Last Moment Tuitions

C Programming

Identifiers, Keywords & Variables

Prepare for CDAC CCAT Exam with LMT Study Materials

Get Complete Study Notes, Practice Problems and Previous Year Question Papers, Syllabus and More

2. Getting Started

★ Identifiers

Identifiers are used for the naming of variables, functions, and arrays. It is a string of alphanumeric characters that begins with an alphabet, or an underscore(_) that are used for variables, functions, arrays, structures, unions, and so on. It is also known as the user-defined word. Identifier names must differ in spelling and case from any keywords. We cannot use keywords as identifiers; they are reserved for special use. Once an identifier is declared, we can use the identifier anywhere in the program to refer to the associated value.

➢ Keyword's in C

- ✔ Keyword is a reserved word.
- ✔ It has a predefined meaning in c language.
- ✔ It cannot be used as a variable name, constant name etc.
- ✔ There are only 32 keywords in C language.

auto	break	case	char	const	continue	default	do
double	else	enum	float	for	goto	if	extern
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

➢ Basic Data Types

Like before learning any language first we learn its alphabets then words then sentence and finally paragraphs. So, before learning C language we have to learn its alphabets.

1. **Character (char)**
   - ✔ Size is 1 bytes.
   - ✔ Range: -128 to 127
   - ✔ Example: a,b,c,d,e,f…………z,A,B,C,D,E etc
2. **Integer (int)**

✔ Size is 2 bytes.

✔ Range: −32,768 to 32,767

✔ Example: 1,2,-1,5,-100 etc.

3. **Float(float)**

✔ Size is 4 bytes.

✔ This have both integer as well as decimal part.

✔ Precision: 6 Decimal places.

✔ Example: 12.345 etc.

4. **Double(double)**

✔Size is 8 bytes.

✔This have both integer as well as decimal part.

✔Precision: 15 Decimal places.

✔Example: 12.3567863 etc.

5. **long Double (long double)**

✔Size is 10 bytes.

✔This have both integer as well as decimal part.

✔Precision: 19 Decimal places.

✔Example: 12.356786398765432678 etc.

★ Variables

In programming, a variable is a container (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location. For example:

int playerScore = 95;

Here, playerScore is a variable of int type. Here, the variable is assigned an integer value 95. The value of a variable can be changed, hence the name variable.

```
char ch = 'a';
// some code
ch = 'l';
```

Rules for naming a variable

1. A variable name can only have letters (both uppercase and lowercase letters), digits and underscore.
2. The first letter of a variable should be either a letter or an underscore.
3. There is no rule on how long a variable name (identifier) can be. However, you may run into problems in some compilers if the variable name is longer than 31 characters.

C is a strongly typed language. This means that the variable type cannot be changed once it is declared. For example:

int number = 5; // integer variable

number = 5.5; // error

double number; // error

Here, the type of number variable is int. You cannot assign a floating-point (decimal) value 5.5 to this variable. Also, you cannot redefine the data type of the variable to double. By the way, to store the decimal values in C, you need to declare its type to either double or float.

Literals:

Literals are data used for representing fixed values. They can be used directly in the code. For example: 1, 2.5, 'c' etc.
Here, 1, 2.5 and 'c' are literals. Why? You cannot assign different values to these terms.

1. Integers

An integer is a numeric literal(associated with numbers) without any fractional or exponential part. There are three types of integer literals in C programming:
- decimal (base 10)
- octal (base 8)
- hexadecimal (base 16)

For example:

Decimal: 0, -9, 22 etc

Octal: 021, 077, 033 etc

Hexadecimal: 0x7f, 0x2a, 0x521 etc

In C programming, octal starts with a 0, and hexadecimal starts with a 0x.

2. Floating-point Literals

A floating-point literal is a numeric literal that has either a fractional form or an exponent form. **For example:**
-2.0
0.0000234
-0.22E-5

3. Characters

A character literal is created by enclosing a single character inside single quotation marks. For example: 'a', 'm', 'F', '2', '}' etc.

4. Escape Sequences

Sometimes, it is necessary to use characters that cannot be typed or has special meaning in C programming. For example: newline(enter), tab, question mark etc. In order to use these characters, escape sequences are used.

Escape Sequences	Character
\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab

\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character

For example: \n is used for a newline. The backslash \ causes escape from the normal way the characters are handled by the compiler.

5. String Literals

A string literal is a sequence of characters enclosed in double-quote marks. For example:

"good" **//string constant**

"" **//null string constant**

"   " **//string constant of six white space**

"x" **//string constant having a single character.**

"Earth is round\n" **//prints string with a newline**

Constants

If you want to define a variable whose value cannot be changed, you can use the const keyword. This will

create a constant. For example,

const double PI = 3.14;

Notice, we have added keyword const.

Here, PI is a symbolic constant; its value cannot be changed.

const double PI = 3.14;

PI = 2.9; //Error

➢ First C Program

Here we will try to understand each and every line which is used in the below program.

```
  #include<stdio.h>
  #include<conio.h>
 void main ()
 {
  printf("Learning c......");
 // this is a comment .
          getch();
 }
```

1) **#include<stdio.h>:-** This line includes all the standard input output functions in our program. printf () will not work is this header file not included.
2) **void main ():-** A program always starts its execution from main() function or we can say main() function is the entry point to the program.
3) **printf ():-**This function is used to print whatever written in double quotes inside the opening and closing braces on console (output screen).
4) **getch ():-** This function is present inside the *conio* header file. Its use is to take a character from user, it is written in end to observe the output as this function will hold the screen until user enters an character from the key board.
5) **comment:-** '//' is a single line comment, whatever written after '//' doesn't get executed.(basically used to explain functioning of program)

★ 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

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 −

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

Example Program:-

```
// 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;
}
```

Relational Operators

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

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.

Example Program:-

```
// 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;
}
```

Logical Operators

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

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 the Logical NOT operator will make it false.	!(A && B) is true.

Example Program:-

```
// 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;

    }
```

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

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then −

Operator	**Description**	**Example**
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operand's value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Example Program:-

```
#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;

}
```

Assignment Operators

The following table lists the assignment operators supported by the C language −

Operator	Description	Example
=	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

%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Misc Operators ↦ sizeof & ternary

Besides the operators discussed above, there are a few other important operators including sizeof and ? : supported by the C Language.

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

Example Program:-

```
// 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;
}
```

Operators Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition 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 adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right

Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right