1

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    int i;

    for(i=0;i<5;i++){

         int i=10;

         printf(" %d",i);

         i++;

    }

    return 0;

}

|  |  |  |
| --- | --- | --- |
| (A) | 10 11 12 13 14 |  |
| Correct | 10 10 10 10 10 |  |
| (C) | 0 1 2 3 4 |  |
| (D) | Compilation error |  |
| Explanation: | | | |
| Default storage class of local variable is auto. Scope of auto variables are block in which it has been declared. When program control goes out of the scope auto variables are dead. So variable *i* which has been declared inside for loop has scope within loop and in each iteration variable *i* is dead and re-initialized.  Note: If we have declared two variables of same name but different scope then local variable will have higher priority. | | | |

2

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    register a,b,x;

    scanf("%d %d",&a,&b);

    x=a+~b;

    printf("%d",x);

    return 0;

}

|  |  |  |
| --- | --- | --- |
| (A) | 0 |  |
| Wrong | It will be difference of a and b |  |
| (C) | It will be addition of a and b |  |
| Correct | Compilation error |  |
| Explanation: | | | |
| Register variables are stored in CPU. So it has not memory address. Hence it is incorrect to write &a. | | | |

3

What will be output if you will execute following c code?

#include<stdio.h>

auto int a=5;

int main(){

    int x;

    x=~a+a&a+a<<a;

    printf("%d",x);

    return 0;

}

|  |  |  |
| --- | --- | --- |
| Wrong | 5 |  |
| (B) | 0 |  |
| (C) | 153 |  |
| Correct | Compilation error |  |
| Explanation: | | | |
| We cannot declare auto variable outside of any function since it auto variables gets are created (i.e. gets memory) at run time. | | | |

4

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    register int a,b;

    int c;

    scanf("%d%d",&a,&b);

    c=~a + ~b + ++a + b++;

    printf(" %d",c);

    return 0;

}

//User input is: 1 2

|  |  |  |
| --- | --- | --- |
| Wrong | -1 |  |
| (B) | 0 |  |
| (C) | 1 |  |
| Correct | Compilation error |  |
| Explanation: | | | |
| Register variables are stored in CPU. So it has not memory address. Hence it is incorrect to write &a. | | | |

5

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    int arr[3]={10,20,30};

    int x=0;

    x = ++arr[++x] + ++x + arr[--x];

    printf("%d ",x);

    return 0;

}

|  |  |  |
| --- | --- | --- |
| (A) | 22 |  |
| (B) | 23 |  |
| Correct | 43 |  |
| Wrong | 44 |  |
| Explanation: | | | |
| **In Turbo C 3.0 and 4.5 compilers**  Output: 43  Consider on expression:  = ++arr[**++x**] + ++x + arr[--x] //x = 0 + 1  = ++arr[++x] + **++x** + arr[--x] //x = 1 + 1  = ++arr[++x] + ++x + arr[**--x**] //x = 2 - 1  = ++arr[**1**] + **1** + arr[**1**] //x = 1  = **++**arr[1] + 1 + arr[1]  //arr[1] = 20+1  = arr[1] + 1 + arr[1] //arr[1] = 21  = 21 + 1 + 21  = 43  **In Linux GCC complier**  Output: 44  Consider on expression:  = ++arr[**++x**] + ++x + arr[--x] //x = 0 + 1  = ++arr[1] + **++x** + arr[--x] ////x = 1 + 1  = ++arr[++x] + 2 + arr[**--x**] //x = 2 - 1  = **++**arr[1] + 2 + arr[1] //arr[1] = 20+1  = arr[1] + 1 + arr[1] //arr[1] = 21  = 21 + 2 + 21  = 44 | | | |

6

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    int a[]={10,20,30,40};

    int i=3,x;

    x=1\*a[--i]+2\*a[--i]+3\*a[--i];

    printf("%d",x);

    return 0;

}

|  |  |  |
| --- | --- | --- |
| (A) | 30 |  |
| Correct | 60 |  |
| (C) | 90 |  |
| (D) | Compilation error |  |
| Explanation: | | | |
| **In Turbo C 3.0 and 4.5 compilers**  Output: 60  Consider on expression:  = 1 \* a[**--i**] + 2 \* a[--i] + 3 \* a[--i] //i = 3 - 2  = 1 \* a[--i] + 2 \* a[**--i**] + 3 \* a[--i] //i = 2 - 1  = 1 \* a[--i] + 2 \* a[--i] + 3 \* a[**--i**] //i = 1 - 1  = 1 \* a[**0**] + 2 \* a[**0**] + 3 \* a[**0**] //i = 0  = 1 \* **10** + 2 \* **10** + 3 \* **10** //a[0] = 10  = 10 + 20 + 30  = 60  **In Linux GCC complier**  Output: 90  Consider on expression:  = 1 \* a[**--i**] + 2 \* a[--i] + 3 \* a[--i] //i = 3 - 2  = 1 \* a[**--i**] + 2 \* a[**--i**] + 3 \* a[--i] //i = 2 - 1  = 1 \* a[1] + 2 \* a[1] + 3 \* a[**--i**] //i = 1 - 1  = 1 \* a[1] + 2 \* a[1] + 3 \* a[0]  = 1 \* **20** + 2 \* **20** + 3 \* **10**  = 20 + 40 + 30  = 90 | | | |

7

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    static int a[][2][3]={0,1,2,3,4,5,6,7,8,9,10,11,12};

    int i=-1;

    int d;

    d=a[i++][++i][++i];

    printf("%d",d);

    return 0;

}

|  |  |  |  |
| --- | --- | --- | --- |
| (A) | 9 |  | |
| Correct | 10 |  | |
| (C) | 11 |  | |
| (D) | Compilation error |  | |
| Explanation: | | |
| = a[i++][++i][**++i**] //i = -1 + 1  = a[i++][**++i**][++i] //i = 0 + 1  = a[1][1][1] //i = 1 + 1  = 10 | | |

8

What will be output if you will execute following c code?

#include<stdio.h>

int f(int);

int main(){

    int i=3,val;

    val=sizeof (f(i)+ +f(i=1)+ +f(i-1));

    printf("%d %d",val,i);

    return 0;

}

int f(int num){

        return num\*5;

}

|  |  |  |
| --- | --- | --- |
| Correct | 2 3 |  |
| (B) | 4 3 |  |
| Wrong | 3 2 |  |
| (D) | Compilation error |  |
| Explanation: | | | |
| **Turbo C 3.0 and Turbo C 4.5 compiler:**  2 3  **Linux GCC complier:**  4 3  Any expression inside sizeof operator is never changed the value of the any variable. So value of variable *i* will remain 3. After the evaluation of expression inside sizeof operator we will get an integer value. So value of variable *val* will be sizeof int data type.  Note: Size of into in turbo C 3.0 and 4.5 is two byte while Linux gcc complier is four byte | | | |

9

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    int x,a=3;

    x=+ +a+ + +a+ + +5;

    printf("%d  %d",x,a);

    return 0;

}

|  |  |  |
| --- | --- | --- |
| (A) | 10 3 |  |
| Correct | 11 3 |  |
| Wrong | 10 5 |  |
| (D) | Compilation error |  |
| Explanation: | | | |
| Consider on expression: + +a  Here both + are unary plus operation. So  = + +a+ + +a+ + +5;  = + +3+ + +3+ + 5  = 3+ 3+ 5  = 11  Note: Increment operator ++ cannot have space between two plus symbol. | | | |

10

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    int num,i=0;

    num=-++i+ ++-i;

    printf("%d",num);

    return 0;

}

|  |  |  |
| --- | --- | --- |
| Wrong | 0 |  |
| (B) | 1 |  |
| (C) | -2 |  |
| Correct | Compilation error |  |
| Explanation: | | | |
| After operation of any operator on operand it returns constant value. Here we are performing unary minus operator on variable i so it will return a constant value and we can perform ++ operation on constant. | | | |

11

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    int num,a=5;

    num=-a--+ +++a;

    printf("%d  %d",num,a);

    return 0;

}

|  |  |  |  |
| --- | --- | --- | --- |
| (A) | 1 5 |  | |
| (B) | -1 6 |  | |
| Wrong | 1 6 |  | |
| Correct | 0 5 |  | |
| Explanation: | | |
| = -a--+ +++a  = -a-- + + ++a  = -a-- + + **++**a  = -6 + + 6 //a = 6 -1  = -6 + 6 //a = 5  = 0 | | |

12

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    int num,a=15;

    num=- - - -a--;

    printf("%d  %d",num,a);

    return 0;

}

|  |  |  |  |
| --- | --- | --- | --- |
| Correct | 15 14 |  | |
| (B) | 14 15 |  | |
| Wrong | 14 14 |  | |
| (D) | 15 15 |  | |
| Explanation: | | |
| = - - - -a  = - - - -15 //a = 15 – 1  = 15  //a = 14 | | |

13

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    int x,a=2;

    x=++a,++a,a++;

    printf("%d  %d",x,a);

    return 0;

}

|  |  |  |
| --- | --- | --- |
| (A) | 5 5 |  |
| Correct | 3 5 |  |
| Wrong | 4 5 |  |
| (D) | 5 4 |  |
| Explanation: | | | |
| x = **++**a, ++a, a++  x = 3, ++a, a++ // a = 2 + 1  x **=** 3, ++a, a++ // = operator has higher precedence than comma operator  x = 3, **++**a, a++ // a = 3 + 1  x = 3, 4, a**++**  x = 3, 4, 4 // a = 4 + 1  x = 3 // a = 5 | | | |

14

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    int x,i=2;

    x=~-!++i;

    printf("%d",x);

    return 0;

}

|  |  |  |
| --- | --- | --- |
| (A) | -2 |  |
| Correct | -1 |  |
| (C) | 0 |  |
| Wrong | 1 |  |
| Explanation: | | | |
| = ~-!**++**i  = ~-**!**3 //i = 2 + 1  = ~**-**0 //!3 = 0  = ~0 //-0 = 0  = -(0 + 1) //~ is 1's complement operator.  = -1 | | | |

15

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    static double \*p,\*q,\*r,\*s,t=5.0;

    double \*\*arr[]={&p,&q,&r,&s};

    int i;

    \*p=\*q=\*r=\*s=t;

    for(i=0;i<4;i++)

        printf("%.0f  ",\*\*arr[i]);

    return 0;

}

|  |  |  |
| --- | --- | --- |
| (A) | 5 5 5 5 5 |  |
| (B) | 5 6 7 8 9 |  |
| Wrong | Infinite loop |  |
| Correct | Run time error   |  | | --- | |  | |  |
| Explanation: | | | |
| **Turbo C 3.0:**  5 5 5 5 5  **Turbo C 4.5 and Linux GCC complier:**  Run time error | | | |

16

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    float x;

    x=0.35==3.5/10;

    printf("%f",x);

    return 0;

}

|  |  |  |
| --- | --- | --- |
| Correct | 0.000000 |  |
| (B) | 1.000000 |  |
| Wrong | 0.350000 |  |
| (D) | Compilation error |  |
| Explanation: | | | |
| **Turbo C 3.0 and Turbo C 4.5 compiler:**  Output: 0.000000  3.5/10 is little greater than .35  **Linux GCC compilers:**  Output: 1.000000  Note: == is logic operator. It returns 1 if both operands are equal otherwise it returns 0. | | | |

17

#include<stdio.h>

int main(){

    int arr[]={6,12,18,24};

    int x=0;

    x=arr[1]+(arr[1]=2);

    printf("%d",x);

    return 0;

}

|  |  |  |  |
| --- | --- | --- | --- |
| Correct | 4 |  | |
| (B) | 8 |  | |
| Wrong | 14 |  | |
| (D) | Compilation error |  | |
| Explanation: | | |
| = arr[1] + **(**arr[1] = 2**)** //arr[i] = 2  = arr[1] + arr[1]  = 2 + 2  = 4 | | |

18

What will be output if you will execute following c code?

#include<stdio.h>

int sq(int);

int main(){

    int a=1,x;

    x=sq(++a)+sq(a++)+sq(a++);

    printf("%d",x);

return 0;

}

int sq(int num){

    return num\*num;

}

|  |  |  |
| --- | --- | --- |
| (A) | 15 |  |
| (B) | 16 |  |
| Correct | 17 |  |
| (D) | 18 |  |
| Explanation: | | | |
| = sq(**++a**) + sq(a++) + sq(a++) //a= 1 + 1  = sq(2) + sq(**2**) + sq(a++) //a = 2 + 1  = sq(2) + sq(2) + sq(**3**)  //a = 3 + 1  = 4 + 4 + 9  = 17  Note: Pre-increment fist increment then assign while post increment operator first assign then increment. | | | |

19

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    printf("%c",\*"abcde");

return 0;

}

|  |  |  |
| --- | --- | --- |
| Wrong | acbcd |  |
| (B) | e |  |
| Correct | a |  |
| (D) | NULL |  |
| Explanation: | | | |
| String constant "abcde" will return memory address of first character of the string constant. \*"abcde" will return the first character of string constant. | | | |

20

What will be output if you will execute following c code?

#include<stdio.h>

int main(){

    printf("%d","abcde"-"abcde");

return 0;

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| (A) | 0 | | |  |
| (B) | -1 | | |  |
| (C) | 1 | | |  |
| Correct | Garbage | | |  |
| Explanation: | | | | | |
| Memory address of string constants depends upon operating system. | | | | | |
| Total correct answers: | | 5 |
| Total incorrect answers: | | 15 |
| Total not attempted questions: | | 0 |
| Marks obtained: | | 0 |
| Result: | | Fail |