INTRODUCTION TO PROBLEM SOLVING USING C

Module-2

Character Set, Structure of a ‘C’ Program, Data Types, Operators, Expressions, Assignment Statement, Conditional Statements, Looping Statements, Nested Looping Statements, Multi Branching Statement (Switch), Break and Continue, Differences between Break and Continue, Unconditional Branching (Go to Statement)

C - PROGRAM STRUCTURE

A C program basically consists of the following parts:

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments
-

Let us look at a simple code that would print the words "Hello World":

```
#include <stdio.h>
int main()
{
 /* my first program in C */
 printf("Hello, World! \n");

 return 0;
}
```

1. The first line of the program #include <stdio.h> is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.

2. The next line int main() is the main function where the program execution begins.

3. The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.

4. The next line printf(...) is another function available in C which causes the message "Hello, World!" to be displayed on the screen. 5. The next line return 0; terminates the main() function and returns the value 0.

C - Data Types

Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in C can be classified as follows −

Sr.No.	Types & Description
1	**Basic Types** They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.
2	**Enumerated types** They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.
3	**The type void** The type specifier *void* indicates that no value is available.
4	**Derived types** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

The array types and structure types are referred collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see the basic types in the following section, where as other types will be covered in the upcoming chapters.

Integer Types

The following table provides the details of standard integer types with their storage sizes and value ranges −

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 bytes	-32,768 to 32,767
unsigned int	2 bytes	0 to 65,535
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	**-2^{32} to $+2^{32}$**
unsigned long	8 bytes	0 to **$+2^{64}$**

Floating-Point Types

The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision −

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

C - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

We will, in this chapter, look into the way each operator works.

Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then −

Show Examples

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Relational Operators

The following table shows all the relational operators supported by C. Assume variable

A holds 10 and variable **B** holds 20 then −

Show Examples

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then −

Show Examples

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation.

The truth tables for &, |, and ^ is as follows −

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Expressions in C

These are built from combinations of operators, let’s see them as described below.

1. Arithmetic Expressions
Addition (+), Subtraction(-), Multiplication(*), Division(/), Modulus(%), Increment(++) and Decrement(–) operators are said to “Arithmetic expressions”. This operator works in between operands. like A+B, A-B, A–, A++ etc.

2. Relational Expressions
== (equal to), != (not equal to), != (not equal to), > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to) operators are said to “Relational expressions”.This operators works in between operands. Used for comparing purpose. Like A==B, A!=B, A>B, A<B etc

3. Logical Expressions
&&(Logical and), ||(Logical or) and !(Logical not) operators are said to “Logical expressions”. Used to perform a logical operation. This operator works in between operands. Like A&&B, A||B,A!B etc.

4. Conditional Expressions
?(Question mark) and :(colon) are said to “Conditional expressions”. Used to perform a conditional check. It has 3 expressions first expression is condition. If it is true then execute expression2 and if it is false then execute expression3. Like (A>B)?”A is Big”:”B is Big”.

Assignment Statement with Operators

The following table lists the assignment operators help us to built assignment statement

Show Examples

Operator	Description	Example of Assignment Statement
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A

-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A

Control Structures- if Selection Statement

In decision control statements (if-else and nested if), group of statements are executed when condition is true. If condition is false, then else part statements are executed.

There are 3 types of decision making control statements in C language. They are,

- **Simple if statements**
- **if else statements**
- **nested if statements**

Simple if statements

Syntax for each C decision control statements are

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

In these type of statements, if condition is true, then respective block of code is executed.

Example

```
void main()
{
  int m=40,n=40;
  if (m == n)
  {
  printf("m and n are equal");
  }
}
```

if … else statements

In these type of statements, group of statements are executed when condition is true. If condition is false, then else part statements are executed.

Syntax:

```
if (condition)
{ Statement1; Statement2; }
else
{ Statement3; Statement4; }
```

Example

```
#include <stdio.h>
void main()
{
  int m=40,n=20;
  if (m == n)
  {
  printf("m and n are equal");
  }
  else
  {
  printf("m and n are not equal");
  }
}
```

Nested If statements

If condition 1 is false, then condition 2 is checked and statements are executed if it is true. If condition 2 also gets failure, then else part is executed.

Syntax:

```
if (condition1)
{
            if(condition2){
            Statement1…
            }
} else
{
Statement2…
}
```

Example : Find largest number among 3 numbers

```
#include <stdio.h>
void  main()
{
  int a=23,b=45,c=34;
   if (a>b) {
        if(a>c)
                   printf("Large = %d",a);
      else
                   printf("Large = %d",c);
  }else {
        if(b>c)
                   printf("Large = %d",b);
      else
                   printf("Large = %d",c);
  }
  getch();
}
```

Loop control statements

Loop control statements in C are used to perform looping operations until the given condition is true. Control comes out of the loop statements once condition becomes false.
There are 3 types of loop control statements in C language. They are,

1. for
2. while
3. do-while

1. for loop
   for loop is a statement which allows code to be repeatedly executed. For loop contains 3 parts Initialization, Condition and Increment or Decrements.

   Syntax
   ```
   for (exp1; exp2; expr3)
   {
   statements;
   }
   ```
   Example
   ```
   void main()
   {
   int i;
   clrscr();
   for(i=1;i<5;i++)
   {
   printf("\n%d",i);
   }
   getch();
   }
   ```

2. **do…while() loop**
   A do-while loop is similar to a while loop, except that a do-while loop is execute at least one time.
   A do while loop is a control flow statement that executes a block of code at least once, and then repeatedly executes the block, or not, depending on a given condition at the end of the block (in while).

Syntax

```
do {

statements;

}while (condition);
```

Example

```
void main()
{
int i;
i=1;
do
{
    printf("\n%d",i);
    i++;
}while(i<5);
getch(); }
```

Multi Branching Statement (Switch)

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

Syntax

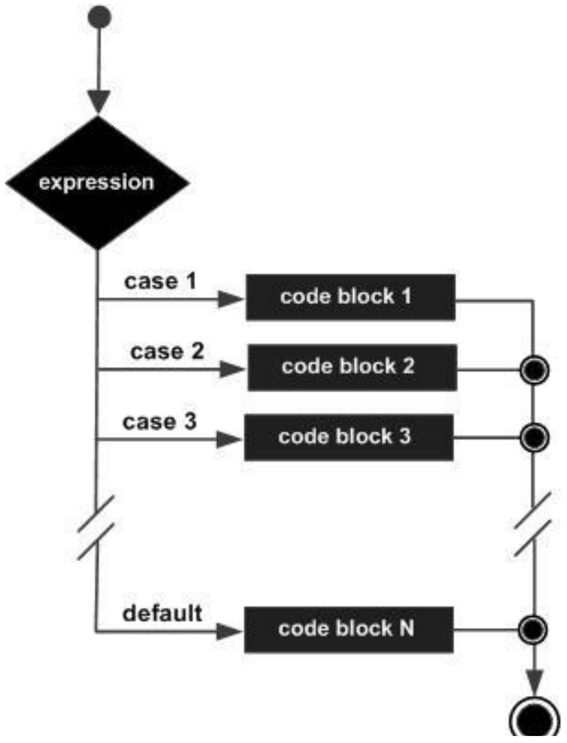
The syntax for a **switch** statement in C programming language is as follows −

```
switch(expression) {

   case constant: statement(s);
      break;

   case constant  :  statement(s);
      break;

   default :    statement(s);
}
```



```
#include <stdio.h>

int main () {

   /* local variable definition */
   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 );

   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Well done
Your grade is B
```

break and continue statements

C provides two commands to control how we loop:

- break -- exit form loop or switch.
- continue -- skip 1 iteration of loop.

You already have seen example of using break statement. Here is an example showing usage of **continue** statement.

```
#include
main()
{
   int i;
   for( i = 0; i <= 5; i ++ )
   {
     if( i == 3 )
     {
       continue;
     }
     printf("Hello %d\n", i );
   }
}
```

Output

```
Hello 0
Hello 1
Hello 2
Hello 4
Hello 5
```

In the above program when i value is 3 then control goes to the for loop again however it skip the printf statement.

Difference Between break and continue

break	**continue**
A break can appear in both switch and loop (for, while, do) statements.	A continue can appear only in loop (for, while, do) statements.
A break causes the control comes of the loop or switch	A continue doesn't terminate the loop, The control goes to beginning of the loop
The break statement can be used in both switch and loop statements.	The continue statement can appear only in loops.
A break causes the innermost enclosing loop or switch to be exited immediately.	A continue inside a loop nested within a switch causes the next loop iteration.

Goto Statement

A goto statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

NOTE − Use of goto statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.

Syntax

The syntax for a goto statement in C is as follows −

```
goto label;
..
.
label: statement;
```

Example

```
#include <stdio.h>
void main () {
  int a = 0;
start:
     a = a + 1;
    printf("value of a: %d\n", a);
    if(a<15)       goto start;
}
```

In the above example printf() function repeat without loop. Using goto we can transfer the control at label location(start)