SBE 201 - Data Structure
Midterm Exam
Model Answer

---

**1.**

a. _____ form of access is used to add and remove nodes from a stack.

i. **LIFO, Last In First Out**
ii. FIFO , First In First Out
iii. Both 1 and 2
iv. None of these

b. New nodes are added to the _____ of the queue.

i. front
ii. **back**
iii. middle
iv. Both 1 and 2

c. In C, why is the void pointer useful? When would you use it? Explain your answer by an example.

**The void pointer is useful because it is a generic pointer that any pointer can be cast into and back again without loss of information.**

**Example:**

**void* malloc (size_t size);**
**The malloc function allocates a block of *size* bytes of memory, returning a pointer to the beginning of the block. The type of this pointer is alwaysvoid*, which can be cast to the desired type of data pointer.**

d. What is the difference between Strings and Arrays in C?

**String is a sequence of characters ending with NULL .it can be treated as a one dimensional array of characters terminated by a NULL character.**

SBE 201 - Data Structure
Midterm Exam
Model Answer

e. Can the size of an array be declared at runtime? Explain your answer by an example.

**No. In an array declaration, the size must be known at compile time. You can't specify a size that's known only at runtime. For example, if i is a variable, you can't write code like this:**

**char array[i]; /* not valid C */**

**Some languages provide this latitude. C doesn't. If it did, the stack would be more complicated, function calls would be more expensive, and programs would run a lot slower.**

**If you know that you have an array but you won't know until runtime how big it will be, declare a pointer to it and use malloc() or calloc() to allocate the array from the heap.**

**If you know at compile time how big an array is, you can declare its size at compile time. Even if the size is some complicated expression, as long as it can be evaluated at compile time, it can be used.**

f. Can you add pointers together? Why would you?

**No, you can't add pointers together. If you live at 1332 Tahrir Street, and your neighbor lives at 1364 Tahrir Street, what's 1332+1364? It's a number, but it doesn't mean anything. If you try to perform this type of calculation with pointers in a C program, your compiler will complain.**

**The only time the addition of pointers might come up is if you try to add a pointer and the difference of two pointers:**

**p = p + ( p2 - p1 );** **OR** **p += p2 - p1;**

SBE 201 - Data Structure
Midterm Exam
Model Answer

**2.** Write a function to swap two adjacent elements in a doubly linked list by adjusting only the pointers (and not the data).

**Answer**

**here's the code you need to swap two adjacent node pointers assuming that x is previous to y:**

```
void swapAdjacent( Node &x, Node &y )
{
        if( x.prev ) x.prev->next = &y;
        if( y.next ) y.next->prev = &x;
        y.prev = x.prev;
        x.next = y.next;
        x.prev = &y;
        y.next = &x;
}
```

SBE 201 - Data Structure
Midterm Exam
Model Answer

**3.** We have a stream of numbers coming from Analog to Digital (A/D) converter; e.g. taken from continuous bio-signal measurement. We would like to store only the last n numbers. The oldest stored number should be removed to add the new coming number in case n numbers are already available. At the start of the measurement, the data structure is empty. What is the most appropriate data structure to store the stream of numbers. Justify your choice. And then implement the corresponding "insert" operation.

**Answer**

A circular queue is the most appropriate data structure to store the stream of numbers.

```
# define MAXQUEUE 1000                    /* size of the queue numbers*/
typedef struct {
        int front;
        int rear;
        float numbers[MAXQUEUE + 1];
} QUEUE;

QUEUE myQueue;
myQueue.front = 0;
myQueue.rear = MAXQUEUE;

void insert(QUEUE *qptr, float newNumber)
{
        if(qptr->front == (qptr->rear + 2) % MAXQUEUE)
                remove( qptr );
        qptr->rear++;
        qptr-> numbers[qptr->rear]= newNumber;
}
```

SBE 201 - Data Structure
Midterm Exam
Model Answer

**4.** Describe how you could use a single array to implement three stacks taking in consideration that any stack can grow as long as there is any free space in the array. Write a push function that can be used to insert an item into a selected stack of the three stacks.

**Answer:**

**In this approach, we sequentially allocate space to the stacks and we link new blocks to the previous block. This means any new element in a stack keeps a pointer to the previous top element of that particular stack.**

**In this implementation, we face a problem of unused space. For example, if a stack deletes some of its elements, the deleted elements may not necessarily appear at the end of the array .So, in that case, we would not be able to use those newly freed spaces.**

**To overcome this deficiency, we can maintain a free list and the whole array space would be given initially to the free list. For every insertion, we would delete an entry from the free list. In case of deletion, we would simply add the index of the free cell to the free list.**

**In this implementation we would be able to have flexibility in terms of variable space utilization but we would need to increase the space complexity.**

```
typedef struct {        int previous;  int value;} StackNode;

int stackSize = 300;
int indexUsed = 0;
int stackPointer[3]= {-1,-1,-1};
StackNode buffer[stackSize * 3];
void push(int stackNum, int value) {
        int lastIndex = stackPointer[stackNum];
        stackPointer[stackNum] = indexUsed;
        indexUsed++;
        buffer[stackPointer[stackNum]].previous = lastIndex
        buffer[stackPointer[stackNum]].value = value;
}

int pop(int stackNum) {
        int value = buffer[stackPointer[stackNum]].value;
        int lastIndex = stackPointer[stackNum];
        stackPointer[stackNum] = buffer[stackPointer[stackNum]].previous;
        buffer[lastIndex] = null;
        indexUsed--;
        return value;
}
int peek(int stack) { return buffer[stackPointer[stack]].value; }
boolean isEmpty(int stackNum) { return stackPointer[stackNum] == -1; }
```