

School of Computer Science and Software Engineering  
Clayton Campus, Monash University

**CSE1303 Part A**  
**Summer Semester, 2002**

## **Tutorial 4: Linked Structures Solutions**

### **Exercise 1.**

```
/* Return the size of the stack */
int
stackSize(Stack* stackPtr)
{
    int count = 0;
    Node* nodePtr = stackPtr->top;

    while (nodePtr != NULL)
    {
        nodePtr = nodePtr->next;
        count++;
    }

    return count;
}
```

### **Exercise 2.**

(a) 2A40 (b) 3040 (c) 1.5 (d) FE20

### **Exercise 3.**

```
/* Delete all the items in the queue */
void
clearQueue(Queue* queuePtr)
{
    Node* nodePtr = queue->front;
    Node* oldNode = NULL;

    while (nodePtr != NULL)
    {
        oldNodePtr = nodePtr;
        nodePtr = nodePtr->next;
        free(oldNodePtr);
    }

    queuePtr->front = NULL;
    queuePtr->rear = NULL;
    queuePtr->count = 0;
}
```

**Exercise 4.**

(a)

```
/* Given a position in the list, this function returns a
 * pointer to the node in that position. Otherwise it
 * returns NULL.
 */
```

```
Node*
```

```
setPosition(List* listPtr, int position)
```

```
{
```

```
    int i;
```

```
    Node* nodePtr = listPtr->head;
```

```
    if (position < 0 || position >= listPtr->count)
```

```
    {
```

```
        nodePtr = NULL;
```

```
    }
```

```
    else
```

```
    {
```

```
        for (i = 0; i < position; i++)
```

```
        {
```

```
            nodePtr = nodePtr->next;
```

```
        }
```

```
    }
```

```
    return nodePtr;
```

```
}
```

(b)

```
/* Insert an item at a position in a Linked List */

void
insert(List* listPtr, float item, int position)
{
    Node* newNodePtr = makeNode(item);
    Node* nodePtr = setPosition(listPtr, position-1);

    if (newNodePtr != NULL)
    {
        if (position == 0)
        {
            newNodePtr->next = listPtr->head;
            listPtr->head = newNodePtr;
        }
        else if (nodePtr != NULL)
        {
            newNodePtr->next = nodePtr->next;
            nodePtr->next = newNodePtr;
        }
        else
        {
            free(newNodePtr);
            fprintf(stderr, "Error inserting item\n");
            exit(1);
        }
        listPtr->count++;
    }
}
```

**Exercise 5.**

(a)

```
/* Make a new Node for a Double Linked List*/
Node*
makeNode(const char* s)
{
    Node* newNode = (Node*) malloc(sizeof(Node));

    if (newNode == NULL)
    {
        return NULL;
    }

    newNode->str = strdup(s);
    newNode->next = NULL;
    newNode->previous = NULL;

    return newNode;
}
```

(b)

```
/*
 * Deallocate the memory associated with a Node
 * for a Double Linked List
 */
void
killNode(Node* nodePtr)
{
    free(nodePtr->str);
    free(nodePtr);
}
```

**Exercise 6.**

```

char command[100];
char first[5];
char second[95];
long int num;

fgets(command, 100, stdin);
if (sscanf(command, "%s %s", first, second) == 2)
{
    if (strlen(first) == 2){
        if (first[0] == '$' && first[1] == 'r')
            printf("Reading from file: %s", second);
    }
    else if (strlen (first) == 1){
        if (first[0] == 'd'){
            num = atoi(second);
            printf("Deleting line %d", num);
        }
        else if (second[0] == 'p'){
            num = atoi(first);
            printf("Printing line number: %d", num-1);
        }
    }
}

```

\*\*\*\*strtol(char\* string, char\*\* endptr, int base) would be a better function to use as you can make sure that number did convert by checking the endptr.

---

or more elegantly:

```

char command[100];
char temp[100];
int num;
char ch;
char ch2;

fgets(command, 100, stdin);

if (sscanf(command, "$r %s", temp) == 1)
    printf("Reading from file: %s", temp);

else if (sscanf(command, "d %d", &num) == 1)
    printf("Deleting line %d", num);

else if (sscanf(command, "%d %c%c", &num, &ch, &ch2){
    if (ch2 == '\n' && ch == 'p')
        printf("Printing line number: %d", num-1);
}

```