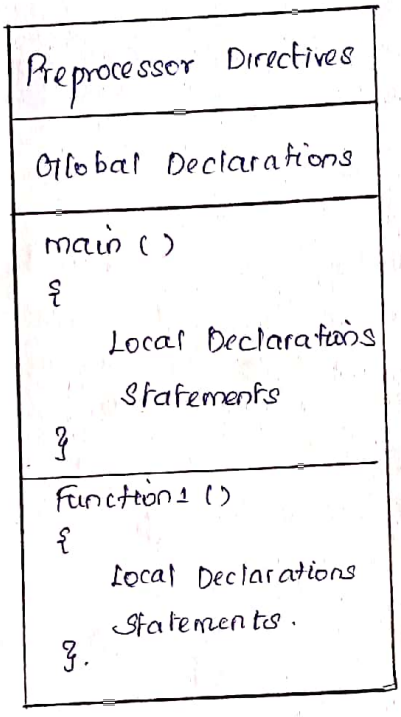
CS8251- PROGRAMMING IN C

UNIT 1.

1. Explain the Structure of C Program and the compilation process.

-> C program is composed of preprocessor commands, global declaration section and one or more functions.

-> The following diagram illustrates the general structure of C Programming.



Preprocessor Directives:

The preprocessor directives contain special instructions that include how to prepare the program for compilation. The most important preprocessor command is include which tells the compiler to execute the program.

Global Declaration:

Variables which are going to be used throughout the program will be identified and declared here.

Main Function:

Execution of C Program starts from here.

Sub function:

A C Program contains one or more functions, where a function is defined as a group of C statements. The function is divided into two parts:

a) Declaration section b) Statement Section.

The data declared within a function are called local declaration. The data will be visible within that function. In other terms, the data will be valid only till the function ends.

Eg:

```
#include <stdio.h>
int main()
{
    printf("welcome to the world");
}
```

Explanation:

#include <stdio.h>

- This is a preprocessor command.
- All the preprocessor command starts with # symbol.
- #include statement tells the compiler to include the standard input output library header file.
- It has some built-in functions. By including this file, the predefined functions can be used directly.

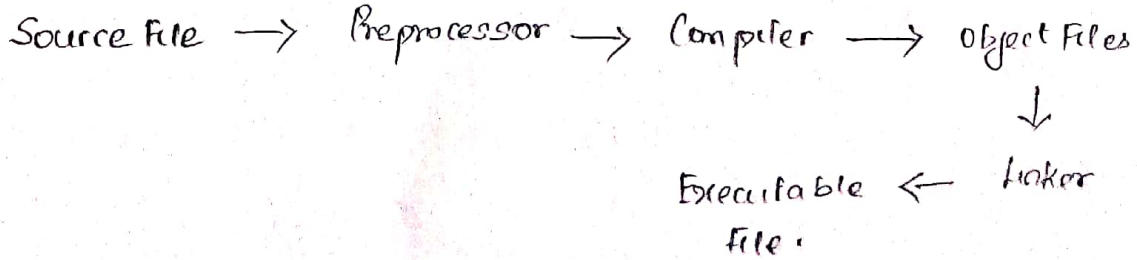
main()

- Every C Program has a main() function.
- The two curly braces { } are used to group all the related statements of the main function.

printf()

- The printf() is defined in the stdio.h file and it is used to print the text on the monitor.
- The message that has to be displayed on the screen is enclosed within double quotes.
- \n is an escape sequence and represents a new line character. It is used to print the text on the new line

COMPILATION PROCESS



8. Explain Storage classes in detail with example.

Storage class defines the scope (visibility) and life time of variables and functions within a C program.

Four different storage classes are as follows:

a. auto
b. register
c. static
d. extern.

a. auto storage class:

- The auto storage class specifier is used to explicitly declare a variable with automatic storage.

  Eg: auto int x;

  x is an integer that has automatic storage.
- The auto storage class can be used to declare variables in a block
- All local variables declared within a function belong to automatic storage class by default.
- They should be declared at the start of the program.
- Memory for the variable is automatically allocated upon entry to a block and freed automatically upon exit from that block.

Eg:
```
#include<stdio.h>
void func1()
{
  int a = 10;
  printf("\n a = %d", a);
}
void func2()
{
  int a = 20;
  printf("\n a = %d", a);
}
void main()
{
  int a = 30;
  func1();
  func2();
  printf("%d", a);
}
```

b. register storage class:

- When a variable is declared as register, it is stored in a CPU register instead of a RAM.
- Since the variable is stored in a register, the maximum size of the variable is equal to the register size.
- Register variables cannot be operated using & operator.
- These variables are used when quick access to the variable is needed.

Example:

```
#include <stdio.h>
int exp(int a, int b);
int main()
{
  int a = 3, b = 5, res;
  res = exp(a,b);
  printf("%d To the power of %d = %d", a, b, res);
  return 0;
}
int exp(int a, int b)
{
  register int res = 1;
  int i;
  for(i = 0; i <= b; i++)
    res = res * a;
  return res;
}
```

c. extern storage class:

- The extern storage class is used to give a reference of a global variable that is visible to all program files.
- These variables may be declared outside any function in a source code.
- Memory is allocated when the program begins execution and remain allocated until the program terminates.

Eg: FILE1.C

```
#include <stdio.h>
#include <FILE2.C>
int x;
void print(void);
int main()
{
  x = 10;
  printf("x in File1 is %d", x);
  print();
  return 0;
}
```

// FILE 2

```
#include<stdio.h>
extern int x;
void print()
{
  printf("\n x in File2 = %d", x);
}
main()
{
  Statements;
}
```

d. Static Storage class.

- Static is default storage class for all global variables.
- These variables have a life time over the entire program.
- Static variables defined within a function are initialized at run-time
- The difference between an auto variable and a static variable is that static variable is not re-initialized when the function is called again and again.

Eg:

```
#include <stdio.h>
void print ();
int main()
{
   print();
   print();
   return 0;
}
void print()
{
   static int x;
   int y = 0;
   printf("%d %d", x, y);
   x++;
   y++;
}
```

3. Explain the different type of operators in detail.

An operator is a symbol that specifies the mathematical, logical or relational operation to be performed.

Operators are categorized as follows.

a. Arithmetic operators.
b. Relational operators.
c. Equality operators.
d. Logical operators.
e. Unary operators.
f. Conditional operator.
g. Bitwise operator
h. Assignment operator
i. Size of operator.

a. Arithmetic operators:

- Used to perform mathematical operations like addition, subtraction, multiplication and division. Assume a = 9, b = 3, result;

Operation	Operator	Syntax / comment
Addition	+	result = a+b;
Subtraction	-	result = a-b;
Multiplication	*	result = a*b;
Division	/	result = a/b;
Modulo	%	result = a%b;

b. Relational operators:

- Relational operator is known as a comparison operator.
- It is used to compare two values.
- Expressions that contain relational operators are called relational expressions.

Operator	Meaning	Example.	Result
<	less than	3 < 5	1 → True.
>	Greater than	7 > 9	0 → False.
<=	Less than or equal to	100 <= 100	1 → True.
>=	Greater than or equal to	50 >= 100	0 → False.

c. Equality operators:

- Used to compare the operands for strict equality or inequality.

The operators are:

Operator	Meaning
==	Returns 1, if both operands a and b are equal, 0 otherwise.
!=	Returns 1 if both operands a and b do not have same value.

d. Logical operators:

Used to combine two or more relational expressions.

The operators are: (a) Logical AND (b) Logical OR (c) Logical NOT.

Logical AND

- Evaluates two relational expressions.
- If both conditions are true, the whole expression evaluates to true.
- If any one condition is false, the whole expression evaluates to false.

Logical OR

- Returns false if both operands are false.

Logical NOT

- Takes a single expression and negates the value of the
- expression

A	!A
0	1
1	0

e. Unary operator:

- These operators operate on single operand. There are three Unary operators
  a. Unary minus
  b. Increment operator
  c. decrement operator.

Unary Minus:

- When an operand is preceded by a minus sign, the unary operator negates it value.

Eg: a = 5
b = -(a)

Increment operator:

- Used to increase the value of the operand/variable by 1.
- ++ is used to denote increment operator.

Two types of operator:
a. pre-increment
b. post-increment.

++a	First increments and then displays the value.
a++	First fetch the value and then performs increment operation

Eg:

```
int x = 10, y;
y = ++x;
```

↙

This is interpreted as

```
x = x+1;
y = x;
```

If the statement is written as

```
y = x++;
```

↙

This is interpreted as

```
y = x;
x = x+1;
```

Decrement operator:

- Used to decrease the value of the operand / variable by 1.
- Represented by --
- This also have predecrement and post decrement operation.

f. Conditional operator:

- Conditional operator (?) is useful in situations in which there are two or more alternatives for an expression.

Syntax:

```
exp1 ? exp2 : exp3;
```

- exp1 is evaluated
  - → If the condition is true, exp2 is result; otherwise exp3 is result.

```
large = (a>b) ? a : b;
```

g. Bitwise operators:

- Performs operations at bitwise level.
- Bitwise AND, OR and NOT operators are available.

- Bitwise AND compares each bit of its first operand with the corresponding bit of its second operand.
  → If both bits are 1, result is 1
  otherwise 0.
- Bitwise OR compares each bit of its first operand with second operand.
  → If one or both bits are 1, result is 1
  otherwise 0.
- Bitwise XOR.
  If bits of first operand and second operand is same,
  result is 0,
  otherwise 1.

h. Assignment operator.
- Used to assign a value to a variable.

```
int a = 5;
```

i. sizeof operator:
- Used to calculate size of datatypes.

```
int a = 10;
int result;
result = sizeof(a);
```

---

4. Explain Decision making statements in detail.

Control statements enables us to specify the flow of program control i.e., the order in which the instructions in a program must be executed.

C Provides the following Decision making statements / control statements.

a. if condition
b. if-else condition
c. if-else-if condition
d. nested if
e. switch-case.

DESCRIPTION	SYNTAX	CODE

if condition:
• Single way decision making statement.
• If the condition is true, the statements are executed.
	if (condition)
	statement;
		void main()
		{
		int age = 15;
		if (age > 0)
		{
		printf("Entered age is valid");
		}
		}

if- else condition
• Two way decision making statement.
• If the condition is true, the statements are executed
• If the condition is false, the statements inside else block is executed.
	if (condition)
	statement;
	else
	statement;
		void main()
		{
		int age;
		printf("Enter the age");
		scanf("%d", &age);
		if (age > 0)
		printf("valid age");
		else
		printf("Invalid age");
		}

if-else-if
• To test additional conditions from the initial condition.
• If we have multiple choices as output we will be having a chained condition to check and display the output.
	if (condition)
	{
	statement1;
	}
	else if (condition)
	{
	statement2;
	}
	else
	{
	statement3;
	}
		void main()
		{
		int a, b, c;
		a=5, b=6, c=7;
		if ((a>b) && (a>c))
		printf("a is greater");
		else if ((b>a) && (b>c))
		printf("b is greater");
		else
		printf("c is greater");
		}

nested - if
• If condition inside another if condition is called as nested - if
	if (condition)
	{
	if (condition)
	{
	statement1;
	}
	else
	{
	statement;
	}
	}
	else
	{
	statement;
	}
		void main()
		{
		int marks;
		printf("Enter marks");
		scanf("%d", &marks);
		if (marks > 50)
		{
		if (marks > 80)
		{
		printf("A Grade");
		}
		else
		{
		printf("B Grade");
		} }
		else
		{
		printf("Failed");
		} }

DESCRIPTION	SYNTAX	CODE

switch-case:
• Multi way decision making statement
• Switch statements are used as an alternative to long if statements that compare a variable to several integral values
• In the syntax, a keyword called break is used. The break statement is used at the end of each case.
• The expression in switch statement must be an integer value or character constant
	switch (expression)
	{
	case 1 :
	stmt;
	break;
	case 2 :
	stmt;
	break;
	:
	case n:
	stmt;
	break;
	default:
	stmt;
	break;
	}
		void main ()
		{
		char grade = 'a';
		switch (grade)
		{
		case 'o':
		printf("Outstanding Student");
		break;
		case 'a':
		printf(" Excellent Student")
		break;
		case 'b':
		printf(" Good Student");
		break;
		case 'f':
		printf (" Failed");
		break;
		default:
		} } printf("Invalid grade");

5. Explain looping statements in detail:

Looping process can be defined as repeated the same process multiple times until a specific condition satisfies. It simplifies the complex problem into easy ones and also enables us to alter the flow of the program.

C Program provides the following looping statements:

a. while loop
b. do-while loop
c. for loop
d. break;
e. continue.

DESCRIPTION	SYNTAX	CODE

while loop:
• Top tested loop
• The condition is executed at the top
• If the condition is true, the control enters into the loop
• If the condition is false, the statement outside the loop gets executed.
	while (condition)
	{
	//statements;
	// increment or decrement;
	}
	statementx;
		void main ()
		{
		int i = 0;
		while (i <= 5)
		{
		printf ("%d", i);
		i++;
		}
		}

DESCRIPTION	SYNTAX	CODE

do-while loop:
• Bottom tested loop
• Statements are executed atleast once.
• Then the condition is checked.
• If the condition is true, the loop is executed.
• If the condition is false, it terminates.
	do
	{
	// statement;
	// increment/decrement;
	}while(condition);
	statement x;
		void main()
		{
		int i = 0;
		do
		{ printf("%d", i);
		i++;
		}while(i<=5);
		}

for loop:
• Used to iterate the statements or a part of the program.
• With the initialized value, the condition is checked.
• If the condition is true, the control enters the loop
• Then the increment/decrement operation is performed
• This happens until the condition fails
	for(initialization; condition; inc/dec)
	{
	// statements;
	}
		void main()
		{
		int i;
		for(i=0; i<100; i++)
		{
		printf("%d", i);
		}
		}.

break statement:
• When the break statement is executed the control comes out of the loop.
• Remaining iterations will be terminated
	break	void main()
		{
		int i;
		for(i=1; i<=5; i++)
		{
		if(i==3)
		{ break;
		}
		printf("%d", i);
		} }
		O/p: 1, 2

Continue statement:
• When the continue statement is executed, the control skips the current iteration and goes to the beginning of the loop.
	continue	void main()
		{
		int i;
		for(i=1; i<=5; i++)
		{
		if(i==3)
		{ continue;
		}
		printf("%d", i);
		}
		}
		O/p: 1, 2, 4, 5.