Test -2 (October 13, 2020)

Total points 23/30

Please fill your ID and name correctly in the space provided.

There are thirty questions in this test, each carrying one mark, and the duration of the test is 30 minutes. Only one of the many options provided is the correct answer in the given questions. Select the correct answer and move to the next question. You are expected to maintain professional honesty while you attempt the answers to the questions.

Do not forget to submit your responses.

The respondent's email address (**f20181119@pilani.bits-pilani.ac.in**) was recorded on submission of this form.

0 of 0 points

ID *

2018A7PS1119P

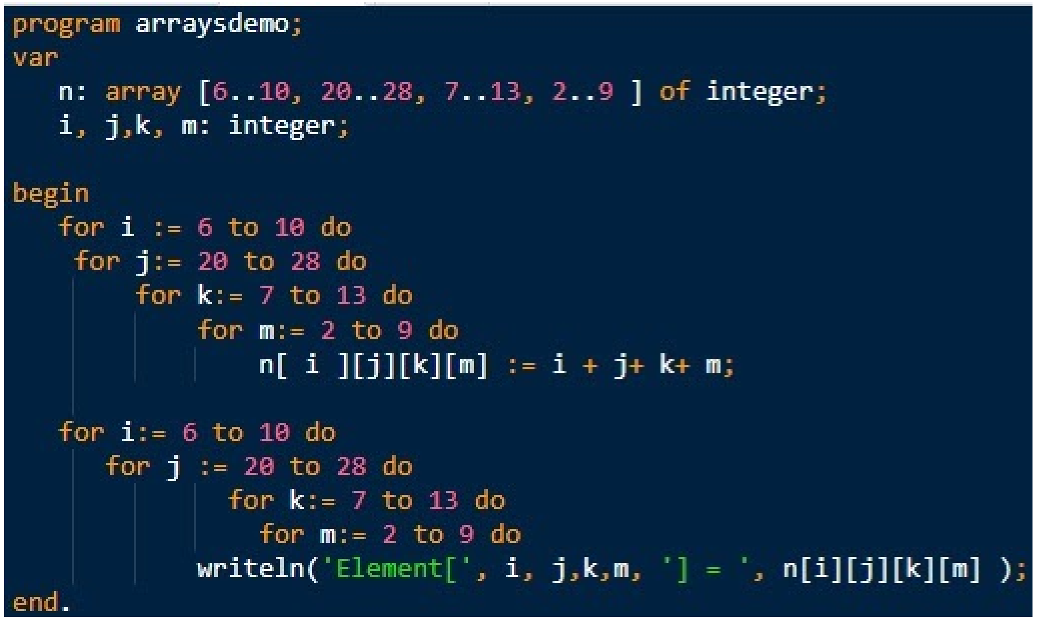
Name *

Shreyas Bhat Kera

Questions 1-30 23 of 30 points

Consider a four dimensional array 'n' of integers declared in a Pascal like language as shown in the following figure. The ranges 6..10, 20..28, 7..13 and 2..9 are defined in four directions R1, R2, R3 and R4 respectively. The role of R1, R2, R3, and R4 in four dimensional array resembles the notion of row and column in two dimensional arrays, where we can name 'row' as R1 and 'column' as R2. Consider the size of an integer as 4 bytes. If the compile time layout is implemented with R3, R4, R1, R2 form (in this order, where R2 is laid faster than R1, R1 is laid faster than R4, and so on), then the offset of element n[9][26][9][5] from base address of array 'n' is given by the number

0/1



```
program arraysdemo;
var
   n: array [6..10, 20..28, 7..13, 2..9 ] of integer;
   i, j,k, m: integer;

begin
   for i := 6 to 10 do
    for j:= 20 to 28 do
        for k:= 7 to 13 do
            for m:= 2 to 9 do
                n[ i ][j][k][m] := i + j+ k+ m;

   for i:= 6 to 10 do
      for j := 20 to 28 do
              for k:= 7 to 13 do
                for m:= 2 to 9 do
            writeln('Element[', i, j,k,m, '] = ', n[i][j][k][m] );
end.
```

100

Correct answer

3552

Anonymous variables are 1/1

- ● dynamically allocated memory on heap segment
- ○ none of these
- ○ statically allocated memory on heap segment
- ○ dynamically allocated memory on call stack
- ○ statically allocated memory on call stack

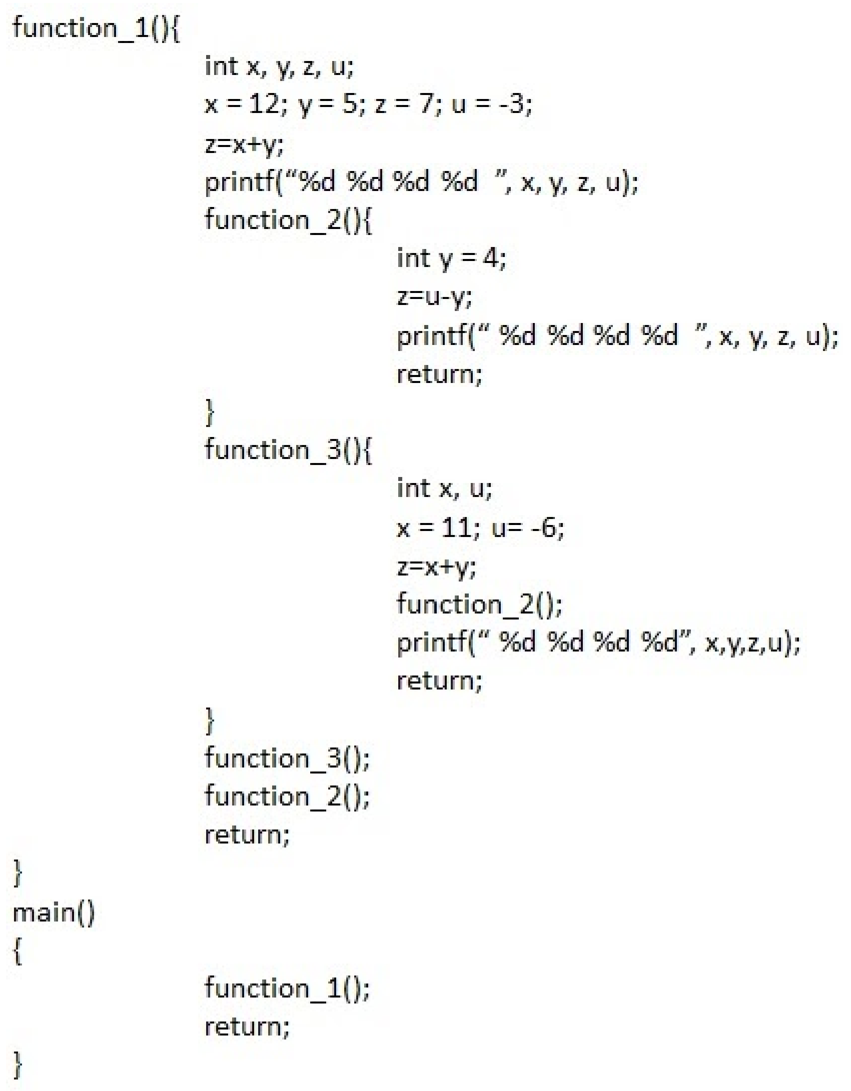
An arithmetic expression in a C-like language supports MINUS (-) operation on integers and real numbers. The arguments of minus should be of same type and the type of result is also as one of the argument's type. Type checking rule for the arithmetic expression derived from the production E2-->E1 - T is applied while traversing the corresponding sub tree. The language does not support implicit or explicit type conversion. The nonterminal E derives an expression involving array elements derived by nonterminal T. The basic element type is integer or float. If 'type' is an attribute associated with both non-terminals, then which of the following rules best describes type checking in expressions? [Note: Operations && and || below represent the logical AND and logical OR respectively. The subscripts 1 and 2 simply specify the position of the nonterminal in the rule E-->E-T] 1/1

- ○ None of these
- ○ { if ((E2.type = = INT) && (T.type == INT)) then E1.type = INT else if ((E2.type = = REAL) && (T.type == REAL)) then E1.type = REAL
- ● { if ((E1.type = = INT) && (T.type == INT)) then E2.type = INT else if ((E1.type = = REAL) && (T.type == REAL)) then E2.type = REAL }
- ○ { if ((E1.type = = INT) && (T.type == INT)) || ((E1.type = = REAL) && (T.type == REAL)) then E2.type = REAL }

A dangling pointer is a pointer 1/1

- ○ that contains the address of a stack-dynamic variable that has been allocated memory.
- ○ none of these
- [✓ that contains the address of a heap-dynamic variable that has been deallocated.]
- ○ that contains the address of a heap-dynamic variable that has been allocated memory.
- ○ that contains the address of a stack-dynamic variable that has been deallocated.

A language allows nested function definitions and implements static scoping rules. Consider the following C-like code written in this language. Which of the following blank separated value sequences best describes the output? 0/1



```
function_1(){
            int x, y, z, u;
            x = 12; y = 5; z = 7; u = -3;
            z=x+y;
            printf("%d %d %d %d  ", x, y, z, u);
            function_2(){
                        int y = 4;
                        z=u-y;
                        printf(" %d %d %d %d  ", x, y, z, u);
                        return;
            }
            function_3(){
                        int x, u;
                        x = 11; u= -6;
                        z=x+y;
                        function_2();
                        printf(" %d %d %d %d", x,y,z,u);
                        return;
            }
            function_3();
            function_2();
            return;
}
main()
{
            function_1();
            return;
}
```

- 12 5 17 -3 11 5 16 -6 12 4 -7 -3 12 4 -7 -3
- 12 5 17 -3 12 4 -7 -3 11 5 -7 -6 12 4 -7 -3
- [✓ 12 5 17 -3 11 4 -10 -6 11 5 -10 -6 12 4 -7 -3]
- None of these

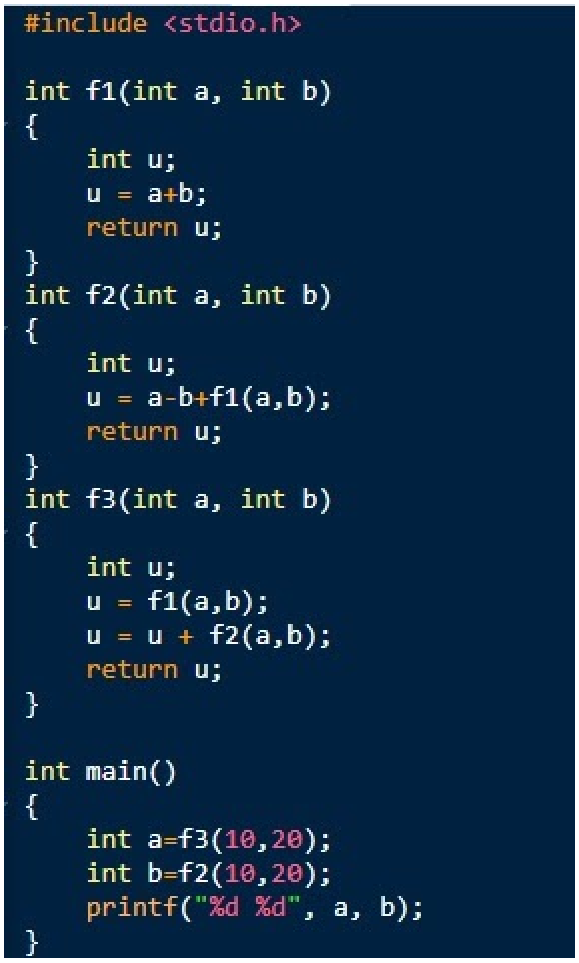
Correct answer

- [✓ 12 5 17 -3 12 4 -7 -3 11 5 -7 -6 12 4 -7 -3]

Consider an array type declared as "array [23..56] of integer" where it supports the subrange starting with index 23 and ending with 56. Which of the following languages supports subrange based array type? 1/1

- [✓ Ada and Pascal]
- C and C#
- Ada and C#
- None of these
- Pascal and C

Consider the following code written in C language. Which of the following ordering best describes the order of completion of execution of function calls? [A-->B indicates that execution of A finishes before the completion of execution of B] 1/1



```
#include <stdio.h>

int f1(int a, int b)
{
    int u;
    u = a+b;
    return u;
}
int f2(int a, int b)
{
    int u;
    u = a-b+f1(a,b);
    return u;
}
int f3(int a, int b)
{
    int u;
    u = f1(a,b);
    u = u + f2(a,b);
    return u;
}

int main()
{
    int a=f3(10,20);
    int b=f2(10,20);
    printf("%d %d", a, b);
}
```

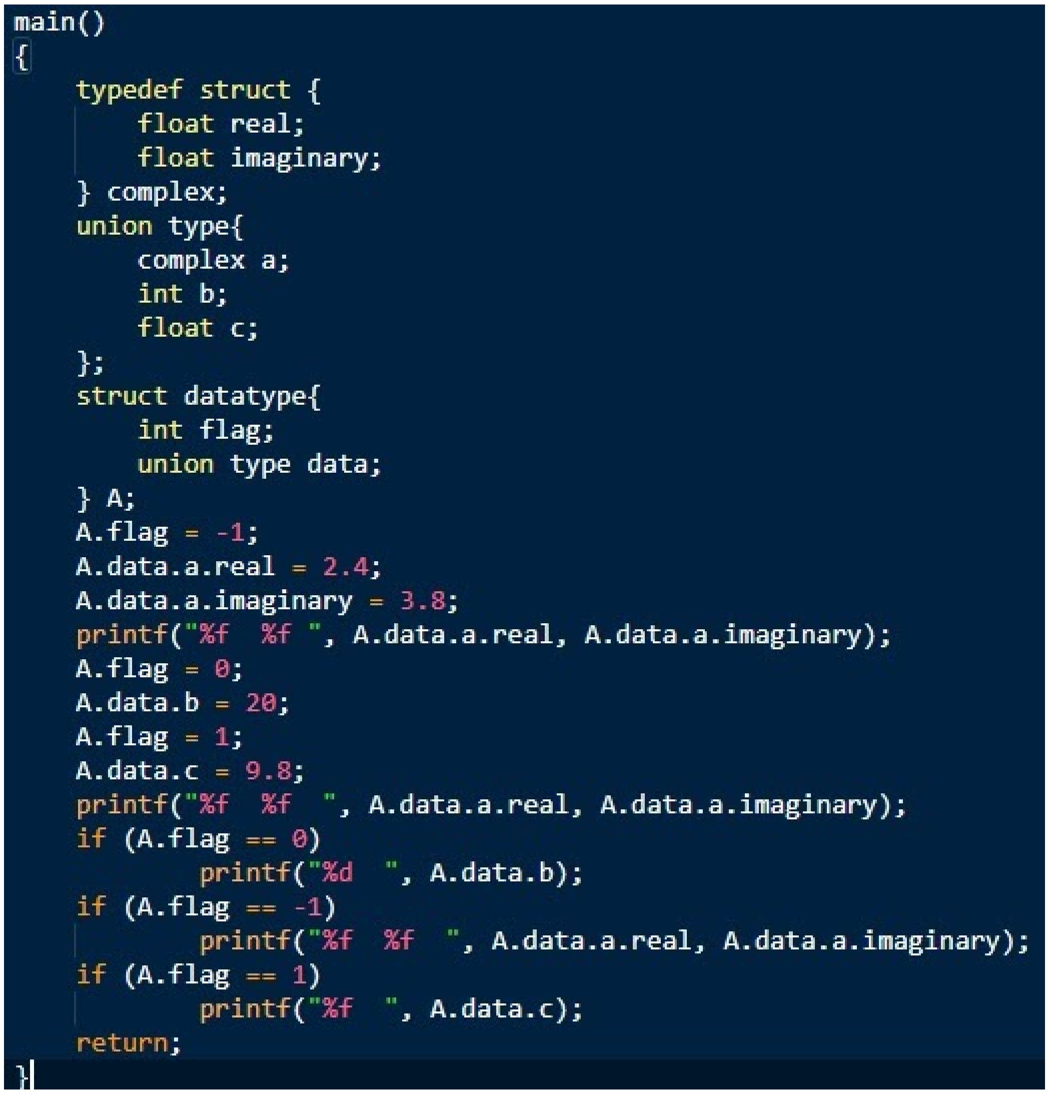
- f1() --->f1() ---> f2() --->f3() --->f2()--->f1()
- none of these
- f3() --->f2() ---> f1() --->f2() --->f1()--->f1()
- [✓ f1() --->f1() ---> f2() --->f3() --->f1()--->f2()]
- f3() ---> f1() --->f2() --->f1() --->f2()--->f1()

The grammar for declaring arrays in C-like language is <D>--><T> TK_ID TK_SQO <size> TK_SQC where the token names are used in their usual notations. If <T> --> TK_INT | TK_FLOAT, then the rule for nonterminal <size> for supporting variable sized arrays has on its right hand side

1/1

- ○ TK_FLOAT
- [✓ TK_ID]
- ○ TK_NUM
- ○ None of these

Consider the following code written in C language. What is the output of the program? 1/1



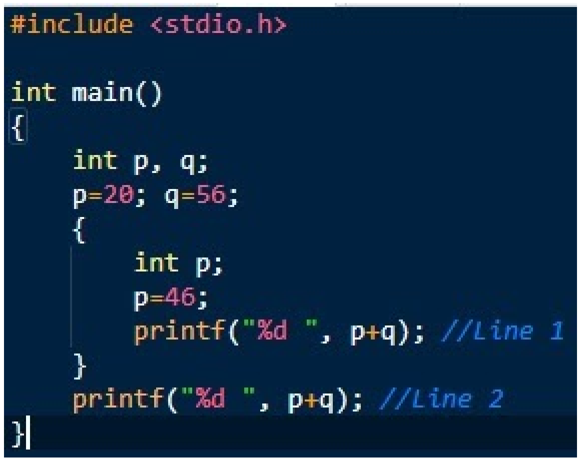
```
main()
{
    typedef struct {
        float real;
        float imaginary;
    } complex;
    union type{
        complex a;
        int b;
        float c;
    };
    struct datatype{
        int flag;
        union type data;
    } A;
    A.flag = -1;
    A.data.a.real = 2.4;
    A.data.a.imaginary = 3.8;
    printf("%f  %f ", A.data.a.real, A.data.a.imaginary);
    A.flag = 0;
    A.data.b = 20;
    A.flag = 1;
    A.data.c = 9.8;
    printf("%f  %f  ", A.data.a.real, A.data.a.imaginary);
    if (A.flag == 0)
            printf("%d  ", A.data.b);
    if (A.flag == -1)
            printf("%f  %f  ", A.data.a.real, A.data.a.imaginary);
    if (A.flag == 1)
            printf("%f  ", A.data.c);
    return;
}
```

- ○ 2.4 3.8 2.4 3.8 2.4 3.8
- ○ 2.4 3.8 2.4 3.8 20 2.4 3.8 9.8
- ○ None of these
- [✓ 2.4 3.8 9.8 3.8 9.8]

The scope of a variable refers to 1/1

- [✓ the range of statements in which a variable is visible.]
- the storage area where the variable is allocated memory
- the program text where the variable is not visible.
- the occurrence of a variable in an expression.

Consider the following code written in C language. Which statement best describes the code? 1/1



```
#include <stdio.h>

int main()
{
    int p, q;
    p=20; q=56;
    {
        int p;
        p=46;
        printf("%d ", p+q); //Line 1
    }
    printf("%d ", p+q); //Line 2
}
```

- [✓ The code implements static scoping and the output is 102 76]
- The code implements static scoping and the output is 76 102
- None of these
- The code implements dynamic scoping and the output is 76 102
- The code implements static scoping and the output is 102 102
- The code implements dynamic scoping and the output is 76 76
- The code implements dynamic scoping and the output is 102 76

The associative arrays supported in Perl language are named as 1/1

- None of these
- Dictionaries
- Jagged arrays
- [✓ Hashes]

The type checking in untagged union is done at run time. 1/1

- true
- [✓ false]

A function A() calls the function B(). Which of the following statements are correct? (1) Activation record of B() is at the top of the activation record of A(). (2) Activation record of function A() maintains the formal parameters used for communication between the caller and the callee, (3) Activation record of function B() maintains the base address of function A(). (4) Activation record of B() maintains the formal parameters. (5) Activation record of function A() maintains the actual parameters

1/1

- ○ Statements 2, 3, 4 and 5
- ○ Statements 1, 2 and 5
- ○ None of these
- ○ Statements 1, 3 and 5
- ○ Statements 2 and 3
- [✓ Statements 1, 3, 4 and 5]

A language implements nested function definitions with dynamic scope. The non-local variable in the active function is searched in the activation record of the 0/1

- ○ none of these
- ○ callee function who is the dynamic parent and sits immediately below the caller's activation record.
- [✓ caller function who is the static parent and sits immediately below the callee's activation record.]
- ○ callee function who is the static parent and sits immediately below the caller's activation record.
- ○ caller function who is the dynamic parent and sits immediately below the callee's activation record.

Correct answer

- [✓ caller function who is the dynamic parent and sits immediately below the callee's activation record.]

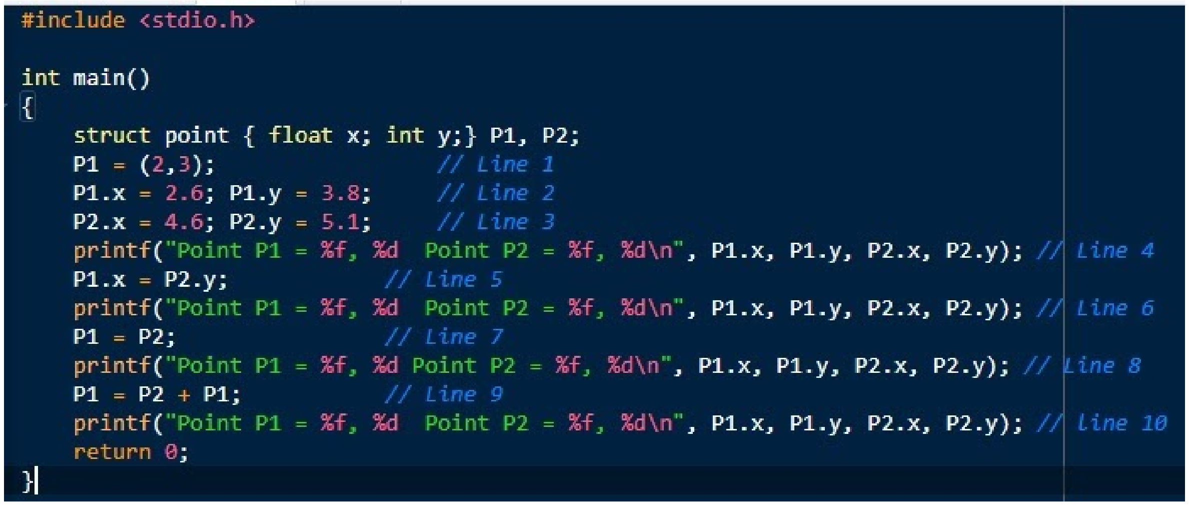
Use of untagged union in a program is 1/1

- ○ type safe
- ○ type equivalent
- ○ none of these
- [✓ type unsafe]

Which of the following languages allow nested subprograms? 1/1

- ○ C, Ada and Pascal
- [✓ Python, Common Lisp and Ada]
- ○ None of these
- ○ Ada, Python and C

Consider the following code written in C programming language. Which line numbers get the compilation errors? 1/1



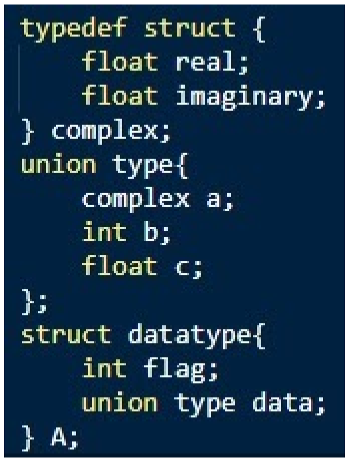
```
#include <stdio.h>

int main()
{
    struct point { float x; int y;} P1, P2;
    P1 = (2,3);                 // Line 1
    P1.x = 2.6; P1.y = 3.8;     // Line 2
    P2.x = 4.6; P2.y = 5.1;     // Line 3
    printf("Point P1 = %f, %d  Point P2 = %f, %d\n", P1.x, P1.y, P2.x, P2.y); // Line 4
    P1.x = P2.y;            // Line 5
    printf("Point P1 = %f, %d  Point P2 = %f, %d\n", P1.x, P1.y, P2.x, P2.y); // Line 6
    P1 = P2;                // Line 7
    printf("Point P1 = %f, %d Point P2 = %f, %d\n", P1.x, P1.y, P2.x, P2.y); // Line 8
    P1 = P2 + P1;           // Line 9
    printf("Point P1 = %f, %d  Point P2 = %f, %d\n", P1.x, P1.y, P2.x, P2.y); // Line 10
    return 0;
}
```

- ○ none of these
- ○ Lines 1, 5 and 7
- ○ Lines 1 and 7
- [✓ Lines 1 and 9]

The control and access links are usually the part of the activation record of a function. Which statement best describes the applicability of the control and access links for the given language? 1/1

- ○ The control link is used for all languages irrespective of the scoping method while the access link is used for languages that implement the dynamic scope.
- ○ None of these
- ○ The control link is used for languages that implement the static scope while the access link is used for languages that implement the dynamic scope.
- ○ The access link is used for all languages irrespective of the scoping method while the control link is used for languages that implement the static scope.
- [✓ The control link is used for all languages irrespective of the scoping method while the access link is used for languages that implement the static scope.]
- ○ The access link is used for all languages irrespective of the scoping method while the control link is used for languages that implement the dynamic scope.
- ○ The control link is used for languages that implement the dynamic scope while the access link is used for languages that implement the static scope.

Consider the following declaration of variable A. The value of flag field is used to prevent users from accessing spurious values at run time. Which type expression best defines the type of A at compile time? 0/1



```c
typedef struct {
    float real;
    float imaginary;
} complex;
union type{
    complex a;
    int b;
    float c;
};
struct datatype{
    int flag;
    union type data;
} A;
```

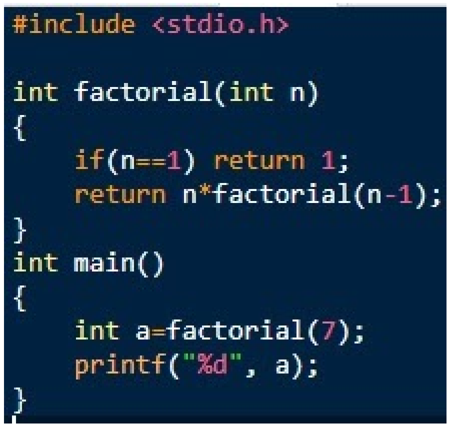
- <a X b X c> implemented as a record of type information at symbol table level with all fields used for type checking at run time based on value of flag.
- <<float X float> X int X float> implemented as a union of type information at symbol table level with only one of the fields used for type checking at run time based on value of flag.
- [✓ <<float X float> | int | float> implemented as a union of type information at symbol table level with only one of the fields used for type checking at run time based on value of flag.]
- <<float X float> | int | float> implemented as a record of type information at symbol table level with only one of the fields used for type checking at run time based on value of flag.
- None of these
- <<float X float> X int X float> implemented as a record of type information at symbol table level with all fields used for type checking at run time based on value of flag.

Correct answer

- [✓ <<float X float> | int | float> implemented as a record of type information at symbol table level with only one of the fields used for type checking at run time based on value of flag.]

Consider the following code written in C language. If the sizes of the activation records (in bytes) of functions main() and factorial() are p and q respectively, then the maximum size of the call stack in executing the given program is achieved as

1/1

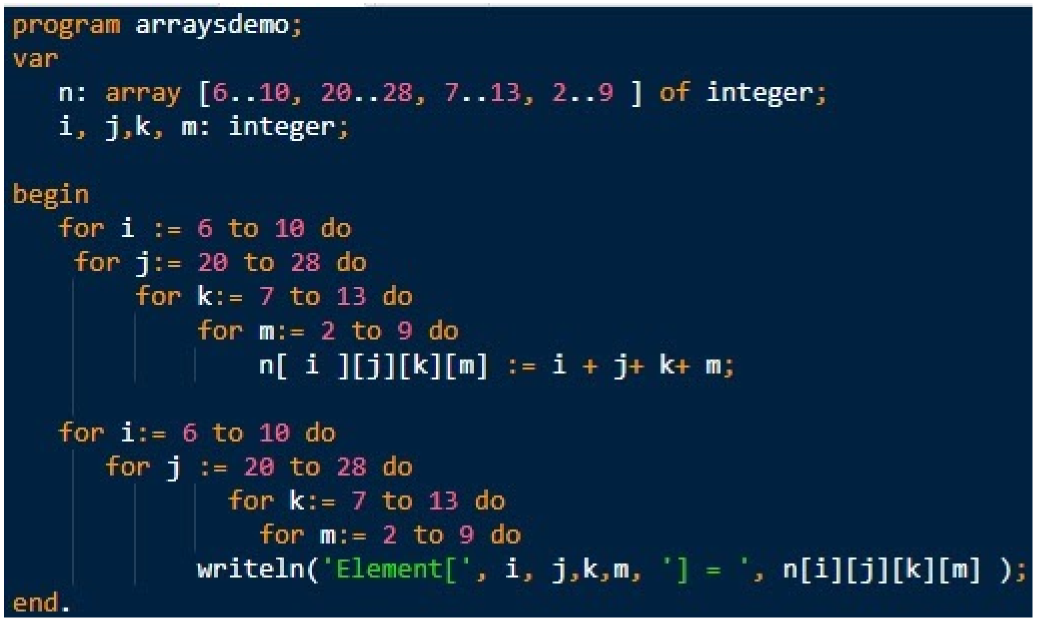


```
#include <stdio.h>

int factorial(int n)
{
    if(n==1) return 1;
    return n*factorial(n-1);
}
int main()
{
    int a=factorial(7);
    printf("%d", a);
}
```

- ○ none of these
- [✓ p+7*q]
- ○ 6*p+7*q
- ○ 7*q
- ○ p*q+7
- ○ p+6*q
- ○ p*7+q

Consider a four dimensional array 'n' of integers declared in a Pascal like language as shown in the following figure. The ranges 6..10, 20..28, 7..13 and 2..9 are defined in four directions R1, R2, R3 and R4 respectively. The role of R1, R2, R3, and R4 in four dimensional array resembles the notion of row and column in two dimensional arrays, where we can name 'row' as R1 and 'column' as R2. Consider the size of an integer as 4 bytes. If the compile time layout is implemented with R1, R2, R3, R4 form (in this order, where R4 is laid faster than R3, R3 is laid faster than R2, and so on), then the offset of element n[9][26][9][5] from base address of array 'n' is given by the number

0/1



```
program arraysdemo;
var
   n: array [6..10, 20..28, 7..13, 2..9 ] of integer;
   i, j,k, m: integer;

begin
   for i := 6 to 10 do
    for j:= 20 to 28 do
        for k:= 7 to 13 do
            for m:= 2 to 9 do
                n[ i ][j][k][m] := i + j+ k+ m;

   for i:= 6 to 10 do
      for j := 20 to 28 do
              for k:= 7 to 13 do
                for m:= 2 to 9 do
            writeln('Element[', i, j,k,m, '] = ', n[i][j][k][m] );
end.
```

100

Correct answer

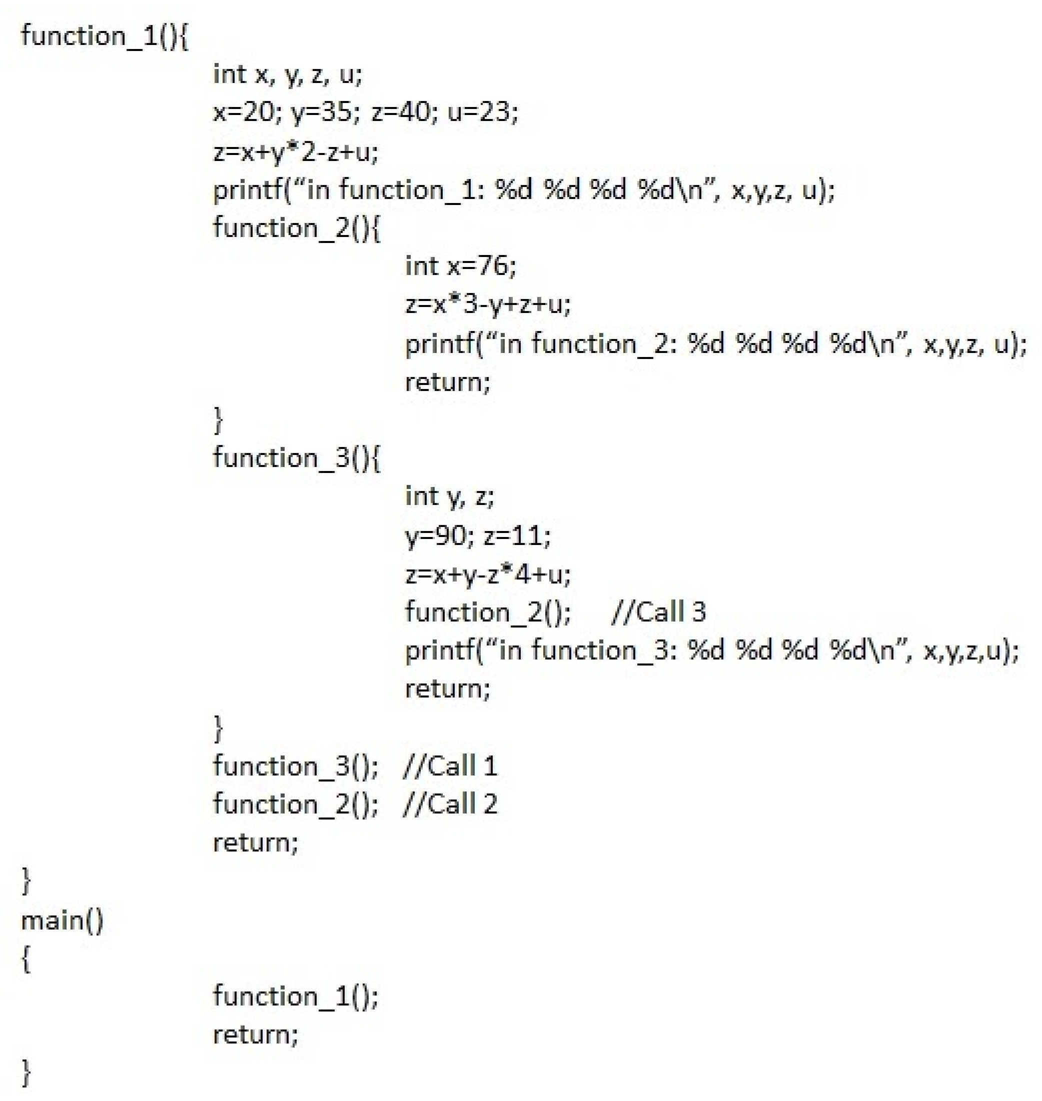
7468

Which languages support concatenation operation on single dimension arrays? 1/1

- ○ None of these
- ○ Java, C and Python
- [✓ Python, Ada and Ruby]
- ○ Python, C and C#

A language allows nested function definitions and implements dynamic scoping rules. Consider the following C-like code written in this language. Function function_1() invokes two functions function_3() (say call 1) and function _2() (say call 2). Function function_3() during call 1, in turn calls function_2() (say call3). The values of variable 'y' during calls 1, 2 and 3 each are

0/1



```
function_1(){
            int x, y, z, u;
            x=20; y=35; z=40; u=23;
            z=x+y*2-z+u;
            printf("in function_1: %d %d %d %d\n", x,y,z, u);
            function_2(){
                        int x=76;
                        z=x*3-y+z+u;
                        printf("in function_2: %d %d %d %d\n", x,y,z, u);
                        return;
            }
            function_3(){
                        int y, z;
                        y=90; z=11;
                        z=x+y-z*4+u;
                        function_2();     //Call 3
                        printf("in function_3: %d %d %d %d\n", x,y,z,u);
                        return;
            }
            function_3();  //Call 1
            function_2();  //Call 2
            return;
}
main()
{
            function_1();
            return;
}
```

- 90, 35 and 90 respectively during calls 1, 2 and 3
- [✓ 90, 90 and 35 respectively during calls 1, 2 and 3]
- None of these
- 35, 90 and 35 respectively during calls 1, 2 and 3

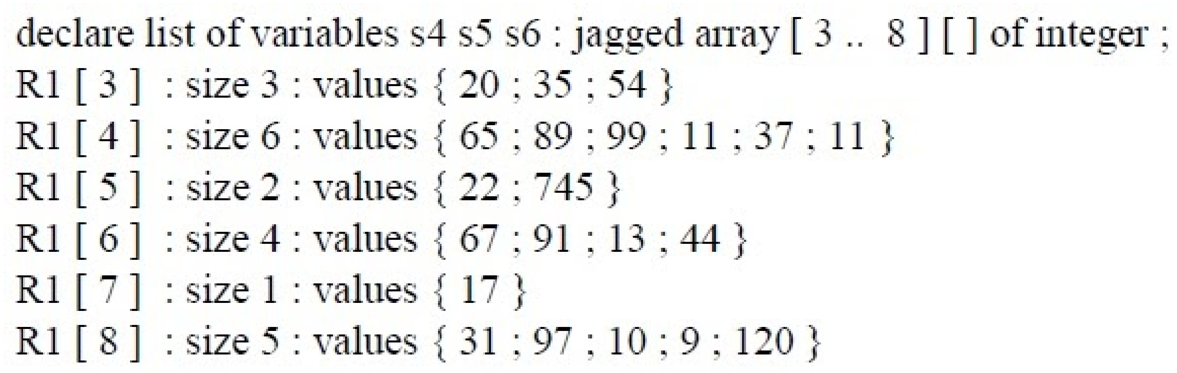
Correct answer

[✓ 90, 35 and 90 respectively during calls 1, 2 and 3]

A language with nested function definitions supports dynamic scoping. Which of the following statements is correct? 1/1

- Both, the offset of a non-local variable used in a function and its association with the declaration statement, are computed at compile time.
- The offset of a non-local variable used in a function is computed at run time but association to the declaration statement is possible only at compile time.
- None of these
- [✓ The offset of a non-local variable used in a function is computed at compile time but association to the declaration statement is possible only at run time.]
- Both, the offset of a non-local variable used in a function and its association with the declaration statement, are computed at run time.

Consider the following jagged array. An expression uses two elements as s4[50/1 8] + s5[4 3]. Which type expression best suits for bound checking for the array elements at compile time?



```
declare list of variables s4 s5 s6 : jagged array [ 3 ..  8 ] [ ] of integer ;
R1 [ 3 ]  : size 3 : values { 20 ; 35 ; 54 }
R1 [ 4 ]  : size 6 : values { 65 ; 89 ; 99 ; 11 ; 37 ; 11 }
R1 [ 5 ]  : size 2 : values { 22 ; 745 }
R1 [ 6 ]  : size 4 : values { 67 ; 91 ; 13 ; 44 }
R1 [ 7 ]  : size 1 : values { 17 }
R1 [ 8 ]  : size 5 : values { 31 ; 97 ; 10 ; 9 ; 120 }
```

- <type =jaggedArray, dimensions=6, range_R1=(3, 6, 2, 4, 1, 5), range_R2 = (3, 8), basicElementType = integer>
- [✓ < dimensions=2, range_R1=(3, 8), range_R2 = (3, 6, 2, 4, 1, 5), basicElementType = integer>]
- none of these
- <type =jaggedArray, dimensions=2, range_R2 = (3, 6, 2, 4, 1, 5), basicElementType = integer>
- <type =jaggedArray, dimensions=2, range_R1=(3, 8), range_R2 = (3, 6, 2, 4, 1, 5), basicElementType = integer>

Correct answer

- [✓ <type =jaggedArray, dimensions=2, range_R1=(3, 8), range_R2 = (3, 6, 2, 4, 1, 5), basicElementType = integer>]

The static array is allocated memory in the global area. The disadvantage of static array is 1/1

- ○ The array is available to all functions during the entire execution of the program.
- ○ The array need not be explicitly deallocated.
- [✓ The storage for the array is fixed for the entire duration of the program.]
- ○ None of these.
- ○ The subscript ranges are bound at compile time.

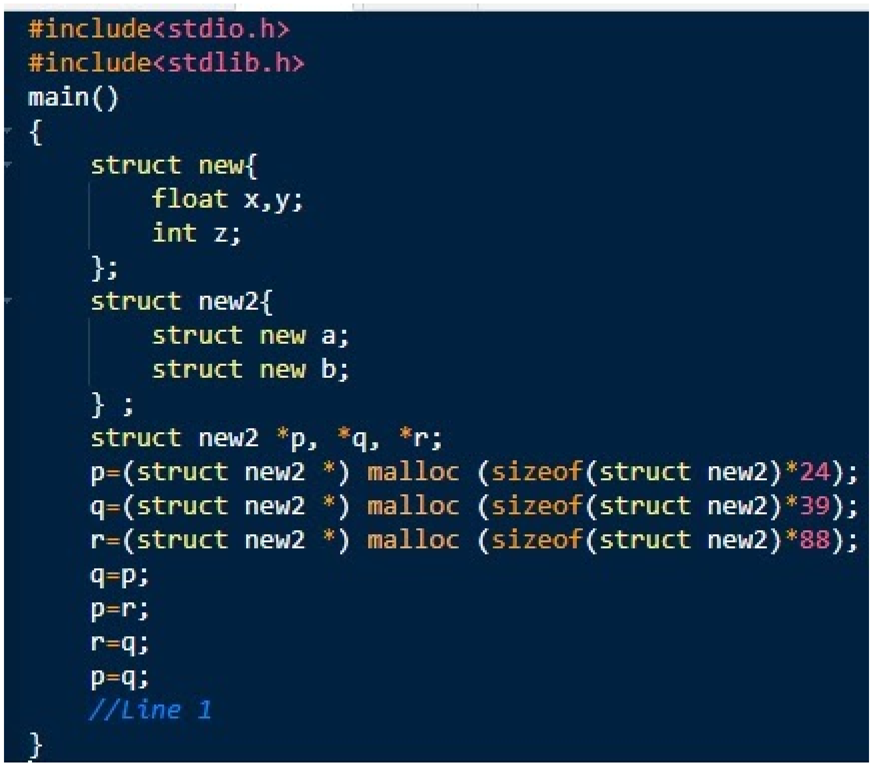
A language supports nested function definitions. Which of the following statements is correct? 1/1

- ○ The type checking of non-local variables in an expression is done at run time, if the language implements static scoping.
- [✓ The type checking of non-local variables in an expression is done at compile time, if the language implements static scoping.]
- ○ The type checking of non-local variables in an expression is done at compile time, if the language implements dynamic scoping.
- ○ None of these

Scope rules of a language determine 1/1

- ○ Which function calls another function having a variable named x
- ○ Which variables are using static scoping and which ones are using dynamic scoping
- ○ Which scoping method should be used for computation of expressions
- ○ none of these
- [✓ which declaration of a name x applies to an occurrence of x in a program.]

Consider the following code written in C language. If an integer uses 4 bytes and a floating point number uses 8 bytes, then what is the size of memory leak at line 1? 1/1



```
#include<stdio.h>
#include<stdlib.h>
main()
{
    struct new{
        float x,y;
        int z;
    };
    struct new2{
        struct new a;
        struct new b;
    } ;
    struct new2 *p, *q, *r;
    p=(struct new2 *) malloc (sizeof(struct new2)*24);
    q=(struct new2 *) malloc (sizeof(struct new2)*39);
    r=(struct new2 *) malloc (sizeof(struct new2)*88);
    q=p;
    p=r;
    r=q;
    p=q;
    //Line 1
}
```

5080

This form was created inside BITS Pilani University.

Google Forms