BYJU'S EXAM PREP

Vision 2024

A Course for GATE & PSUs

Computer Science Engineering

Programming and Data structures

CHAPTER 1

C Programming

CHAPTER 1

PROGRAMMING & DATA STRUCTURES

C PROGRAMMING

All C programs must have a function in it named as **main().**

(a) Execution starts in function **main**

(b) C is **case sensitive**

(c) Comments start with /* and end with */. Comments may span over many lines.

(d) C is a "free format" language

(e) All C statements must end in a semicolon (;).

(f) The **#include <stdio.h>** instruction instructs the C compiler to insert the entire contents of file stdio.h in its place and compile the resulting file.

1. PRINTF () FUNCTION IN C LANGUAGE

- In C, printf() function is used to print on the output screen the "character, string, float value, integer value, octal value, and hexadecimal value"

Data Type	Format Specifier
Int	%d
char	%c
float	%f
double	%lf
short int	%hd
unsigned int	%u
long int	%li
long long int	%lli
unsigned long int	%lu
unsigned long long int	%llu
signed char	%c
unsigned char	%c
long double	%Lf

Table 1.1: Data types and format specifiers

Here's a list of commonly used C data types and their format specifiers.

Example Program for printf in C

1.
```
#include <stdio.h>
int main()
{
  char ch = 'A';
  char str[20] = "byjus.co ";
  float flt = 10.234;
  int no = 150;
  double dbl = 20.123456;
  printf("Character is %c \n", ch);
  printf("String is %s \n" , str);
  printf("Float value is %f \n", flt);
  printf("Integer value is %d\n" , no);
  printf("Double value is %lf \n", dbl);
  printf("Double value upto 2 decimal is %2lf \n", dbl);
  printf("Octal value is %o \n", no);
  printf("Hexadecimal value is %x \n", no);
  return 0;
}
```

Output:

```
Character is A
String is byjus.co
Float value is 10.234000
Integer value is 150
Double value is 20.123456
Double value upto 2 decimal is 20.12
Octal value is 226
Hexadecimal value is 96
```

2.
```
#include <stdio.h>
int main()
{
   // Displays the string inside quotations
   printf("C Programming");
   return 0;
}
```

Output

C Programming.

3.
```
#include <stdio.h>
int main()
{
   int testInteger = 5;
   printf("Number = %d", testInteger);
   return 0;
}
```

Output

Number = 5

2. SCANF() FUNCTION IN C LANGUAGE

- The scanf() function is used in the programming language C to read the character, string and numeric keyboard data.
- Consider the example below where a character is entered by a user. This is assigned to and then displayed in the variable "ch."
- Then the user enters a string and this value is assigned to the variable "str" and then displayed.

Example-

1.
```
#include <stdio.h>
 int main()
 {
        char ch;
        char str[100];
        printf("Enter any character \n");
        scanf("%c", &ch);
        printf("Entered character is %c \n", ch);
        printf("Enter any string ( upto 100 character ) \n");
        scanf("%s", &str);
        printf("Entered string is %s \n", str);
 }
```

Output:

Enter any character

a

Entered character is a

Enter any string (upto 100 character)

hero

Entered string is hero

- The format specifier %d is used in scanf() statement. So, the value entered is received as an integer and %s for string.
- Ampersand is used before the variable name "ch" in scanf() statement as &ch.

2.
```
#include <stdio.h>
int main () {
char str1[20], str2[30];

printf("Enter name: ");
scanf("%s", str1);

printf("Enter your website name: ");
scanf("%s", str2);

printf("Entered Name: %s\n", str1);
printf("Entered Website:%s", str2);

return(0);
}
```
Output:
```
Enter name: admin
Enter your website name:xyx

Entered Name: admin
Entered Website:xyz
```

3.
```
#include <stdio.h>
int main()
{
  int a;
  scanf("This is the value %d", &a);
  printf("Input value read : a = %d", a);
  return 0;
}
```
Input : This is the value 100
Output : Input value read : a = 100

3. C – DATA TYPES

There are four data types in the C language. They are

Types	Data Types
Basic data types	int, char, float, double
Enumeration data type	enum
Derived data type	Pointer, array, structure, union
Void data type	void

Table 1.2: Data types in C

3.1. Integer type

Integers are used to store whole numbers.

Type	Size(bytes)	Range
int or signed int	2	-32,768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
unsigned short int	1	0 to 255
long int or signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

Table 1.3: Range of integer type

3.2. Floating point type

Floating type is used for storing real numbers.

Size and range of the 16-bit machine floating type.

Type	Size(bytes)	Range
Float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

Table 1.4: Range of floating type

3.3. Character type

Character type is used to store the value of the character.

Size and range of character type on 16-bit machine

Type	Size(bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

Table 1.5: Range of char type

3.4. Void type

Void type means no value. This is usually used to specify the type of functions which returns nothing. We will get acquainted with this datatype as we start learning more advanced topics in C language, like functions, pointers etc.

In C programming, data types are declared for variables. This determines the type and size of data associated with variables.

For example:

int myVar;

Here, myVar is a variable of int (integer) type. The size of int is 4 bytes.

int

Integers are whole numbers that can have both zero, positive and negative values but no decimal values. For example, 0, -5, 10

We can use int for declaring an integer variable.

int id;

Here, id is a variable of type integer.

You can declare multiple variables at once in C programming. For example,

int id, age;

The size of int is usually 4 bytes (32 bits). And, it can take 2^{32} distinct states from -2147483648 to 2147483647.

float and double

float and double are used to hold real numbers.

float salary;

double price;

In C, floating-point numbers can also be represented in exponential. For example,

float normalization factor = 22.442e2;

What's the difference between float and double?

The size of float is 4 bytes. And the size of double (double precision float data type) is 8 bytes.

char

Keyword char is used for declaring character type variables. For example,

char test = 'h';

The size of the character variable is 1 byte.

void

void is an incomplete type. It means "nothing" or "no type". You can think of void as **absent**.

For example, if a function is not returning anything, its return type should be void.

Note that, you cannot create variables of void type.

short and long

If you need to use a large number, you can use a type specifier long. Here's how:

long a;

long long b;

long double c;

Here variables a and b can store integer values. And c can store a floating-point number.

If you are sure, only a small integer ([−32,767, +32,767] range) will be used, you can use short.

short d;

You can always check the size of a variable using the sizeof() operator.

```
#include <stdio.h>
int main() {
  short a;
  long b;
  long long c;
  long double d;
  printf("size of short = %d bytes\n", sizeof(a));
  printf("size of long = %d bytes\n", sizeof(b));
  printf("size of long long = %d bytes\n", sizeof(c));
  printf("size of long double= %d bytes\n", sizeof(d));
  return 0;
}
```

signed and unsigned

In C, signed and unsigned are type modifiers. You can alter the data storage of a data type by using them. For example,

unsigned int x;

int y;

Here, only zero and positive values can be found in variable x because the unsigned modifier has been used.

Considering the size of int is 4 bytes, variable y can hold values from -2^{31} to $2^{31}-1$, whereas variable x can hold values from 0 to $2^{32}-1$.

Other data types defined in C programming are:

- bool Type
- Enumerated type
- Complex types

Derived Data Types

Data types derived from basic data types are derived. For instance: arrays, points, types of functions, structures, etc.

```
#include <stdio.h>
int main()
{
   int testInteger = 5;
   printf("Number = %d", testInteger);
   return 0;
}
```

Output

Number = 5

We use %d format specifier to print int types. Here, the %d inside the quotations will be replaced by the value of testInteger.

```
#include <stdio.h>
int main()
{
   float number1 = 13.5;
   double number2 = 12.4;

   printf("number1 = %f\n", number1);
   printf("number2 = %lf", number2);
   return 0;
}
```

Output

number 1 = 13.500000

number2 = 12.400000

To print float, we use %f format specifier. Similarly, we use %lf to print double values.

```
#include <stdio.h>
int main()
{
   char chr = 'a';
   printf("character = %c", chr);
   return 0;
}
```

Output

character = a

4. C TOKENS

The smallest individual units are known as C tokens. C has six types of tokens: Keywords, Identifiers, Constants, Operators, Delimiters / Separators and Special symbols.

4.1. Keywords: All keywords are mainly character sequences which have a fixed meaning or more.

- Keywords are predefined words in a C compiler.
- Each keyword is meant to perform a specific function in a C program.
- since keywords are referred names for the compiler, they can't be used as variable names.
- All C keywords must be written in lowercase letters.

4.2. Identifiers: C identifier is a name for the identification of a user-defined variable, function, or other object.

RULES FOR CONSTRUCTING IDENTIFIER NAME IN C:

- First character should be an alphabet or underscore.
- Succeeding characters might be digits or letters.
- Punctuation and special characters aren't allowed except underscore.
- Identifiers should not be keywords.

4.3. Constants: Fixed values that do not change during the execution of a C program.

Example: 100 is integer constant, 'a' is character constant, etc.

4.4. Operators: O Operator is a symbol telling a computer to perform certain logical or mathematical manipulations.

Example: Arithmetic operators (+, -, *, /), Logical operators, Bitwise operators, etc.

4.5. Delimiters / Separators: The constants, variables and statements are separated.

Example: comma, semicolon, apostrophes, double quotes, blank space etc.

4.6. Strings: String constants are specified in double quotes.

Example: "gateexam" is string constant

Example of token-

```
int main()
{
  int x, y, total;
  x = 10, y = 20;
  total = x + y;
  printf ("Total = %d \n", total);
}
```

where,

- main – identifier
- {,}, (,) – delimiter
- int – keyword
- x, y, total – identifier
- main, {, }, (,), int, x, y, total – tokens

IDENTIFIERS IN C LANGUAGE:

- Each program element in a C program is given a name called identifiers.
- Names given to identify Variables, functions and arrays are examples for identifiers.
  eg. x is a name given to the integer variable in the above program.

RULES FOR CONSTRUCTING IDENTIFIER NAME IN C:

1. The alphabet or underscore should be the first character.
2. Digits or letters may be the following characters.
3. Punctuation and special characters aren't allowed except underscore.
4. Identifiers should not be keywords.

KEYWORDS IN C LANGUAGE:

- Keywords are predefined words in a C-Compiler.
- A specific function in a C-Program must be done in each keyword.
- Keywords are compiler referred names, so variable names cannot be used.

5. TYPES OF OPERATORS

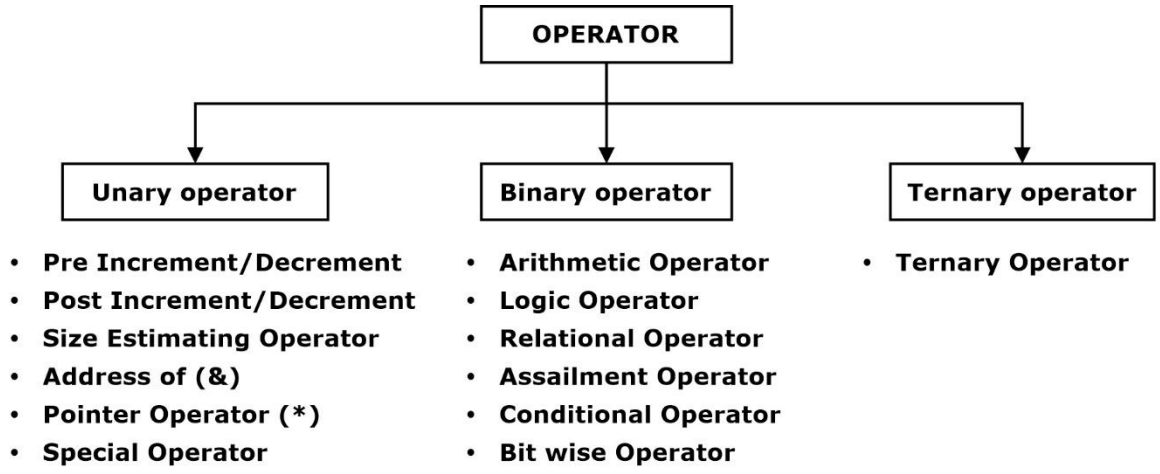


Figure 1.1: Types of operators in C

Operators	Symbols
Arithmetic operators	+, -, *, /, %, ++, --
Assignment operator	=, +=, -=, *=, etc
Relational operators	<, <=, >, >=, !=, ==
Logical operators	&&, ||, !
Bitwise operators	&, |, ~, ^, <<, >>
Special operators	sizeof(), comma, ☐
Pointer operators	* - Value at address (indirection), & - Address Operator

Table 1.6: Operators and symbols

5.1. Post Increment & Pre-Increment

Example 1: Arithmetic Operators

```
// Working of arithmetic operators
#include <stdio.h>
int main ()
{
   int a = 9, b = 4, c;

   c = a+b;
   printf("a+b = %d \n",c);
   c = a-b;
   printf("a-b = %d \n",c);
   c = a*b;
   printf("a*b = %d \n",c);
   c = a/b;
   printf("a/b = %d \n",c);
   c = a%b;
   printf("Remainder when a divided by b = %d \n",c);

   return 0;
}
```

Output

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

The operators +, - and * computes addition, subtraction, and multiplication respectively as you might have expected.

In normal calculation, 9/4 = 2.25. However, the output is 2 in the program.

It is because both the variables a and b are integers. Hence, the output is also an integer. The compiler neglects the term after the decimal point and shows answer 2 instead of 2.25.

The modulo operator % computes the remainder. When a=9 is divided by b=4, the remainder is 1. The % operator can only be used with integers.

Suppose a = 5.0, b = 2.0, c = 5 and d = 2. Then in C programming,

```
// Either one of the operands is a floating-point number
a/b = 2.5
a/d = 2.5
c/b = 2.5

// Both operands are integers
c/d = 2
```

Unary operators (Increment and Decrement Operators)

C programming has two operators increment ++ and decrement -- to change the value of an operand (constant or variable) by 1.

Increment ++ increases the value by 1 whereas decrement -- decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

Example 2: Increment and Decrement Operators

```
// Working of increment and decrement operators
#include <stdio.h>
int main()
{
   int a = 10, b = 100;
   float c = 10.5, d = 100.5;

   printf("++a = %d \n", ++a);
   printf("--b = %d \n", --b);
   printf("++c = %f \n", ++c);
   printf("--d = %f \n", --d);

   return 0;
}
```

Output

```
++a = 11
--b = 99
++c = 11.500000
--d = 99.500000
```

Here, the operators ++ and -- are used as prefixes. These two operators can also be used as postfixes like a++ and a--

Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Table 1.7: Types of assignment operators

Example 3: Assignment Operators

```
// Working of assignment operators
#include <stdio.h>
int main()
{
   int a = 5, c;

   c = a;      // c is 5
   printf("c = %d\n", c);
   c += a;     // c is 10
   printf("c = %d\n", c);
   c -= a;     // c is 5
   printf("c = %d\n", c);
   c *= a;     // c is 25
   printf("c = %d\n", c);
   c /= a;     // c is 5
   printf("c = %d\n", c);
   c %= a;     // c = 0
   printf("c = %d\n", c);

   return 0;
}
```

Output

c = 5

c = 10

c = 5

c = 25

c = 5

c = 0

Relational Operators

The relationship between two operands is checked by a relation operator. If the relationship is true, the value returns 1; if the relationship is false, the value returns 0.

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 is evaluated to 0
>	Greater than	5 > 3 is evaluated to 1
<	Less than	5 < 3 is evaluated to 0
!=	Not equal to	5 != 3 is evaluated to 1
>=	Greater than or equal to	5 >= 3 is evaluated to 1
<=	Less than or equal to	5 <= 3 is evaluated to 0

Table 1.8: Types of relational operators

Example 4: Relational Operators

```
// Working of relational operators
#include <stdio.h>
int main()
{
   int a = 5, b = 5, c = 10;

   printf("%d == %d is %d \n", a, b, a == b);
   printf("%d == %d is %d \n", a, c, a == c);
```

```
    printf("%d > %d is %d \n", a, b, a > b);
    printf("%d > %d is %d \n", a, c, a > c);
    printf("%d < %d is %d \n", a, b, a < b);
    printf("%d < %d is %d \n", a, c, a < c);
    printf("%d != %d is %d \n", a, b, a != b);
    printf("%d != %d is %d \n", a, c, a != c);
    printf("%d >= %d is %d \n", a, b, a >= b);
    printf("%d >= %d is %d \n", a, c, a >= c);
    printf("%d <= %d is %d \n", a, b, a <= b);
    printf("%d <= %d is %d \n", a, c, a <= c);

    return 0;
}
```

Output

```
5 == 5 is 1
5 == 10 is 0
5 > 5 is 0
5 > 10 is 0
5 < 5 is 0
5 < 10 is 1
5 != 5 is 0
5 != 10 is 1
5 >= 5 is 1
5 >= 10 is 0
5 <= 5 is 1
5 <= 10 is 1
```

Logical Operators

An expression containing a logical operator returns either 0 or 1 depending upon whether the expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0.

Table 1.9: Types of logical operators

Example 5: Logical Operators

```
// Working of logical operators

#include <stdio.h>
int main()
{
   int a = 5, b = 5, c = 10, result;

   result = (a == b) && (c > b);
   printf("(a == b) && (c > b) is %d \n", result);

   result = (a == b) && (c < b);
   printf("(a == b) && (c < b) is %d \n", result);

   result = (a == b) || (c < b);
   printf("(a == b) || (c < b) is %d \n", result);

   result = (a != b) || (c < b);
   printf("(a != b) || (c < b) is %d \n", result);

   result = !(a != b);
   printf("!(a != b) is %d \n", result);

   result = !(a == b);
   printf("!(a == b) is %d \n", result);

   return 0;
}
```

Output

(a == b) && (c > b) is 1

(a == b) && (c < b) is 0

(a == b) || (c < b) is 1

(a != b) || (c < b) is 0

!(a != b) is 1

!(a == b) is 0

Explanation of logical operator program

- (a == b) && (c > 5) evaluates to 1 because both operands (a == b) and (c > b) is 1 (true).
- (a == b) && (c < b) evaluates to 0 because operand (c < b) is 0 (false).

- (a == b) || (c < b) evaluates to 1 because (a = b) is 1 (true).
- (a != b) || (c < b) evaluates to 0 because both operand (a != b) and (c < b) are 0 (false).
- !(a != b) evaluates to 1 because operand (a != b) is 0 (false). Hence, !(a != b) is 1 (true).
- !(a == b) evaluates to 0 because (a == b) is 1 (true). Hence, !(a == b) is 0 (false).

Bitwise Operators

Mathematical operations such as: addition, subtraction, multiplication, division, etc., are converted into bits during computation, thus speeding up processing and saving power. In the programming of C, bit-level operators are used to perform operations.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Table 1.10: Types of bitwise operator

Comma Operator

Comma operators are used to link related expressions together. For example:

int a, c = 5, d;

The size of operator

The size of is a unary operator that returns the size of data (constants, variables, array, structure, etc).

Example 6: Size of operator

```
#include <stdio.h>
int main()
{
   int a;
   float b;
   double c;
   char d;
   printf("Size of int=%lu bytes\n",sizeof(a));
   printf("Size of float=%lu bytes\n",sizeof(b));
   printf("Size of double=%lu bytes\n",sizeof(c));
   printf("Size of char=%lu byte\n",sizeof(d));

   return 0;
}
```

Output

Size of int = 4 bytes

Size of float = 4 bytes

Size of double = 8 bytes

Size of char = 1 byte

Ternary operator

The Ternary operators "**? :"** are just like an if-else statement except that because it is an operator we can use it within expressions. **? :** are ternary operators in that it takes three values. They are the only ternary operator in C language.

Syntax→ Boolean Expression? First Statement: Second Statement

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

This conditional statement can be evaluated as follows with an equivalent if else statement.

```
if (x < 0) flag = 0;
  else flag = 1;
```

Example 7: int a=100, b=200,c=300,x;

```
x = (a>b) ? ((a>c)? a : c) : ((b>c) ? b : c);
```

Sol: a>b false so we evaluate (b > c) which is again false.

therefore x = c

Example 8:

Syntax

use the ternary operator:

```
int a = 10, b = 20, c;

c = (a < b) ? a : b;

printf("%d", c);
```

Output of the example above should be:

10

c is set equal to a, because the condition a < b was true.

Remember that the arguments value_if_true and value_if_false must be of the same type, and they must be simple expressions rather than full statements.

Ternary operators can be nested just like if-else statements. Consider the following code:

```
int a = 1, b = 2, ans;
if (a == 1) {
   if (b == 2) {
      ans = 3;
   } else {
      ans = 5;
   }
} else {
   ans = 0;
}
printf ("%d\n", ans);
```

Here's the code above rewritten using a nested ternary operator:

```
int a = 1, b = 2, ans;
ans = (a == 1 ? (b == 2 ? 3 : 5) : 0);
printf ("%d\n", ans);
```

The output of both sets of code above should be:

3

6. OPERATOR PRECEDENCE RELATIONS

Operator precedence relations are given below from highest to lowest order:

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(type){list}	Compound literal(C99)	
2	++ --	Prefix increment and decrement	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(type)	Type cast	
	*	Indirection (dereference)	
	&	Address-of	
	Sizeof	Size-of	
	_Alignof	Alignment requirement(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10	|	Bitwise OR (inclusive or)	
11	&&	Logical AND	
12	||	Logical OR	
13	?:	Ternary conditional	Right-to-Left
14	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= |=	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

Table 1.11: Precedence chart

The operator's priorities decide whether to group terms and how to evaluate an expression. Some operators have higher precedence than others, e.g. the multiplication operator is higher than the additon operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then add into 7.

The highest precedence operators appear on the top of the table, and the lowest is on the bottom of the table. Higher operators are evaluated first within an expression.

Example:-

```
#include <stdio.h>
main() {
   int a = 20;
   int b = 10;
   int c = 15;
   int d = 5;
   int e;
   e = (a + b) * c / d;      // ( 30 * 15 ) / 5
   printf("Value of (a + b) * c / d is : %d\n",  e );
   e = ((a + b) * c) / d;    // (30 * 15 ) / 5
   printf("Value of ((a + b) * c) / d is  : %d\n" ,  e );
   e = (a + b) * (c / d);   // (30) * (15/5)
   printf("Value of (a + b) * (c / d) is  : %d\n",  e );
   e = a + (b * c) / d;     //  20 + (150/5)
   printf("Value of a + (b * c) / d is  : %d\n" ,  e );
   return 0;
}
```

When you compile and execute the above program, it produces the following result –

```
Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is  : 90
Value of (a + b) * (c / d) is  : 90
Value of a + (b * c) / d is  : 50
```

7. C VARIABLE

- C variable is a named memory location for a programme to manipulate the data. The variable value is stored at this location.
- The value can change in the programme of the C variable.
- C variable may be included in any type of data such as int, float, char etc.

7.1. DECLARING & INITIALIZING C VARIABLE:

- Before use in the C programme, variables should be declared.
- Space in memory for a variable is not assigned during declaration. It only occurs on definition of variable.
- Initialization of variables means that the variable is assigned a value.

Type	Syntax
Variable declaration	data_typevariable_name; Example: int x, y, z; char flat, ch;
Variable initialization	data_typevariable_name = value; Example: int x = 50, y = 30; char flag = 'x', ch='l';

Table 1.12: Initialization of a variable

THERE ARE THREE TYPES OF VARIABLES IN C PROGRAM

1. Local variable
2. Global variable
3. Environment variable

7.1.1 LOCAL VARIABLE IN C:

- The scope of local variables is declared only within the function, which are not accessible outside the function.

7.1.2 GLOBAL VARIABLE IN C:

- Global variables are going to be covered in the entire programme. These variables are available throughout the programme from anywhere.
- Defines this variable outside of the main function. The main function and all other subfunctions can also see this variable.

7.1.3 ENVIRONMENT VARIABLES IN C:

- Environment variable is a variable that will be available for all C applications and C programs.
- We can access these variables without declaring and initialising them into an application or a c programme from anywhere in a C programme.
- • Environment features are called the embedded functions for accessing, modifying, and setting the environment variables.

Variable Definition in C

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows –

type variable_list;

Here, **type** must be a valid C data type including char, w_char, int, float, double, bool, or any user-defined object; and **variable_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here –

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

The line **int i, j, k;** declares and defines the variables i, j, and k; which instruct the compiler to create variables named i, j and k of type int.

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows −

type variable_name = value;

Some examples are −

```
extern int d = 3, f = 5;    // declaration of d and f.
int d = 3, f = 5;           // definition and initializing d and f.
byte z = 22;                // definition and initializes z.
char x = 'x';               // the variable x has the value 'x'.
```

For definition without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables are undefined.

Variable Declaration in C

A variable statement provides the compiler with assurance that there is a variable with the given type and name so that the compiler can compile further without the variable being detailed. The compiler requires an actual variable definition when the programme is linked to a variable definition when the compiler is compiled only.

When you use multiple files, a variable declaration is useful, and you define the variable in one of the files that are available when you link the programme. To declare a variable, you use the external keyword. Even if in your C programme you can declare a variable several times, the variable can only be defined once in a file, function or block of code.

Example:

```
#include <stdio.h>
// Variable declaration:
extern int a, b;
extern int c;
extern float f;

int main () {

   /* variable definition: */
   int a, b;
   int c;
   float f;
   /* actual initialization */
   a = 10;
   b = 20;
   c = a + b;
   printf("value of c : %d \n", c);
   f = 70.0/3.0;
   printf("value of f : %f \n", f);
   return 0;
}
```

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

value of c: 30

value of f: 23.333334

The same concept applies on function declaration where you provide a function name at the time of its declaration and its actual definition can be given anywhere else. For

Example :

```
// function declaration
int func();

int main() {

   // function call
   int i = func();
}

// function definition
int func() {
   return 0;
}
```

8. TYPE CONVERSIONS

8.1. Implicit Type Conversion: Some cases occur in which data will be converted automatically from one type to another:

8.1.1. When data is being stored in a variable, if the data being stored does not match the type of the variable.

8.1.2. The stored data is converted to a storage variable type.

8.1.3. When an operation is being performed on data of two different types. The "smaller" data type will be converted to match the "larger" type.

- The following example converts the value of x to a double precision value before performing the division. Note that if the 3.0 were changed to a simple 3, then integer division would be performed, losing any fractional values in the result.
- average = x / 3.0;

8.1.4. When data is passed to or returned from functions.

8.2. Explicit Type Conversion: Data may also be expressly converted, using the typecast operator. The example below converts x into a double-precision value before the division is performed. (y will then be implicitly promoted, following the guidelines listed above.)

8.2.1. average = (double) x / y;

8.2.2. Note that x itself is unaffected by this conversion.

Implicit Type Conversion

Also known as 'automatic type conversion'.

- Done by the own compiler, without externally triggered by the user.
- Generally, takes place when in an expression more than one data type is present. In such conditions type conversion (type promotion) takes place to avoid loss of data.
- All the data types of the variables are upgraded to the data type of the variable with the largest data type.
- **bool -> char -> short int -> int ->**
- **unsigned int -> long -> unsigned ->**
- **long long -> float -> double -> long double**

It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

```
// An example of implicit conversion
#include<stdio.h>
int main()
{
    int x = 10; // integer x
    char y = 'a'; // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

Output-

x = 107, z = 108.000000

Explicit type conversion-

This process is also called type casting and it is user defined. Here the user can type cast the result to make it of a particular data type.

The syntax in C:

(type) expression

Type indicates the data type to which the final result is converted.

```
// C program to demonstrate explicit type casting
#include<stdio.h>

int main()
{
  double x = 1.2;

// Explicit conversion from double to int
  int sum = (int)x + 1;

  printf("sum = %d", sum);

  return 0;
}
```

Output:

sum = 2

Advantages of Type Conversion

- This is done to take advantage of certain features of type hierarchies or type representations.
- It helps us to compute expressions containing variables of different data types.

9. C FLOW CONTROL STATEMENTS

The Control Declaration is one of a programming language instructions, statements or group of statements that determines how other instructions or statements are to be executed. Two flow control styles are provided by C.

a. Branching (conditional) (deciding what action to take)

b. Looping(iterative) (deciding how many times to take a certain action)

c. Unconditional

9.1. If Statement:

It takes a parenthesis expression and a declaration or block of declarations. If the values evaluated are non-zero, expressions shall be assumed to be true.

Syntax of if Statement:

(i)

```
if (condition) statement
```

(ii)

```
if (condition)
{
        statement1
        statement2
        :
}
```

Example of if statement

```
#include <stdio.h>
int main()
{
   int x = 20;
   int y = 22;
   if (x<y)
   {
      printf("Variable x is less than y");
   }
   return 0;
}
```

Output: Variable x is less than y

We can use multiple if statements to check more than one condition.

```
#include <stdio.h>
int main()
{
   int x, y;
   printf("enter the value of x:");
   scanf("%d", &x);
   printf("enter the value of y:");
   scanf("%d", &y);
   if (x>y)
   {
  printf("x is greater than y\n");
   }
   if (x<y)
   {
  printf("x is less than y\n");
   }
   if (x==y)
   {
  printf("x is equal to y\n");
   }
   printf("End of Program");
   return 0;
}
```

In the above example the output depends on the user input.

Output:

enter the value of x:20

enter the value of y:20

x is equal to y

End of Program

9.1.1. If else statement

```
if (condition)
    {statements}
else
    {statements}
```

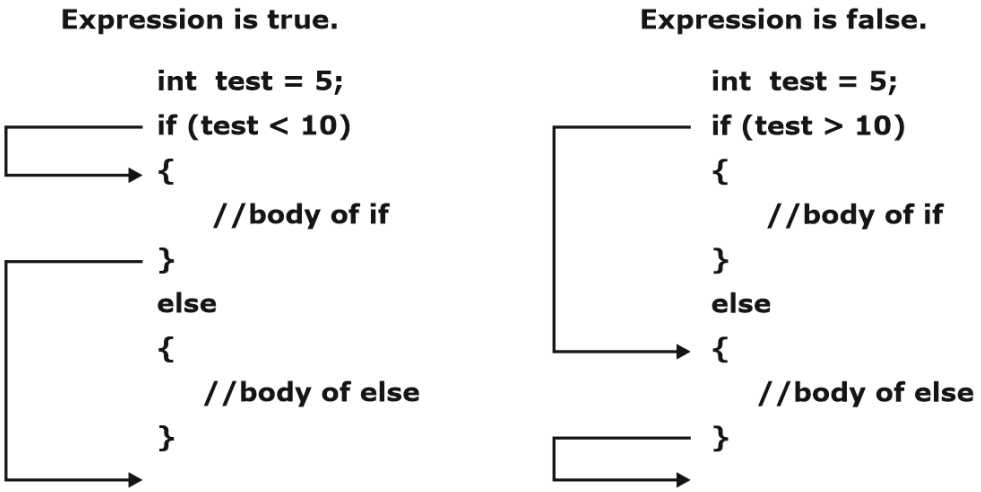
How if...else statement works?

If the test expression is evaluated to true,

- statements inside the body of if are executed.
- statements inside the body of else are skipped from execution.

If the test expression is evaluated to false,

- statements inside the body of else are executed
- statements inside the body of if are skipped from execution.



Example: if...else statement

```
// Check whether an integer is odd or even

#include <stdio.h>
int main() {
   int number;
   printf("Enter an integer: ");
   scanf("%d", &number);
```

```
    // True if the remainder is 0
    if  (number%2 == 0) {
        printf("%d is an even integer.",number);
    }
    else {
        printf("%d is an odd integer.",number);
    }

    return 0;
}
```

Output

Enter an integer: 7

7 is an odd integer.

When the user enters 7, the test expression number%2==0 is evaluated to false. Hence, the statement inside the body of else is execute

9.2. The switch Statement:

The switch declaration will test a variable value (or expression) for a list of cases, and a block of statements related to that case is made when a match is found.

```
 switch (control variable)
    {
     case constant-1: statement(s);
                    break;
         case constant-2: statement(s);
                break;
                        :
         case constant-n: statement(s);
                  break;
                  default: statement(s);
    }
```

Note: Here constant can be either character or integer

Example of Switch Case in C

Let's take a simple example to understand the working of a switch case statement in a C program.

```
#include <stdio.h>
int main()
{
    int num=2;
    switch(num+2)
```

```
    {
        case 1:
          printf("Case1: Value is: %d", num);
        case 2:
          printf("Case1: Value is: %d", num);
        case 3:
          printf("Case1: Value is: %d", num);
        default:
          printf("Default: Value is: %d", num);
    }
    return 0;
}
```

Output:

Default: value is: 2

Explanation: In switch I gave an expression, you can give variable also. I gave num+2, where num value is 2 and after addition the expression resulted in 4. Since there is no case defined with value 4 the default case is executed.

9.3. Loop Control Structure :

Loops offer a way for commands and control to be repeated. This involves repeating a certain portion of the programme until a particular condition is fulfilled by a certain number of times.

9.3.1. while Loop:

The syntax of the while loop is:

```
while (testExpression)
{
   // statements inside the body of the loop
}
```

- Variable initialization. (e.g int x = 0;)
- Condition (e.g while(x <= 10))
- Variable increment or decrement (x++ or x-- or x = x + 2)

(a) Entry Controlled Loop

(b) A while loop in C programming repeatedly executes a target statement as long as a given condition is true.

Example:

While loop with Break:

```
#include <stdio.h>
 int main()
{    int num = 5;
        while (num> 0)
```

```
{   if (num == 3)
break;
printf("%d\n", num);
num--;
                }
          }
```

Output: 5 4

9.3.2. do while Loop:

The syntax of the do...while loop is:

```
do
{
  // statements inside the body of the loop
}
while (testExpression);
```

(a) Exit Controlled Loop

(b) The do while loops runs at least once in any condition(even if the loops control condition is false)

9.3.3. for Loop:

The syntax of the for loop is:

```
for (initializationStatement; testExpression; updateStatement)
{
   // statements inside the body of loop
}
```

Example:

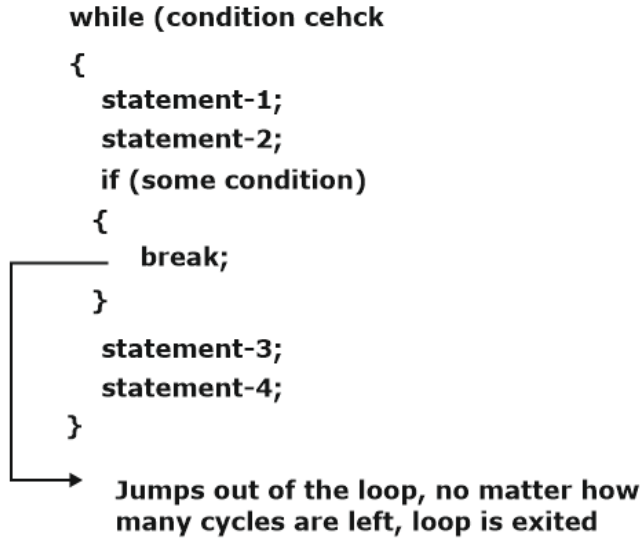
```
#include <stdio.h>
int main()
{
   int num=10, count, sum = 0;
printf("Enter a positive integer: ");
scanf("%d", &num);
   // for loop terminates when num is less than count
for(count = 1; count <= num; ++count)
   {
      sum += count;
   }

printf("Sum = %d", sum);
   return 0;
}
```

Output: 55

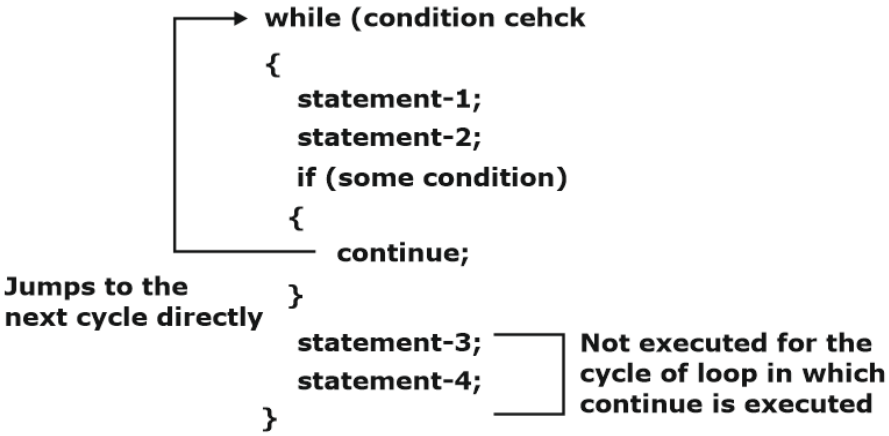
9.4. Unconditional Flow Control Statements

(break, continue, goto, return)

9.4.1. The break Statement: The break statement is used to jump out of a loop instantly, without waiting to get back to the conditional test.



9.4.2. The continue Statement: The 'continue' statement is used to take the control to the beginning of the loop, by passing the statement inside the loop, which has not yet been executed.



9.4.3. Goto statement

C supports an unconditional control statement, goto, to transfer the controller into a C programme from one point to another.

Below is the syntax for goto statement in C.

```
{
        …….
        go to label;
        …….
        …….
        LABEL:
statements;
```

****