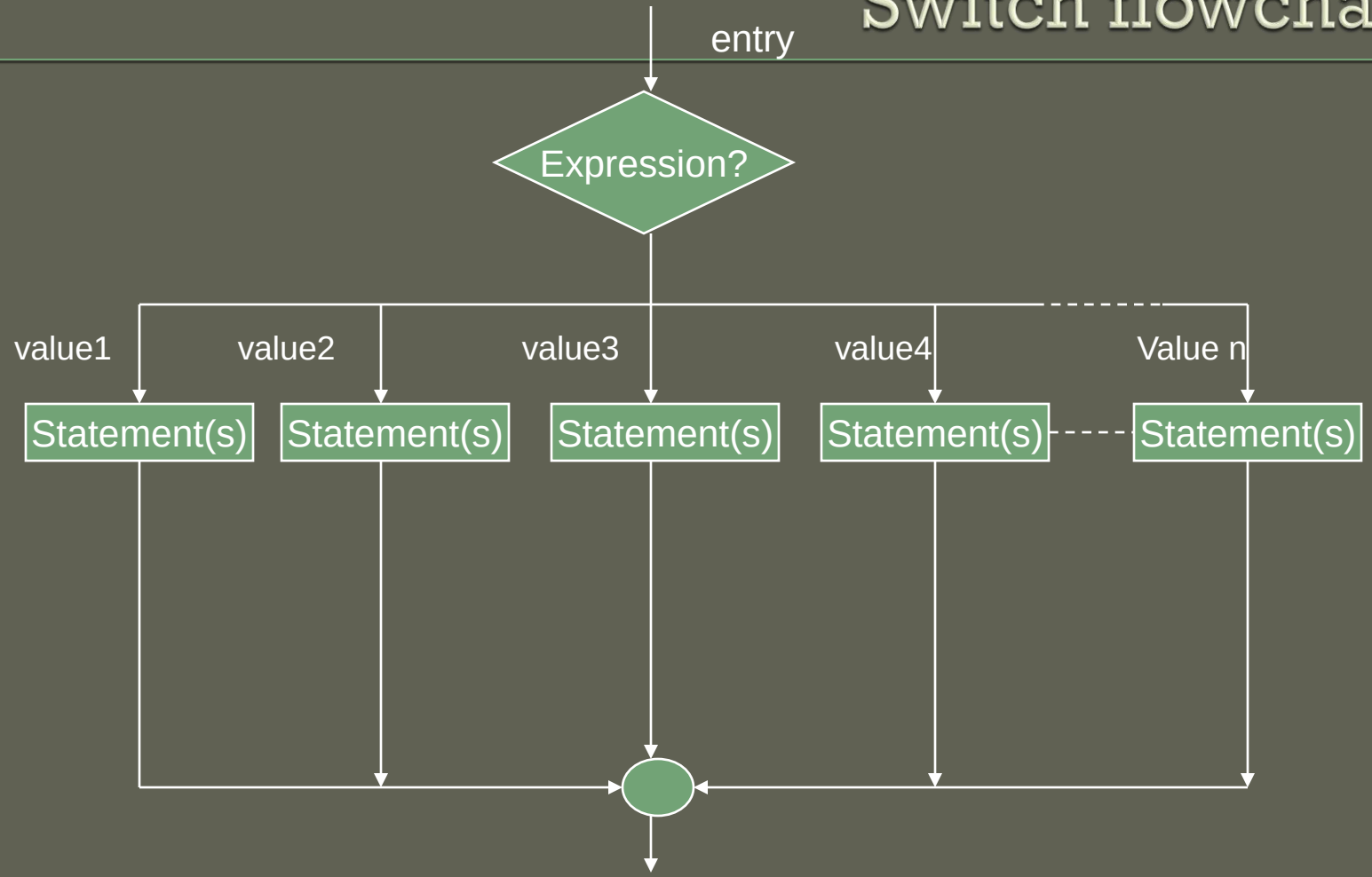
CASE STATEMENT

Syntax:

```
Switch( expression or value)
{
        Case 1:  statements;
                 ----------------;
                 break;
        Case 2:  statements;
                 ----------------;
                 break;
        Case 3:  statements;
                 ----------------;
                 break;
        ……       ……..
        ……       ……..

        Default: statements;
}
```

Switch flowchart



CASE STATEMENT

Syntax:

```
Switch( Character )
{
Case ‘c’:  statements;
           ----------;
Case ‘c’:  statements;
           ----------;
Case ‘c’:  statements;
           ----------;
…..        ……….;

Default:  statements;
}
Where c is any character
```

Sample Program

```
#include<stdio.h>
main()
{
    int choice;
    choice = 0;
    printf( “my menu\n\n” );
    printf( “1 – edit\n 2 – delete 3 – quit”);
    printf( “enter your choice: ”);
    scanf( “%d”, &choice  );
    switch( choice )
    {
    case 1: printf( “Do edit\n” );
            break;
    case 2:  printf( “Do delete\n” );
             break;
    case 3:  printf( “Done\n” );
            break;
    default: printf( “Invalid choice!\n” );
            break;
    }
}
```

Sample Program

```
#include<stdio.h>
main()
{
    char ch;
    printf( “Vowels\n” );

    printf( “enter  the character: ”);
    scanf( “%c”, &ch  );
    switch( choice )
    {
    case ‘a’: printf( “Vowel” );
              break;
    case ‘e’: printf( “Vowel” );
              break;
    case ‘i’: printf( “Vowel” );
              break;
    case ‘o’: printf( “Vowel” );
              break;
    case ‘u’: printf( “Vowel” );
              break;
    default: printf( “Invalid choice!\n” );
             break;
    }
}
```

```
#include<stdio.h>
#include<conio.h>
main()
{
    char c;
    printf("**********Vowel
     check************\n");
    printf("Enter the character:");
    c=getchar();
    switch(c)
    {

          case 'a‘ :
          case 'e‘ :
          case 'i‘  :
          case 'o‘ :
          case 'u‘ : printf("\nVowel");
                     break;
           default : printf("\nNot a Vowel");
    }
    getch();
}
```

While Loop (Top Tested)

Syntax:

```
While( condition )
{
        Statements;
        ---------------;
}
```

Loops(while)

```
#include <stdio.h>

Void main()
{
  int x = 0;  /* Don't forget to declare variables */
  while ( x < 10 )
  {   /* While x is less than 10 */
      printf( "%d\n", x );
      x++;    /* Update x so the condition can be met
   eventually */
  }
}
```

Do-While Loop

Syntax:

```
do
{
        Statements;
        ---------------;
} While( condition );
```

Loops(do while)

```
#include <stdio.h>
void main()
{
  int x;
  x = 0;
  do
  {
    /* value of x is printed at least one time
      even though the condition is false*/
     printf( "%d\n", x );
   x++;
  } while ( x != 10 );
}
```

Exercise

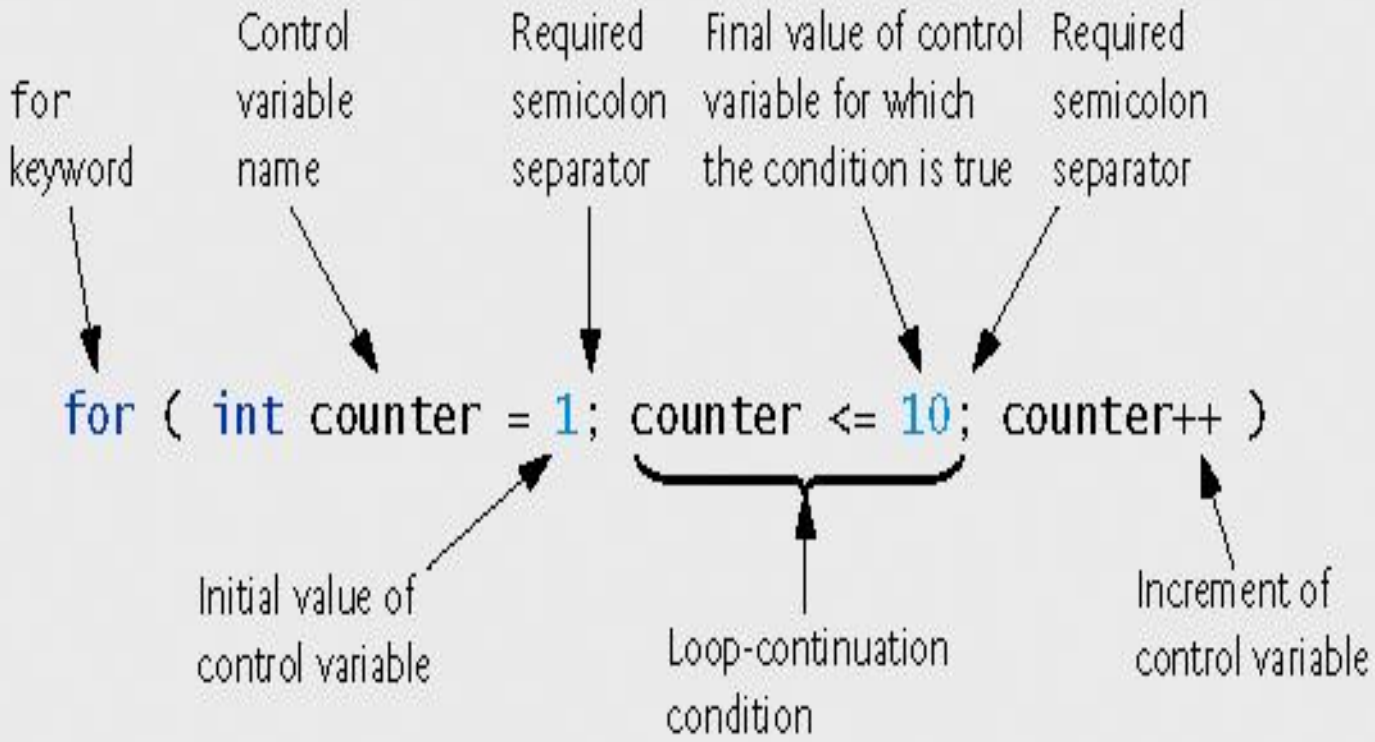
1. Write a ‘C’ program to find sum of ‘n’ natural numbers using while and do-while { sum=1+2+3+…n}
2. Write a ‘C program to find factorial of ‘n’ numbers using while and do-while { factorial = 4! = 4*3*2*1 = 24}
3. Write a ‘C program to reverse a given number using while and do-while. { i/p = 123 ; o/p = 321}
4. Write a ‘C’ Program to check whether the i/p number is palindrom or not?
5. Write a ‘C program to find sum of digits using while and do-while. { i/p = 123 ; o/p = 1+2+3 = 6}

For Loop


Syntax:
1
2
4
for(Initialization; Condition; Inc / Dec)
{
Statements;
----------;
}
3

For Loop

- for repetition statement
  - Specifies counter-controlled repetition details in a single line of code



for Loop (Cont.)

- for statement examples
  - Vary control variable from 1 to 100 in increments of 1
    - `for (i = 1; i <= 100; i++ )`
  - Vary control variable from 100 to 1 in increments of -1
    - `for (i = 100; i >= 1; i-- )`
  - Vary control variable from 7 to 77 in steps of 7
    - `for (i = 7; i <= 77; i += 7 )`
  - Vary control variable from 20 to 2 in steps of -2
    - `for (i = 20; i >= 2; i -= 2 )`
  - Vary control variable over the sequence: 2, 5, 8, 11, 14, 17, 20
    - `for (i = 2; i <= 20; i += 3 )`
  - Vary control variable over the sequence: 99, 88, 77, 66, 55, 44, 33, 22, 11, 0
    - `for (i = 99; i >= 0; i -= 11 )`

For - Loops

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

Output:

1 2 3 4 5 6 7 8 9

Nested For Loop

Syntax:

```
for( Initialization; Condition; Inc / Dec)//Outer Loop
{

        for( Initialization; Condition; Inc / Dec)//Inner Loop
        {
                Statements;
                ---------------;
        }
}
```

Example

```
#include<stdio.h>
#include<conio.h>
Void main( )
{
int n,i,j;
Clrscr( );
Printf(“enter the value of n:”);
Scanf(“%d”,&n);
for(i=0;i<n;i++)
{
   for(j=i;j<n;j++)
        {
              printf(“*”);
        }
Printf(“\n”);
}
}
```

Output:

```
----------
Enter the value of n:4
* * * *
* * *
* *
*
```

Example

```
#include<stdio.h>
#include<conio.h>
Void main( )
{
int n,i,j;
Clrscr( );
Printf("Enter the value of n:");
Scanf("%d",&n);
for(i=1;i<=n;i++)
{
   for(j=i;j<=n;j++)
        printf("%d",j);
Printf("\n");
}
```

Output:

```
Enter the value of n:4
1 2 3 4
2 3 4
3 4
4
```

Try it out

1.
```
1 2 3 4
1 2 3
1 2
1
```

2.
```
1 0 1 0
1 0 1
1 0
1
```

3.
```
1
1 2
1 2 3
1 2 3 4
```

4.
```
1
1 0
1 0 1
1 0 1 0
```

1. Write a ‘C’ Program to check whether the i/p number is Palindrome or not?

---

2. Write a ‘C’ program to generate Fibonacci series
   (Series = 0 1 1 2 3 5 ……)

3. Write a ‘C’ program to read ‘n’ numbers and count the total number of +ve, -ve and zeros entered.

4. i/p = 1246 o/p = 2357

5. Write a ‘C’ program to generate multiplication table.

Palindrome

```
#include<stdio.h>
Void main()
{
int m,n, rev=0,a;
Printf("Enter the number:");
Scanf("%d",&n);
a=n;
While(n!=0)
{
    m=n%10;
    rev = (rev*10) + m;
    n=n/10;
}
If(a==rev)
    printf("The Entered number is palindrome");
else
    printf("The Entered number is not a palindrome");
}
```

Output:

Enter the number:121

The Entered number is palindrome

Binary to Decimal

```
#include<stdio.h>
Void main()
{
int m,n,k=0,dec=0,a;
Printf("Enter the number:");
Scanf("%d",&n);
While(n!=0)
{
    m=n%10;
    dec = dec+(m * 2^k);
    n=n/10;
    k++;
}
Printf("\n The Decimal equivalent is %d",dec);
}
```

Fibonacci Series

```
#include<stdio.h>
Void main()
{
int n,a=-1,b=1,c;
Printf("Enter the number:");
Scanf("%d",&n);
While(n!=0)
{
    c=a+b;
    printf("\n%d",c);
    a=b;
    b=c;
    n- - ;
}
getch();
}
```

Generating Multiplication Table

```
#include<stdio.h>
#include<conio.h>
main()
{
    int i,j,n;
    printf("Enter the value of n:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
       printf("\n****Table%d*****",i);
              for(j=1;j<=10;j++)
                        printf("\n%d * %d = %d",i,j,i*j);
       printf("\n");
    }
getch();
}
```

1. Break (Block Terminator)
2. Continue (Skips an iteration)

Both Break and Continue are used in Loop and is executed based on the condition.

3. Exit( ) – used to terminate the program
4. Go to - Transfers the control from current statement to any of the statement in a program.

Break- Block Terminator

The break statement is used to terminate the loop. When break is executed inside a loop control automatically transferred to the first statement after the loop. Break is usually associated with an if statement.

```
While( condition )
{
   -------;
   if(condition)
   break;
   -------;
}
```

Example

```
#include <stdio.h>
#include<conio.h>
Void main( )
{
int i;
for( i = 1; i <= 5; i++)
{
   if( i == 3 )
   break;
   printf("%d",i);
}
}
```

Output:

1 2

Continue

The continue statement skips an iteration in a loop. It transfers the control to the beginning of the loop.

```
While( condition )
{
   -------;
   if(condition)
   continue;
   -------;
}
```

Example

```
#include stdio.h>
#include<conio.h>
Void main( )
{
int i;
for( i = 1; i <= 5; i++)
{
   if( i == 3 )
   continue;
   printf("%d",i);
}
```

Output:

1 2 4 5

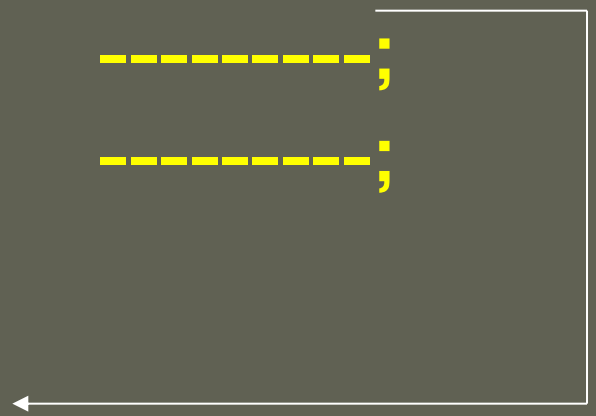
Goto Statement

A goto statement can cause program control almost anywhere in the program unconditionally.

Syntax:

```
goto label;
```

```
goto label;
        ---------;
        ---------;
label:
```



Example

```
#include stdio.h>
#include<conio.h>
#include<stdlib.h>
Void main( )
{
 int a,b;
Printf(“Enter the numbers:”);
Scanf(“%d %d”,&a,&b);
if( a == b )
    goto equal;
else
{
    printf(“A and B are not equal”);
    exit(0);
}
equal:
    printf(“ A & B are equal”);
}
```

Type conversion

- When variables of one type are mixed with variables of another type, a type conversion will occur.
- In an assignment statement, The value of the right side (expression side) of the assignment is converted to the type of the left side (target variable)

Example : Type Conversion

```
int x;
char ch;
float f;
void func(void)
{
ch = x; /* the left high-order bits of the integer variable x are
lopped off, leaving ch with the lower 8 bits */
x = f; // x will receive the nonfractional part of f
f = ch; /* f will convert the 8-bit integer value stored
in ch to the same value in the floating-point format */
f = x; /* f will convert an integer value into floating-point
format*/
}
```