**WEEK-END ASSIGNMENT-01**

# Operating Systems Workshop (CSE 3541)

**Problem Statement:**

Experiment with C operators, role of operator precedence, associativity and expressions.

# Assignment Objectives:

To become familiar with C operators, expression evaluation as per operator precedence and associativity rule.

# Instruction to Students (If any):

**Students are required to write his/her own program by avoiding any kind of copy from any sources. Additionally, They must be able to realise the coutcome of that question in relevant to systems pro- gramming.**

# Programming/ Output Based Questions:

1. Evaluate the arithmetic expression *a b/c d*, where the floating-point variables a , b , c and d have been assigned the values 1.0, 2.0, 3.0 and 4.0. Create a C program to display the value of the expression on standard output device.

— ∗

**Write/paste your code here** ▼

**Output** ▼

#include <stdio.h>

int main() {

double a = 1.0;

double b = 2.0;

double c = 3.0;

double d = 4.0;

double result = a \* b / c \* d;

printf("The value of the expression is: %lf\n", result);

return 0;

}

Result of the expression is : -1.67

1. Which of the following identifiers are (a) C reserved words, (b) standard identifiers, (c) conventionally used as constant macro names, (d) other valid identifiers, and (e) invalid identifiers?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **void** | **MAX\_ENTRIES** | **return** | **printf** | **"char"** |
| **xyz123** | **time** | **part#2** | **G** | **Sue’s** |

## #insert this\_is\_a\_long\_one double hello\_

**Answer here** ▼

(e)Invalid Identifiers:

#insert

Double

"char"

(d) Other Valid Identifiers:

xyz123

time

part#2

G

Sue's

this\_is\_a\_long\_one

\_hello

(c) Conventionally Used as Constant Macro Names:

None of the identifiers you provided are conventionally used as constant macro names. In C, macro names are typically written in all uppercase letters.

(b) Standard Identifiers:

None of the identifiers you provided are standard identifiers in C.

1. C Reserved Words:

None

1. The Pythagorean theorem states that the sum of the squares of the sides of a right triangle is equal to the square of the hypotenuse. For example, if two sides of a right triangle have lengths of 3 and 4, then the hypotenuse must have a length of 5. Together the integers 3, 4, and 5 form a *Pythagorean triple*. There are an infinite number of such triples. Given two positive integers, *m* and *n*, where *m > n*, a Pythagorean triple can be generated by the following formulas:

*side*1 = *m*2 − *n*2

*side*2 = 2*mn hypotenuse* = *m*2 + *n*2

The triple ( *side*1 = 3, *side*2 = 4, *hypotenuse* = 5) is generated by this formula when *m* = 2 and *n* = 1. Write a program that takes values for *m* and *n* as input and displays the values of the Pythagorean triple generated by the formulas above. The values of *m* and *n* should be provided from an input file through input redirection.

**Write/paste your code here** ▼

**Output** ▼

1

2

3

Pythagorean Triple: (-8, 6, 10)

#include <stdio.h>

int main() {

int m, n;

// Read values for m and n from input file through redirection

scanf("%d %d", &m, &n);

// Calculate the sides and hypotenuse

int side1 = m \* m - n \* n;

int side2 = 2 \* m \* n;

int hypotenuse = m \* m + n \* n;

// Display the Pythagorean triple

printf("Pythagorean Triple: (%d, %d, %d)\n", side1, side2, hypotenuse);

return 0;

}

1. Write C statements to carry out the following steps.
   1. If **item** is nonzero, then multiply **product** by **item** and save the result in **product** ; other- wise, skip the multiplication. In either case, print the value of **product**.
   2. Store the absolute difference of **x** and **y** in **y** , where the absolute difference is **( x - y )** or

**(y - x )**, whichever is positive. Do not use the **abs** or **fabs** function in your solution.

* 1. If **x** is 0 , add 1 to **zero count**. If **x** is negative, add **x** to **minus sum**. If **x** is greater than 0 , add **x** to **plus sum**.

#include <stdio.h>

int main() {

int zeroCount = 0;

int minusSum = 0;

int plusSum = 0;

int x;

while (1) {

printf("Enter a value for x (or enter 0 to exit): ");

scanf("%d", &x);

if (x == 0) {

break;

} else if (x < 0) {

minusSum += x;

} else {

plusSum += x;

}

if (x == 0) {

zeroCount++;

}

}

printf("Zero Count: %d\n", zeroCount);

printf("Minus Sum: %d\n", minusSum);

printf("Plus Sum: %d\n", plusSum);

return 0;

}

#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) {

y = x - y;

} else {

y = y - x;

}

printf("Absolute Difference: %d\n", y);

return 0;

}

#include <stdio.h>

int main() {

int product = 1; // Initialize product to 1 or any initial value

int item; // Assuming item is of integer type

printf("Enter the value of item: ");

scanf("%d", &item);

if (item != 0) {

product \*= item;

}

printf("Product: %d\n", product);

return 0;

}

1. Consider the C arithmetic expression 2 ((*i*%5) (4 + (*j* 3)*/*(*k* + 2))) where *i* , *j* and *k* are integer variables. If these variables are assigned the values 8, 15 and 4, respectively, then the given determine the value of the expression. (**Note:** *The interpretation of the remainder operation (%) is unclear when one of the operands is negative. Most versions of C assign the sign of the first operand to the remainder. The % operation is undefined when second operand is zero.*)

∗ ∗ −

**Expression evaluation** ▼

when i is 8, j is 15, and k is 4, the value of the expression is 36.

1. Consider the following C expressions;

**Answer**

(a)

**Cause**

(b)

* 1. Suppose that *i* is an integer variable whose value is 7, and *f* is a floating-point variable whose value is 8.5. The expression (*i* + *f* )%4 is valid or invalid.

(i + f) % 4 is an invalid expression in C.

* 1. Suppose that *i* is an integer variable whose value is 7, and *f* is a floating-point variable whose value is 8.5. The expression ((*int*)(*i* + *f* ))%4 is valid or invalid.

((int)(i + f)) % 4 is a valid expression in C

1. ASCII code for the character ? is 63. Characters are represented by integer codes, C permits con- version of type **char** to type **int** and vice versa. So find the output for the given code snippet;

**Output**

63 ? ?

## int q\_code = (int)’?’;

**printf("%d %c %d\n", q\_code,’?’,’?’);**

1. The following expressions contain different operands and operators assuming **x=3.0** , **y=4.0** , and

**z=2.0** are type **double** , **flag=0** is type **int**. Write each expressions value.

**Answer**

(a)true

(b)false

(c)true

(d)false

* 1. !*flag*

(b) *x* + *y/z <*= 3*.*5

(c) !*flag*||(*y* + *z >*= *x* − *z*)

(d) !(*flag*||(*y* + *z >*= *x* − *z*))

1. What value is assigned to the type **int** variable **ans** in this statement if the value of **p** is **100** and **q**

is **50**?

**Output**

1(True)

## ans = (p > 95) + (q < 95);

1. Evaluate each of the following expressions if a is 6 , b is 9 , c is 14 , and flag is 1 . Which parts of these expressions are not evaluated due to short-circuit evaluation?

**Answer**

1. Both parts are evaluated.
2. Both parts are evaluated.

(c) The first part is evaluated, and the second part is not evaluated.

(d) Both parts are evaluated.

* 1. c == a + b || !flag

(b) a != 7 && flag || c >= 6

(c) !(b <= 12) && a % 2 == 0

(d) !(a > 5 || c < a + b)

1. Suppose that **i** is an integer variable, **x** is a floating-point variable, **d** is a double-precision variable and **c** is a character-type variable. Find the output generated by these statements that make use of the operator **sizeof**.

**printf("integer:%ld bytes\n", sizeof i); printf("integer:%ld bytes\n", sizeof(i)); printf("float:%ld bytes\n", sizeof x); printf("float:%ld bytes\n", sizeof(x)); printf("double:%ld bytes\n", sizeof d); printf("double:%ld bytes\n", sizeof(d)); printf("character:%ld bytes\n", sizeof c); printf("character:%ld bytes\n", sizeof(c));**

**Answer**

integer:4 bytes

integer:4 bytes

float:4 bytes

float:4 bytes

double:8 bytes

double:8 bytes

character:1 bytes

character:1 bytes

***/\* Same way can be used for other data types to find the size \*/***

1. Another way to generate the same information as like previous question is to use a cast rather than a variable within each printf statement. Find the output generated by these statements that make use of the operator **sizeof**.

**printf("integer:%ld bytes\n", sizeof(int)); printf("float:%ld bytes\n", sizeof(float)); printf("double:%ld bytes\n", sizeof(double)); printf("character:%ld bytes\n", sizeof(char));**

**Answer**

integer:4 bytes float:4 bytes double:8 bytes character:1 bytes

***/\* Same way can be used for other data types to find the size \*/***

1. C supports several assignment operators. The most commonly used assignment operator is =. Assign- ment expressions that make use of this operator are written in the form **identifier = expression**, where i**dentifier** generally represents a variable, and **expression** represents a constant, a vari- able or a more complex expression. Determine the expression values assume that i is an integer-type variable, and that the ASCII character set applies.

**Answer**

1. assigned the value 3
2. assigned the value 3
3. assigned the value 5
4. assigned the value 4
5. assigned the value 3
6. assigned the value -3

**i=(’x’-’o’)/3;**

**i=(’y’-’o’)/3;**

**i=2\*j/2; (say j is an integer and j is 5) i=2\*(j/2);**

**i=3.0;**

**i=-3.5;**

**NOTE:** Multiple assignments of the form

## identifier 1 = identifier 2 = ... = expression

are permissible in C. In such situations, the assignments are carried out from **right to left**.

1. C also contains other form assignment operators: +=, -+, \*=, /=, %= etc., called short hand operators. Suppose that i and j are integer variables whose values are 5 and 7, and f and g are floating-point variables whose values are 5.5 and -3.25. Determine the value of the expressions

**Answer**

10

8.75

49

2.9167

0

## i += 5;

**f -= g;**

## j \*= ( i - 3); f /= 3;

**i %= ( j - 2 )**

1. Suppose that x, y and z are integer variables which have been assigned the values 2, 3 and 4, respec- tively. Determine the value of the given expression;

**Answer**

-9.3334

## x\*=-2\*(y+z)/3;

1. The assignment statement that contains a conditional expression on the right-hand side. Determine the value of flag if **i=-5** and **i=-6** respectively.

**Answer**

0

## flag = ( i < 0 ) ? 0 : 100

1. In the following assignment statement, a, b and c are assumed to be integer variables. If **a**, **b** and **c** have the values 1, 2 and 3, respectively, then determine the value of the expression that includes operators of different precedence groups.

**Answer**

c is 5

## c += (a > 0 && a <= 10) ? ++a : a / b ;

1. Illustrate the purpose of the following code snippet over the inputs a,b and c respectively.

**Answer**

this code snippet takes three integer inputs, finds the maximum value among them, and prints that maximum value to the standard output

## int m1,m2,a,b,c;

**printf("Enter the values of a,b,c:"); scanf("%d%d%d",&a,&b,&c); m1=(a>b)?a:b;**

## m2=(m1>c)?m1:c;

**printf("%d\n",m2);**

1. A C program contains the following declarations and initial assignments:

## int i= 8; int j = 5;

**float x = 0.005; float y = -0.01;**

## char c = ’c’ , d = ’d ’ ;

Determine the value of each of the following expressions. Use the values initially assigned to the variables for each expression.

## (a) ( 3 \* i - 2 \*j ) % ( 2 \* d - c )

**Answer**

A)14

1. True
2. 0.01
3. True
4. False
5. 9

**(b) ( x > y ) && (i > 0) && ( j < 5 ) (c) 2 \* x + ( y = = 0)**

## (d) ( 2 \* x + y ) == 0

**(e) 5 \* (i + j ) > ’ c ’**

## (f) i++

1. Suppose a is an unsigned integer variable (say represented in 16 bits format) whose value is 0x6db7. In the following the expression, we will shift all bits of a six places to the right and assign the resulting bit pattern to the unsigned integer variable b. Find the resulting value of b. Also write the lost bits because of shifting.

**Answer**

the resulting value of b in hexadecimal is 0x36D6. The lost bits due to shifting are 000000.

## b = a >> 6 ;

1. Determine the value of each of the following expressions, assume that a is an unsigned integer variable whose initial value is 0x6db7.

**Answer**

1. 0x6db7
2. 0x10b8
3. 0x7fff
4. 0x2b06
5. 0x7f5c00

## a &= 0x7f

* 1. **a ˆ= 0x7f**

## a |= 0x7f

* 1. **a = a & 0x3f06**

## (e) a = a | 0x3f06 << 8

1. Determine the output of the following code snippet.

## int main(){ int m1,a,b,c;

**Answer**

**(1)a=10 b=20 30 m1= 20**

**(2)a=30 b=10 c=20 m1= (3)a=20 b=30 c=10 m1= 30**

**printf("Enter the values of a,b,c:"); scanf("%d%d%d",&a,&b,&c); m1=a>b?a>c?a:c:b; printf("m1=%d\n",m1);**

30

## return 0;

**}**

1. Evaluate the expressions;

**Answer**

1. **A =16**
2. **B +=30**
3. **x +=20**
4. **p=100**

**q=100**

**r=100**

## Assume A, B, num, xy, f, t, p, q, r are int type variables;

**(1) A=10+(num=2)\*3;**

## (2) B +=(xy \*=3); [here xy=10]

**(3) x +=(f=(t\*=20)); [here x=20, t=10] (4) p=q=r=100;**

1. State the output of the following code snippet;

**Answer**

OSW CSE=11

## int a,b,s; s=scanf("%d%d%d",&a,&b,&a); printf("%d\n",s+printf("OSW CSE="));

1. State the output of the following code snippet;

**Answer**

0 -1 1 2 1

## int i=-1,j=-1,k=0,l=2,m; m=++i || k++ && ++j || l++;

**printf("%d %d %d %d %d\n", i,j,k,l,m);**

1. State the output of the following code snippet;

**Answer**

**16**

## int i=10,j=6; printf("%d\n", i+++j++);

1. State the output of the following code snippet;

**Answer**

**1 1 1 1**

## int i=3>4, j=4>3; int k=(i=j);

**int l=(k==j);**

## printf("%d %d %d %d",i,j,k,l);

1. State the output of the following code snippet;

**Answer**

40 0

## int x=400;

**printf("%d %d\n",x=40,x>=50);**

1. verify the output/ error of the following code snippet;

**Answer**

**1**

## int i=2,j=0; int k=i&&j=1;

**printf("%d\n",k);**

1. Find the output of the following code snippet;

**Answer**

**0**

## int i=2,j=2; int k=iˆj&i;

**printf("%d\n",k);**

1. Find the output of the following code snippet;

**Answer**

**1**

## int i=3,j=2;

**int k=i << 1 > 5; printf("%d\n",k);**