08-10-2021 | CS 210 | DS | Class Test

u20cs107@coed.svnit.ac.in Switch account

Draft saved

Your email will be recorded when you submit this form

08-10-2021 | CS 210 | DS | Class Test

Select the correct option

i. First-in-first out types of computations are efficiently supported by STACKS.
ii. Implementing QUEUES on a circular array is more efficient than implementing QUEUES on a linear array with two indices.
iii. Last-in-first-out types of computations are efficiently supported by QUEUES.

Select the correct options.
(A) (i) and (ii) are true
(B) Only (iii) is true
(C) (ii) and (iv) are true
(D) All are true

- ○ A
- ○ B
- ○ C
- ○ D

Which of the following is correct in this context?

We have the following declarations in 2 source files.

file1.c

```
int array[ ] = {1,2,3};
```

file2.c

```
extern int array[ ];
void main()
{
    printf("%d", sizeof(array));
}
```

- sizeof(array) defined in file1.c is only known at run time and so exception of file2.c gives a run time error.
- Compilation of file2.c will fail as sizeof() operates at compile time and is not able to learn the size of an array that is defiled in another file.
- Use of sizeof(array) in file2.c is perfectly fine, outputs size of an integer pointer on the system.
- Segmentation fault occurs at run time

If Q represents a queue and ‘k’ is a global parameter. Suppose the queue is initially empty then what would be the time complexity if function Demo() is called ‘n’ times.

```
Demo(Q){
  m = k
  while (Q is not empty and m  > 0) {
    Dequeue(Q)
    m = m - 1
  }
}
```

- ◯ Θ(n+k)
- ◯ Θ(n)
- ◯ Θ(nk)
- ◯ Θ(n^2)

What is the output of the above program?

```
Int arr[ ] = {2,3,4,1,6}
printf(“%d%d”,*arr,sizeof(arr));
```

- ◯ 2 10
- ◯ 2 9
- ◯ 0 10
- ◯ None

int(*ptr)[10];What does the following declaration mean

- ○ ptr is a pointer to an array of 10 integer
- ○ ptr is an array which points 10 pointer
- ○ ptr is pointer which pointer 10 element
- ○ None of these

What is the time complexity to count the number of elements in the linked list?

- ○ O(1)
- ◉ O(n)
- ○ O(n^2)
- ○ O(log n)

Clear selection

What is the output of following function for start pointing to first node of following linked list?1->2->3->4->5->6

```
void fun(struct node* start)
{
   if(start == NULL)
   return;
   printf("%d  ", start->data);
   if(start->next != NULL )
   fun(start->next->next);
   printf("%d  ", start->data);
}
```

- ○ 1 4 6 6 4 1
- ○ 1 3 5 1 3 5
- ○ 1 2 3 5
- ○ 1 3 5 5 3 1

The maximum size of the queue ?

- ○ can be changed
- ◉ can not be changed
- ○ independent
- ○ None of these

Clear selection

Which of the following stack operations could result in stack underflow?

- ○ Is_empty
- ○ pop
- ○ push
- ○ peek

The output for this program is:

```
int a[5] = {2,3};
printf("\n%d%d%d", a[2],a[3],a[4]);
```

- ○ Garbage value
- ○ 2 3 3
- ○ 3 2 2
- ○ 0 0 0

The following function reverse() is supposed to reverse a singly linked list. There is one line missing at the end of the function.

```
struct node
{
   int data;
   struct node* next;
};
static void reverse(struct node** head_ref)
{
   struct node* prev   = NULL;
   struct node* current = *head_ref;
   struct node* next;
   while (current != NULL)
   {
      next  = current->next;
      current->next = prev;
      prev = current;
      current = next;
   }
   /*ADD A STATEMENT HERE*/
}
```

What should be added in place of "/*ADD A STATEMENT HERE*/", so that the function correctly reverses a linked list.

- (●) *head_ref = prev;
- ( ) *head_ref = current;
- ( ) *head_ref = next;
- ( ) *head_ref = NULL;

Clear selection

Consider the following doubly linked list: head-1-2-3-4-5-tail. What will be the list after performing the given sequence of operations?

```
Node temp = new Node(6, head,head.getNext());
Node temp1 = new Node(0,tail.getPrev(),tail);
head.setNext(temp);
temp.getNext().setPrev(temp);
tail.setPrev(temp1);
temp1.getPrev().setNext(temp1);
```

- ○ head-0-1-2-3-4-5-6-tail
- ○ head-1-2-3-4-5-6-tail
- ◉ head-6-1-2-3-4-5-0-tail
- ○ head-0-1-2-3-4-5-tail

Clear selection

The postfix expression for the infix expression(A+B*(C+D))/(F+D*E) is

- ◉ (AB+CD+*F)/D+E*
- ○ (ABCD+*+)/(FDE*+)
- ○ (A*B+CD)/F*DE++
- ○ (ABCD+*+FDE*+/

Clear selection

Consider a standard Circular Queue 'q' implementation (which has the same condition for Queue Full and Queue Empty) whose size is 11 and the elements of the queue are q[0], q[1], q[2].....,q[10]. The front and rear pointers are initialized to point at q[2] . In which position will the ninth element be added?

- (●) q[0]
- ( ) q[1]
- ( ) q[9]
- ( ) q[10]

Clear selection

```
int *array1[8];
int *(array2[8]);
```

A. Array of pointers
B. Pointer to an array

Which is correct?

(a) array1 is A, array2 is B
(b) array1 is B, array2 is A
(c) array1 is A, array2 is A
(d) array1 is B, array2 is B

- ( ) (a)
- ( ) (b)
- ( ) (c)
- ( ) (d)

Which one of the following is an application of stack data structure?

- ○ Managing function calls
- ○ The stock span problem
- ○ Arithmetic expression evaluation
- ◉ All

Clear selection

The max size of stack to check (()((()))

Your answer

The value of the postfix expression 2 3 2 ^ + 4 8 * +

Your answer

The minimum number of stacks needed to implement a queue is

- ○ 1
- ◉ 2
- ○ 3
- ○ 4

Clear selection

What is the functionality of the following piece of code?

```
public int function()
{
        Node temp = tail.getPrev();
        tail.setPrev(temp.getPrev());
        temp.getPrev().setNext(tail);
        size--;
        return temp.getItem();
}
```

- ○ Return the element at the tail of the list but do not remove it
- ◉ Return the element at the tail of the list and remove it from the list
- ○ Return the last but one element from the list but do not remove it
- ○ Return the last but one element at the tail of the list and remove it from the list

Clear selection

Back Submit Clear form