⇧

C Expressions

An expression is a formula in which operands are linked to each other by the use of operators to compute a value. An operand can be a function reference, a variable, an array element or a constant.

Let's see an example:

1. a-b;

In the above expression, minus character (-) is an operator, and a, and b are the two operands.

There are four types of expressions exist in C:

- Arithmetic expressions
- Relational expressions
- Logical expressions
- Conditional expressions

Each type of expression takes certain types of operands and uses a specific set of operators. Evaluation of a particular expression produces a specific value.

For example:

1. x = 9/2 + a-b;

   The entire above line is a statement, not an expression. The portion after the equal is an expression.

Arithmetic Expressions

An arithmetic expression is an expression that consists of operands and arithmetic operators. An arithmetic expression computes a value of type int, float or double.

When an expression contains only integral operands, then it is known as pure integer expression when it contains only real operands, it is known as pure real expression, and when it contains both integral and real operands, it is known as mixed mode expression.

Evaluation of Arithmetic Expressions

The expressions are evaluated by performing one operation at a time. The precedence and associativity of operators decide the order of the evaluation of individual operations.

When individual operations are performed, the following cases can be happened:

- When both the operands are of type integer, then arithmetic will be performed, and the

result of the operation would be an integer value. For example, 3/2 will yield 1 not 1.5 as the fractional part is ignored.

- When both the operands are of type float, then arithmetic will be performed, and the result of the operation would be a real value. For example, 2.0/2.0 will yield 1.0, not 1.
- If one operand is of type integer and another operand is of type real, then the mixed arithmetic will be performed. In this case, the first operand is converted into a real operand, and then arithmetic is performed to produce the real value. For example, 6/2.0 will yield 3.0 as the first value of 6 is converted into 6.0 and then arithmetic is performed to produce 3.0.

Let's understand through an example.

6*2/ (2+1 * 2/3 + 6) + 8 * (8/4)

Evaluation of expression	Description of each operation
6*2/(2+1 * 2/3 +6) +8 * (8/4)	An expression is given.
6*2/(2+2/3 + 6) + 8 * (8/4)	2 is multiplied by 1, giving value 2.
6*2/(2+0+6) + 8 * (8/4)	2 is divided by 3, giving value 0.
6*2/ 8+ 8 * (8/4)	2 is added to 6, giving value 8.
6*2/8 + 8 * 2	8 is divided by 4, giving value 2.
12/8 +8 * 2	6 is multiplied by 2, giving value 12.

1 + 8 * 2	12 is divided by 8, giving value 1.
1 + 16	8 is multiplied by 2, giving value 16.
17	1 is added to 16, giving value 17.

Relational Expressions

- A relational expression is an expression used to compare two operands.
- It is a condition which is used to decide whether the action should be taken or not.
- In relational expressions, a numeric value cannot be compared with the string value.
- The result of the relational expression can be either zero or non-zero value. Here, the zero value is equivalent to a false and non-zero value is equivalent to true.

Relational Expression	Description
x%2 = = 0	This condition is used to check whether the x is an even number or not. The relational expression results in value 1 if x is an even number otherwise results in value 0.
a!=b	It is used to check whether a is not equal to b. This relational expression results in 1 if a is not equal to b otherwise 0.
a + b = = x + y	It is used to check whether the expression "a + b" is equal to the expression

	"x+y".
a>=9	It is used to check whether the value of a is greater than or equal to 9.

Let's see a simple example:

```
#include <stdio.h>
 int main()
{

    int x=4;
    if(x%2==0)
    {
        printf("The number x is even");
    }
    else
    printf("The number x is not even");
    return 0;
}
```

Output

The number is even

Logical Expressions

- A logical expression is an expression that computes either a zero or non-zero value.
- It is a complex test condition to take a decision.

Let's see some example of the logical expressions.

Logical Expressions	Description
(x > 4) && (x < 6)	It is a test condition to check whether the x is greater than 4 and x is less than 6. The result of the condition is true only when both the conditions are true.
x > 10 || y <11	It is a test condition used to check whether x is greater than 10 or y is less than 11. The result of the test condition is true if either of the conditions holds true value.
! (x > 10) && (y = = 2)	It is a test condition used to check whether x is not greater than 10 and y is equal to 2. The result of the condition is true if both the conditions are true.

Let's see a simple program of "&&" operator.

```
#include <stdio.h>
```

```
int main()
{
   int x = 4;
   int y = 10;
   if ( (x <10) && (y>5))
   {
      printf("Condition is true");
   }
   else
   printf("Condition is false");
   return 0;
}
```

Output

Condition is true

Let's see a simple example of "| |" operator

```
#include <stdio.h>
int main()
{
   int x = 4;
   int y = 9;
   if ( (x <6) || (y>10))
```

```
    {
       printf("Condition is true");
    }
    else
    printf("Condition is false");
    return 0;
}
```

Output

Conditional Expressions

- A conditional expression is an expression that returns 1 if the condition is true otherwise 0.
- A conditional operator is also known as a ternary operator.

The Syntax of Conditional operator

Suppose exp1, exp2 and exp3 are three expressions.

exp1 ? exp2 : exp3

The above expression is a conditional expression which is evaluated on the basis of the value of the exp1 expression. If the condition of the expression exp1 holds true, then the final conditional expression is represented by exp2 otherwise represented by exp3.

Let's understand through a simple example.

```
#include<stdio.h>
#include<string.h>
int main()
{
  int age = 25;
   char status;
   status = (age>22) ? 'M': 'U';
   if(status == 'M')
   printf("Married");
   else
   printf("Unmarried");
   return 0;
}
```

Output **Married**

Data Segments

To understand the way our C program works, we need to understand the arrangement of the memory assigned to our program.

All the variables, functions, and data structures are allocated memory into a special memory segment

known as Data Segment. The data segment is mainly divided into four different parts which are specifically allocated to different types of data defined in our C program.

The parts of Data segments are :

1. Data Area

It is the permanent memory area. All static and external variables are stored in the data area. The variables which are stored in the data area exist until the program exits.

2. Code Area

It is the memory area which can only be accessed by the function pointers. The size of the code area is fixed.

3. Heap Area

As we know that C supports dynamic memory allocation. C provides the functions like malloc() and calloc() which are used to allocate the memory dynamically. Therefore, the heap area is used to store the data structures which are created by using dynamic memory allocation. The size of the heap area is variable and depends upon the free space in the memory.

4. Stack Area

Stack area is divided into two parts namely: initialize and non-initialize. Initialize variables are given priority than non-initialize variables.

1. All the automatic variables get memory into stack area.
2. Constants in c get stored in the stack area.

3. All the local variables of the default storage class get stored in the stack area.
4. Function parameters and return value get stored in the stack area.
5. Stack area is the temporary memory area as the variables stored in the stack area are deleted whenever the program reaches out of scope.