Massey University
ALBANY CAMPUS

MID-TEST 159201
Semester One - 2012

Time Allowed: 40 Mins

INSTRUCTIONS

Attempt **ALL** questions.
Circle ONE answer for each question on this paper.

Write your ID number below.

ID No:_____

This test contributes 15% to the final assessment
(1 mark per question)

Calculators are permitted.

Turn over to pg. 2...

1. The following code snippet adds elements (Nodes) to a linked-list. Circle the correct statement about this code:

```
...
struct Node {  //declaration
  int number;
  Node *next;
};
Node *A;
...
void AddNode(Node * & listpointer, int a) {
  Node *temp;
  temp = new Node;
  temp->number = a;
  temp->next = listpointer;
  listpointer = temp;
}
...
```

a) Adds a node to the middle of the list.
b) Adds a node to the rear of the list.
c) Adds a node to the front of the list.
d) It fails to add a node because listpointer cannot be changed.

2. The following code snippet adds elements (Nodes) to a linked-list. Circle the correct statement about this code:

```
...
void AddNode(Node * listpointer, int a) {
  Node *temp;
  temp = new Node;
  temp->number = a;
  temp->next = listpointer;
  listpointer = temp;
}
...
```

a) Adds a node to the middle of the list.
b) Adds a node to the rear of the list.
c) Adds a node to the front of the list.
d) It fails to add a node because listpointer cannot be changed.

Turn over to pg. 3...

3. The following code snippet adds elements (Nodes) to a linked-list. Circle the correct statement about this code:

```
...
void AddNode(Node * & listpointer, int a) {
  Node *temp;
  Node *current;
  temp = new Node;
  temp->number = a;
  temp->next = NULL;
  if(listpointer==NULL){
   listpointer=temp;
  }
  else {
    current=listpointer;
    while(current->next!=NULL){
      current=current->next;
    }
    current->next=temp;
  }
}
...
```

a) Adds a node to the midlle of the list.
b) Adds a node to the rear of the list.
c) Adds a node to the front of the list.
d) It fails to add a node because listpointer cannot be changed.

4. Mark the option that best describes the names of the four basic operations for a standard **queue** (ADT):

a) Leave, Pop, Top, IsEmpty
b) Tree, Push, Top, IsEmpty
c) Leave, Join, Front, IsEmpty
d) Size, Push, Top, IsEmpty

5. Mark the option that best describes the names of the four basic operations for a standard **stack** (ADT):

a) Size, Push, Top, IsEmpty
b) Pop, Leave, Rear, IsEmpty
c) Pop, Pull, Top, IsEmpty
d) Pop, Push, Top, IsEmpty

6. The "**creeping problem**" can occur if:

Turn over to pg. 4...

a) a stack is implemented using an array.
b) a stack is implemented using a linked-list.
c) a queue is implemented using an array.
d) a stack is implemented using a linked-list.

7. One advantage of the doubly-linked list is that:

a) one can search the list backwards.
b) it saves space in memory.
c) it always sorts the elements automatically.
d) it does not require a pointer to the head of the linked-list.

8. Given the fragment of the code below, what is the correct way to refer to the price of book1 and book2?

```
...
struct Books{
      int callnumber;
      float price;
};
...
main(){
Books book1;
Books *book2;
...
}
```

a) book1.price and book2.price
b) book1->price and book2->price
c) book1.price and book2->price
d) book1->price and book2.price

9. We want to print all the elements of a **circular** linked-list. There is a missing **while** statement in the following code fragment. Which is the best option?

```
void Print_Circular_LL(Node *listpointer) {
  Node *current;
  current = listpointer;
  //MISSING STATEMENT
      printf("%d ",current->number);
      current = current->next;
  }
  printf("%d \n", current->number);
  printf("End of the list.\n");
}
```

a) while (current->next != listpointer->next) {
b) while (current != NULL) {
c) while (current->next != NULL) {
d) while (current->next != listpointer) {

Turn over to pg. 5...

10. While trying to delete an element in a **stack** implemented with a **linked-list**, the application crashes with a message for '*segmentation fault*'. What is the more likely cause, considering the code fragments below?

```
struct Node {
  float data;
  Node *next;
};

class Stack {
private:
  Node *listpointer;
public:
  Stack();
  ~Stack();
  void Push(float newthing);
  void Pop();
  float Top();
  bool isEmpty();
};
...
void Stack::Pop() {
   Node *p;
   if (listpointer != NULL) {
   listpointer = listpointer->next;
   delete p;
   }
}
```

a) a missing statement: `p = NULL;`
b) The method should not be void.
c) a missing statement: `p = listpointer;`
d) listpointer is not passed in the list of parameters

11. About **vectors** (ADT), which of the following options best describe its characteristics:

a) vectors cannot be extended, can be accessed via index, and are very good for random access.
b) vectors can be extended, but elements have to be accessed sequentially.
c) vectors can be extended, can be accessed via index, and are very good for random access.
d) vectors cannot be extended, and elements have to be accessed sequentially.

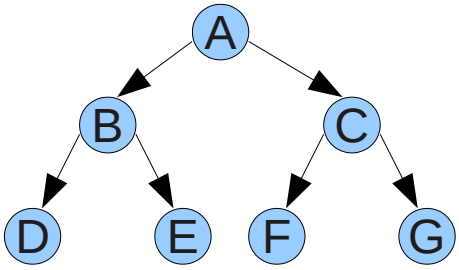
12. About a **list** (ADT), which of the following options best describe its characteristics:

a) lists cannot be extended, can be accessed via index, and are very good for random access.
b) lists can be extended, but elements have to be accessed sequentially.
c) lists can be extended, can be accessed via index, and are very good for random access.
d) lists cannot be extended, and elements have to be accessed sequentially.

13. Consider the binary tree in the figure below. What is the result of traversing the tree using **in-order** traversal?

Turn over to pg. 6...

a) A B D E C F G
b) D E B F G C A
c) D B E A F C G
d) A B C D E F G



14. A programmer wants to implement a generic iterative (non-recursive) **in-order** traversal code. He needs to keep track of tree nodes using some sort of data-structure. Which one would be the best for this purpose?

a) a stack that stores pointers to Trees
b) a stack that stores integers
c) a queue that stores pointers to Trees
d) a vector that stores Tree->next pointers

15. The following code fragment is used to traverse a Tree. Which traverse is this code implementing?

```
...
void Traverse(Tree *T) {
  if (T == NULL) { return; }
  Traverse(T->Left());
  Traverse(T->Right());
  printf("%c ", T->RootData());
}
...
```

a) Pre-order
b) In-order
c) Post-order
d) None of the above

+ + + + + + + + + +

Turn over to pg. ...