Quiz 3 (October 7, 2020)

Total points 24/32

Write your name and ID correctly.
There are 18 questions in this quiz. Each question carries 1 mark. The total time duration for the quiz is 30 minutes. Answer the questions and 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

Name *

Shreyas Kera

ID *

2018A7PS1119P

Questions 1-18 24 of 32 points

The assignment of one variable of pointer type to another variable of the same type, where both of these variables are used to manage storage on heap, can lead to 1/1

- dangling pointer and dereferencing
- none of these
- dangling pointer
- memory leak and parameter passing
- dereferencing
- parameter passing
- memory leak
- memory leak and dangling pointer

The variable p of pointer data type is used to manage storage on heap. The deallocation of memory by user using free(p); is written in C-like language, then an access to the heap dynamic variable pointed to by p can lead to 0/1

- ○ parameter passing
- ○ dangling pointer
- ○ memory leak and dangling pointer
- ○ memory leak
- ○ dereferencing
- ○ memory leak and parameter passing
- ○ none of these
- ◉ dangling pointer and dereferencing

Correct answer

- ◉ dangling pointer

Consider the following code written in Ada programming language. Tick one correct answer choice for each entity written in each row below.

```
with Ada.Text_IO; use Ada.Text_IO;
procedure shapedemo is
  type shape is (circle, triangle, rectangle);
  type colors is  (red, green, blue);
  type figure (form : shape) is
    record
        filled : boolean;
        color: colors;
        case form is
            when circle =>
                diameter : float;
            when triangle =>
                left_side : integer;
                right_side : integer;
                angle : float;
            when rectangle =>
                side_1:integer;
                side_2:integer;
        end case;
    end record;
    f1: figure (form=>rectangle) ;
    f2: figure (form=>triangle);
begin
    f1:= (filled=>true, color=>blue, form=>rectangle, side_1=>12, side_2=>3);
end shapedemo;
```

	variable	name of field	constrained variable declaration	discriminator	data type	tag value	none of these
side_1	○	○	○	○	○	○	◉
form	○	○	○	◉	○	○	○
form=>triangle	○	◉	○	○	○	○	○
figure	○	○	○	○	◉	○	○
f2: figure (form=>triangle);	○	○	◉	○	○	○	○
circle	○	○	○	○	○	◉	○
f1	◉	○	○	○	○	○	○

Correct answers

	variable	name of field	constrained variable declaration	discriminator	data type	tag value	none of these

side_1	○	◉	○	○	○	○	○
form=>triangle	○	○	○	◉	○	○	○

Which representation for variables of set data type can give O(1) time complex set union operation on the variables? 1/1

- ○ none of these
- ○ sets represented as an array
- ◉ set represented as bit string
- ○ set represented as linked list

The type checking of a variable of following types is done at (Tick any one of the four choices for answering for the type mentioned at each row)

	run time	compile time	cannot be checked	linking time	Score
variant record with discriminator	○	○	◉	○	0/1
tagged union	○	○	○	◉	0/1
pointer data type	◉	○	○	○	0/1
untagged union	◉	○	○	○	0/1
record data type	○	◉	○	○	1/1

Correct answers

	run time	compile time	cannot be checked	linking time
variant record with discriminator	◉	○	○	○
tagged union	◉	○	○	○
pointer data type	○	◉	○	○
untagged union	○	○	◉	○

Which data type displays the adjoining properties [You can tick more than 1 choice if you wish so. Answer for each row separately]

	fields can be of different types	fields are always of same types	Memory requirement is fixed and is compile time computable	Memory requirement may change and cannot be computed at compile time.	Score
variant record	☑	☐	☑	☐	1/1
record	☑	☐	☑	☐	1/1
union	☑	☐	☑	☐	1/1

A compiler which scans the source code more than three times is considered to be inefficient. The type expression of a variable of union type with tag, 0/1

- ○ is computed efficiently at run time using the tag value.
- ○ is computed efficiently at compile time using the tag value.
- ◉ is computed efficiently at run time using the type of the tag.
- ○ is computed efficiently at compile time using the type of the tag.
- ○ none of these

Correct answer

- ◉ is computed efficiently at run time using the tag value.

Which of the following statements best suits to define anonymous variable's storage and initial accessibility? The anonymous variable 1/1

- resides in heap segment of the memory and is accessed only by pointer variable stored in heap memory.
- resides in call stack segment of the memory and is accessed only by pointer variable stored in call stack memory.
- **resides in heap segment of the memory and is accessed only by pointer variable stored in call stack.**
- resides in call stack segment of the memory and is accessed only by pointer variable stored in heap memory.

A variable x of integer type is allocated memory on the call stack with starting address as 0198 (represented address in decimal in this question). Another variable p declared as int *p=&x; in C-like language is allocated memory location 0202 at the call stack. if an assignment statement is given by x = 0210; then locations 0202 and 0198 respectively contain following 1/1

- 0210 in both locations
- none of these
- **0198 and 0210 respectively**
- 0210 and 0202 respectively
- 0210 and 0198 respectively

The sets A={10, 20, 18, 6, 7} and B= {7, 10, 19, 18, 13, 5} can best be represented efficiently for set intersection as follows 1/1

- {01010, 10100, 10010, 00110, 00111} and { 00111, 01010, 10011, 10010, 01101, 00110}
- none of these
- A[5] and B[6]
- 101000000010011000000 and 011000010010010100000

The information about the type expression of a variable of tagged union type is maintained at compile time in the symbol table. It maintains the 1/1

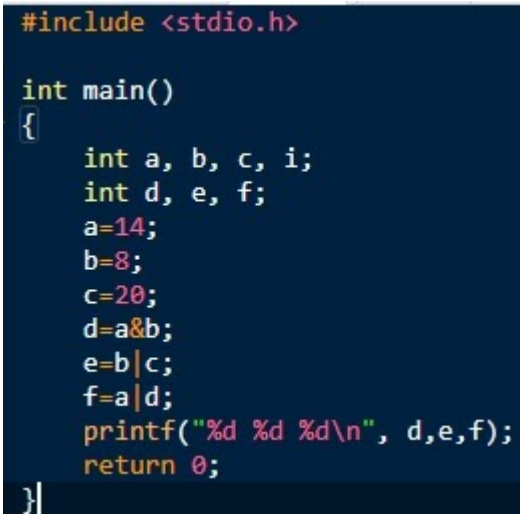
- types of all fields of the tagged union data type irrespective of tag value which is not included at run time.
- types of the field of the tagged union data type being currently used as identified by the tag value
- types of all fields of the tagged union data type and the value of the tag.
- types of all fields of the tagged union data type irrespective of tag value which is included at run time.
- none of these

Consider the following code written in C-like language. If the sizes of integer and float data types are 4 and 8 bytes respectively, what is the size of memory leak at Line 1 (in number of bytes)? 1/1

```c
#include <stdio.h>
#include<stdlib.h>
int f1(void)
{
    int *p;
    float *q;
    int x=20;
    float y;
    p=(int *)malloc(sizeof(int)*6);
    q=(float *) malloc(sizeof(float)*9);
    p=&x;
    free(q);
    return x;
}
int main()
{
    int a;
    a=f1();
    printf("%d\n", a);
}
```

24

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



```c
#include <stdio.h>

int main()
{
    int a, b, c, i;
    int d, e, f;
    a=14;
    b=8;
    c=20;
    d=a&b;
    e=b|c;
    f=a|d;
    printf("%d %d %d\n", d,e,f);
    return 0;
}
```

- 22 -16 -8
- 28 14 8
- true false true
- none of these
- 8 28 14
- 0 1 1

Consider the following C program. The sizes of integer and float data types are 4 and 8 respectively. What is the size of memory leak in terms of number of bytes exactly before the execution of the return statement? 1/1

```c
#include <stdio.h>
#include<stdlib.h>
int main()
{
    int *p, *q, *r;
    float *ptr1, *ptr2, *ptr3;
    int x, y;
    float u, v;
    p=(int *) malloc(sizeof(int)*35);
    q=(int *) malloc(sizeof(int)*78);
    r=(int *) malloc(sizeof(int)*18);
    ptr1= (float *) malloc(sizeof(float)*103);
    ptr2= (float *) malloc(sizeof(float)*22);
    ptr3= (float *) malloc(sizeof(float)*50);
    p = q;
    q = r;
    p = r;
    ptr1=ptr2;
    ptr3=ptr1;
    return 0;
}
```

1676

Unconstrained variant records in Ada language allow the values of their variants to change types during execution. 1/1

- ○ false
- ◉ true

Consider the following code written in C language. If size of the integer data type is 4 bytes, then what is the size of anonymous variable (in bytes)? 1/1

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

    int list[]={10, 28, 19, 97, 22, 87, 30, 55};
    int *p;
    p = (int *) malloc(sizeof(int)*30);
    return 0;
}
```

- none of these
- 156
- 36
- 120 (selected)
- 32

Consider the following code written in C programming language. What is the output of the following code? (Comma added in the output only for clarity) 1/1

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

    int list[]={10, 28, 19, 97, 22, 87, 30, 55};
    int *p;
    p=&list[2];
    printf("%d %d %d %d \n", *p, *(p+2), *list, *list+2); //Line 1
}
```

- ○ 97, 87, 10, 28
- ○ none of these
- ◉ 19, 22, 10, 12
- ○ 10, 19, 10, 22

Consider the following user defined types. If the sizes of integer and float data type are 4 and 8 respectively, what are the sizes of the following data types in bytes?[Select the answer for each row]

```c
#include <stdio.h>

int main()
{
    struct old{
        float a1, b1;
        int a2,b2, c2;
    };
    union new{
            int w;
            int p;
            float q;
            struct old K;
        } ;
    struct info{
        int a, b;
        float c;
        union new M;
        float w;
    };
    printf(" %d %d %d\n", sizeof(struct info), sizeof(union new), sizeof(struct
        old));
    return 0;
}
```

	20	36	52	28	44	none of these	Score
sizeof(struct old)	○	○	○	◉	○	○	1/1
sizeof(union new)	○	○	○	◉	○	○	1/1
sizeof(struct info)	○	○	◉	○	○	○	1/1