CCS0015

Data Structures and Algorithms

FA 1 FROM BILL

The following are examples of non-linear list, except:

- **Stack**
- Tree
- Graph
- Hash Table

It is a step by step procedure to solve a particular function

- Data
- Program
- **Algorithm**
- Process

cout << list.back() << endl; The back member function returns a reference to the last element in the list.

- **True**
- False

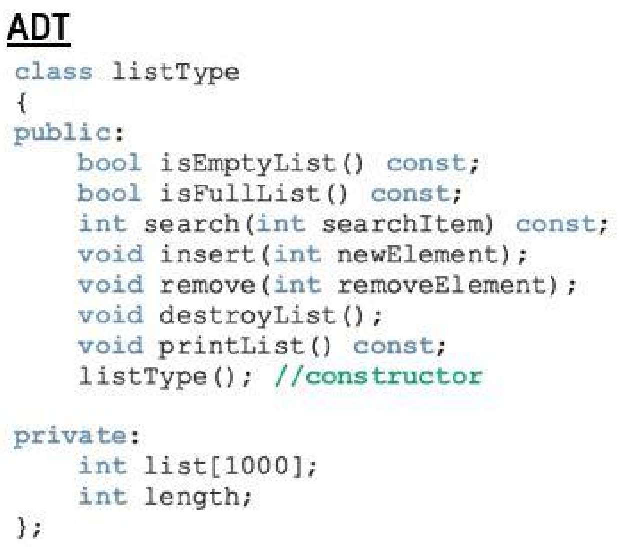
list [int] myList; // is a valid declaration

- True
- **False**

An ADT is a relational model, together with the various operations defined on the model

- True
- **False**

Refer to ADT: What is the data type of the elements that the ADT may accommodate?



ADT

```
class listType
{
public:
    bool isEmptyList() const;
    bool isFullList() const;
    int search(int searchItem) const;
    void insert(int newElement);
    void remove(int removeElement);
    void destroyList();
    void printList() const;
    listType(); //constructor

private:
    int list[1000];
    int length;
};
```

- char
- string
- **int**
- float

char &ch is valid pointer declaration

- True
- **False**

TRUE/FALSE: Each node in a linked list contains one or more members that represent data and a pointer which can point to another node.

- **True**
- False

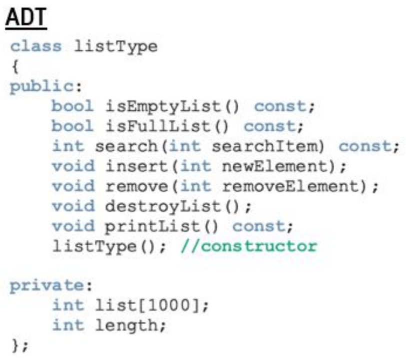
A linked list is variable in size.

- **True**
- False

Graphs & Trees are non-linear lists.

- **True**
- False

Refer to ADT: What is the name of the ADT?

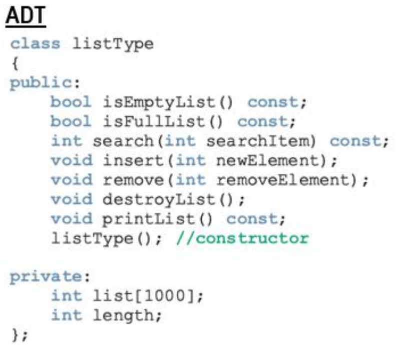


ADT

```
class listType
{
public:
    bool isEmptyList() const;
    bool isFullList() const;
    int search(int searchItem) const;
    void insert(int newElement);
    void remove(int removeElement);
    void destroyList();
    void printList() const;
    listType(); //constructor

private:
    int list[1000];
    int length;
};
```

- **listType**
- Listtype
- ListType
- listtype

TRUE/FALSE: Refer to ADT: The ADT is HOMOGENEOUS.



ADT

```
class listType
{
public:
     bool isEmptyList() const;
     bool isFullList() const;
     int search(int searchItem) const;
     void insert(int newElement);
     void remove(int removeElement);
     void destroyList();
     void printList() const;
     listType(); //constructor

private:
     int list[1000];
     int length;
};
```

- **True**
- False

Heap is a special area of memory that is reserved for dynamically allocated variables.

- **True**
- False

Given:

```
   int *p;

   int x;

    x = 10;
```

p = x is a valid statement.

- True
- **False**

True/False: The & in front of an ordinary variable produces the address of that variable; that is, it produces a pointer that points to the variable.

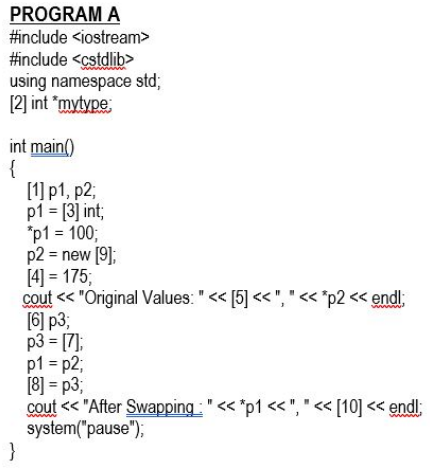
- **True**
- False

The address of operator is a unary operator that returns the address of its operand.

- @
- ->
- **&**
- ~

Consider Program A: Blank #[10]:

Supply the code that is appropriate in the blank.



PROGRAM A

```
#include <iostream>
#include <cstdlib>
using namespace std;
[2] int *mytype;

int main()
{
   [1] p1, p2;
   p1 = [3] int;
   *p1 = 100;
   p2 = new [9];
   [4] = 175;
  cout << "Original Values: " << [5] << ", " << *p2 << endl;
   [6] p3;
   p3 = [7];
   p1 = p2;
   [8] = p3;
   cout << "After Swapping : " << *p1 << ", " << [10] << endl;
   system("pause");
}
```

- ***p2**
- *p1
- p1
- p2

Any new dynamic variable created by a program consumes some of the memory in the freespace

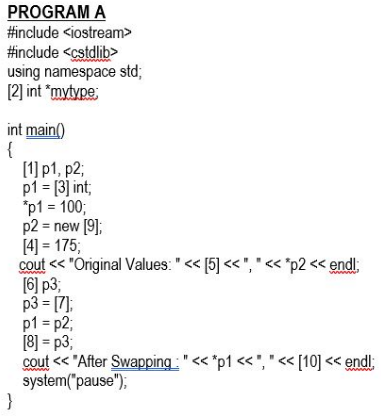
- **True**
- **False**

The new operator creates a new dynamic variable of a specified type and returns a pointer that points to this new variable.

- **True**
- **False**

Consider Program A: Blank #[3]:

Supply the code that is appropriate in the blank.



PROGRAM A

```
#include <iostream>
#include <cstdlib>
using namespace std;
[2] int *mytype;

int main()
{
   [1] p1, p2;
   p1 = [3] int;
   *p1 = 100;
   p2 = new [9];
   [4] = 175;
  cout << "Original Values: " << [5] << ", " << *p2 << endl;
   [6] p3;
   p3 = [7];
   p1 = p2;
   [8] = p3;
   cout << "After Swapping : " << *p1 << ", " << [10] << endl;
   system("pause");
}
```

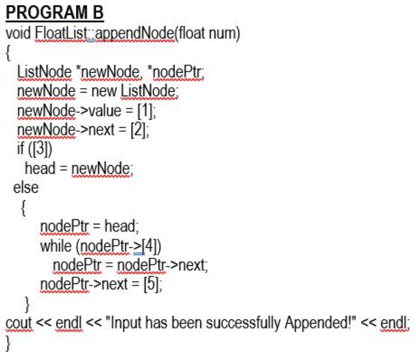
- new()
- New
- New()
- **new**

You need an array to represent each node in a linked list.

- True
- **False**

Consider Program B: Blank #[4]:

Supply the code that is appropriate in the blank.



PROGRAM B

```
void FloatList::appendNode(float num)
{
  ListNode *newNode, *nodePtr;
  newNode = new ListNode;
  newNode->value = [1];
  newNode->next = [2];
  if ([3])
    head = newNode;
 else
  {
      nodePtr = head;
      while (nodePtr->[4])
         nodePtr = nodePtr->next;
      nodePtr->next = [5];
   }
cout << endl << "Input has been successfully Appended!" << endl;
}
```

- Next
- value
- **next**
- Value

Insertion and deletion of nodes is quicker with linked lists than with vectors.

- **True**
- False

A doubly linked list has both a next and previous node pointers.

- **True**
- False

The STL list function push_back is equivalent to inserting a node in a list.

- **True**
- **False**

cout << list.front() << endl;
front returns a reference to the last element of the list.

- True
- **False**

cout << list.back() << endl;

The back member function returns a reference to the last element in the list.

- **True**
- False

A linked list is variable in size.

- **True**
- False

Character is a primitive data structure.

- **True**
- False

Array is an example of homogeneous data structures.

- **True**
- False

The new operator eliminates a dynamic variable and returns the memory that the dynamic variable occupied to the freestore manager so that the memory can be reused.

- **True**
- False

Abstraction is providing only essential information outside the world.

- **True**
- False

You can assign a name definition and then use the type name to declare variables using typedef keyword.

- **True**
- False

Pointer Variable is the content that is stored in the memory address.

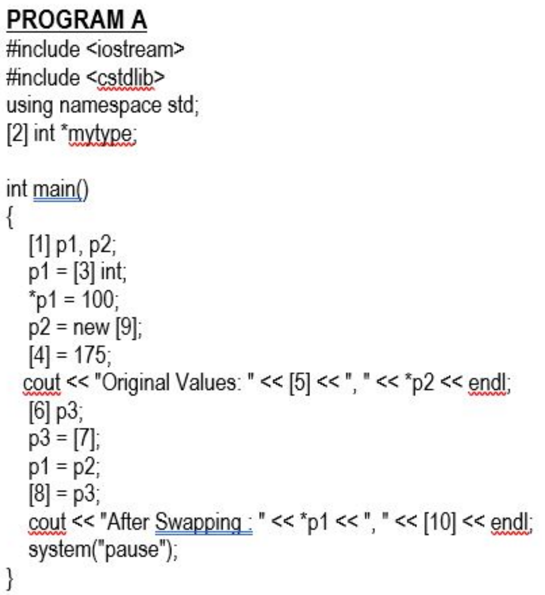
- **True**
- **False**

It is a memory address of a variable.
Variable

- Address of a Variable
- **Pointer**
- Pointer Variable

Consider Program A: Blank #[2]:

Supply the code that is appropriate in the blank.



PROGRAM A

```
#include <iostream>
#include <cstdlib>
using namespace std;
[2] int *mytype;

int main()
{
   [1] p1, p2;
   p1 = [3] int;
   *p1 = 100;
   p2 = new [9];
   [4] = 175;
  cout << "Original Values: " << [5] << ", " << *p2 << endl;
   [6] p3;
   p3 = [7];
   p1 = p2;
   [8] = p3;
   cout << "After Swapping : " << *p1 << ", " << [10] << endl;
   system("pause");
}
```

- type def
- Type def
- **typedef**
- Typedef

If there is insufficient available memory to create the new variable, then new returned a special value named NIL.

- True
- **False**

Stack is a special area of memory that is reserved for dynamically allocated variables.

- True
- **False**

Pointers can be used as parameter to accept an array from outside.

- **True**
- False

A linked list is a series of connected nodes, where each node is a data structure.

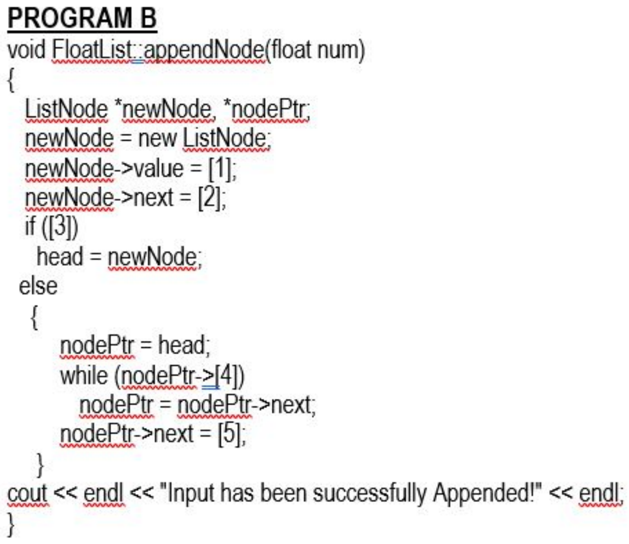
- **True**
- False

The algorithm for displaying the elements of a linked list would require traversal.

- **True**
- False

Consider Program B: Blank #[1]:

Supply the code that is appropriate in the blank.



PROGRAM B

```
void FloatList::appendNode(float num)
{
  ListNode *newNode, *nodePtr;
  newNode = new ListNode;
  newNode->value = [1];
  newNode->next = [2];
  if ([3])
    head = newNode;
  else
  {
      nodePtr = head;
      while (nodePtr->[4])
        nodePtr = nodePtr->next;
      nodePtr->next = [5];
  }
cout << endl << "Input has been successfully Appended!" << endl;
}
```

- Num
- **num**
- newNode
- *newNode

Heap is a special area of memory that is reserved for dynamically allocated variables.

- **True**
- False

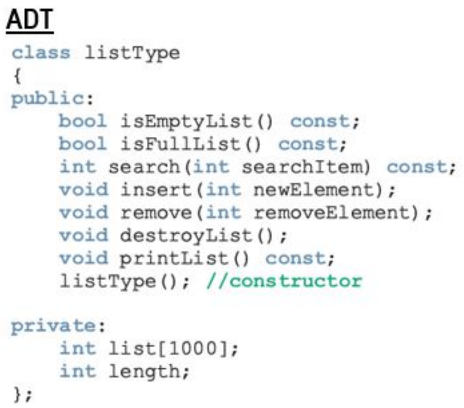
TRUE/FALSE: Program = Algorithms + Data Structures

- False
- **True**

It is is a step by step procedure to solve a particular function.

- Program
- Process
- **Algorithm**
- Data

TRUE/FALSE: Refer to ADT: The ADT is DYNAMIC.



ADT

```
class listType
{
public:
    bool isEmptyList() const;
    bool isFullList() const;
    int search(int searchItem) const;
    void insert(int newElement);
    void remove(int removeElement);
    void destroyList();
    void printList() const;
    listType(); //constructor

private:
    int list[1000];
    int length;
};
```

- True

- **False**

Dynamic Array Is an array whose size is not specified when you write the program, but is determined while the program is running.

- **True**
- False

TRUE/FALSE: There is a name associated with a pointer data type in C++.

- **True**
- **False**

char &ch is valid pointer declaration

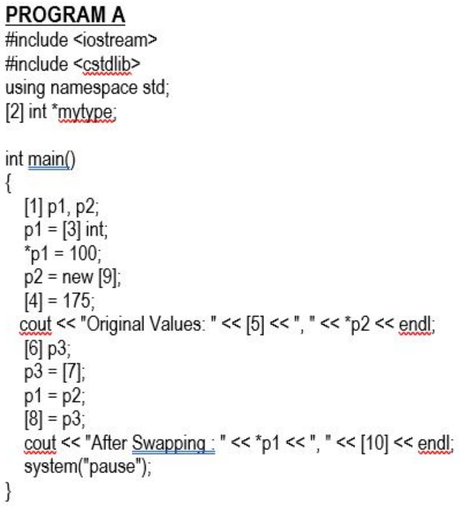
- True
- **False**

Typedef is declared within the main program.

- True
- **False**

Consider Program A: Blank #[8]:

Supply the code that is appropriate in the blank.

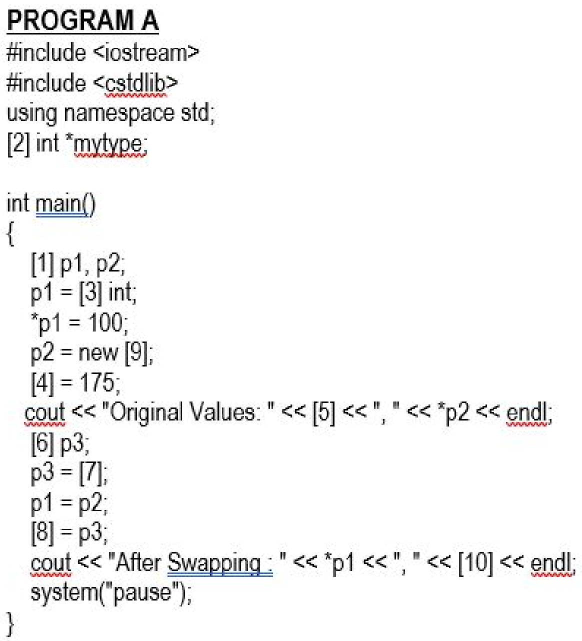


```
PROGRAM A
#include <iostream>
#include <cstdlib>
using namespace std;
[2] int *mytype;

int main()
{
   [1] p1, p2;
   p1 = [3] int;
   *p1 = 100;
   p2 = new [9];
   [4] = 175;
  cout << "Original Values: " << [5] << ", " << *p2 << endl;
   [6] p3;
   p3 = [7];
   p1 = p2;
   [8] = p3;
   cout << "After Swapping : " << *p1 << ", " << [10] << endl;
   system("pause");
}
```

- **p1**
- *p2
- **p2**
- ***p1**

Consider Program A: Blank #[9]:

Supply the code that is appropriate in the blank.



```
PROGRAM A
#include <iostream>
#include <cstdlib>
using namespace std;
[2] int *mytype;

int main()
{
   [1] p1, p2;
   p1 = [3] int;
   *p1 = 100;
   p2 = new [9];
   [4] = 175;
  cout << "Original Values: " << [5] << ", " << *p2 << endl;
   [6] p3;
   p3 = [7];
   p1 = p2;
   [8] = p3;
   cout << "After Swapping : " << *p1 << ", " << [10] << endl;
   system("pause");
}
```

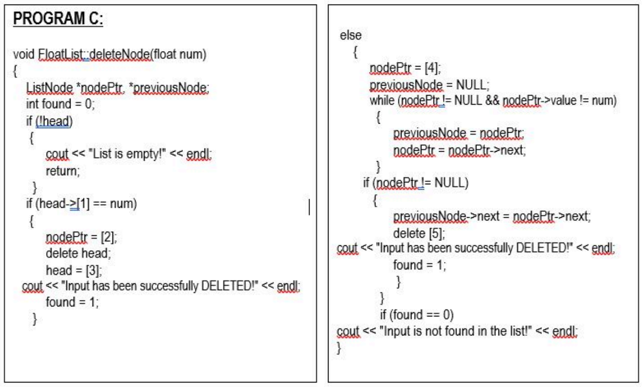
- **double**
- char
- **int**
- **float**

In inserting a node, finding its proper location and following a certain order is necessary.

- **True**
- False

Consider Program C: Blank #[5]:

Supply the code that is appropriate in the blank.



```
PROGRAM C:

void FloatList::deleteNode(float num)
{
   ListNode *nodePtr, *previousNode;
   int found = 0;
   if (!head)
    {
       cout << "List is empty!" << endl;
       return;
    }
   if (head->[1] == num)
    {
       nodePtr = [2];
       delete head;
       head = [3];
  cout << "Input has been successfully DELETED!" << endl;
       found = 1;
    }
```

```
else
   {
      nodePtr = [4];
      previousNode = NULL;
      while (nodePtr != NULL && nodePtr->value != num)
        {
           previousNode = nodePtr;
           nodePtr = nodePtr->next;
        }
     if (nodePtr != NULL)
       {
           previousNode->next = nodePtr->next;
           delete [5];
cout << "Input has been successfully DELETED!" << endl;
           found = 1;
            }
        }
        if (found == 0)
cout << "Input is not found in the list!" << endl;
}
```

- **nodePtr**
- nodePtr->next
- head
- previousNode

list [int] myList; // is a valid declaration

- True
- **False**

STL lists are also efficient at adding elements at their back because they have a built-in pointer to the last element in the list.

- **True**
- False

The following are examples of primitive data structures, except.

- Character
- Integer
- **Array**
- Float

True/False: ADTs support abstraction, encapsulation, and information binding.

- **True**
- False

Given:

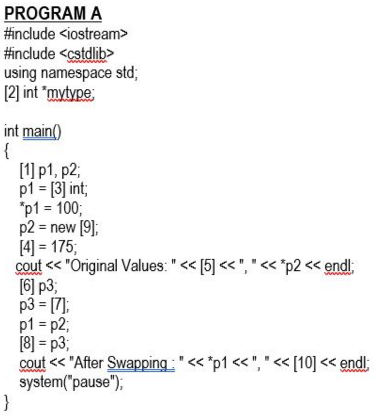
int *p, *q;
int x, y;
x = 2;
y = 7;
p = &y;
q = p;

The statement cout << *q will have an output of memory address.

- True
- **False**

Consider Program A: Blank #[7]:

Supply the code that is appropriate in the blank.

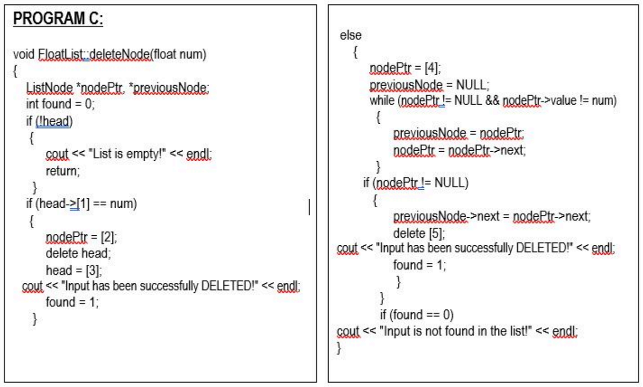


```
PROGRAM A
#include <iostream>
#include <cstdlib>
using namespace std;
[2] int *mytype;

int main()
{
    [1] p1, p2;
    p1 = [3] int;
    *p1 = 100;
    p2 = new [9];
    [4] = 175;
   cout << "Original Values: " << [5] << ", " << *p2 << endl;
    [6] p3;
    p3 = [7];
    p1 = p2;
    [8] = p3;
    cout << "After Swapping : " << *p1 << ", " << [10] << endl;
    system("pause");
}
```

- *p1
- *p2
- **p2**
- p1

Consider Program C: Blank #[4]:

Supply the code that is appropriate in the blank.



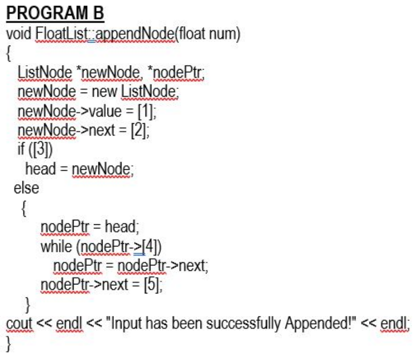
```
PROGRAM C:

void FloatList::deleteNode(float num)
{
   ListNode *nodePtr, *previousNode;
   int found = 0;
   if (!head)
    {
       cout << "List is empty!" << endl;
       return;
     }
   if (head->[1] == num)
    {
       nodePtr = [2];
       delete head;
       head = [3];
  cout << "Input has been successfully DELETED!" << endl;
       found = 1;
     }
```

```
else
   {
      nodePtr = [4];
      previousNode = NULL;
      while (nodePtr != NULL && nodePtr->value != num)
       {
          previousNode = nodePtr;
          nodePtr = nodePtr->next;
       }
    if (nodePtr != NULL)
     {
          previousNode->next = nodePtr->next;
          delete [5];
cout << "Input has been successfully DELETED!" << endl;
          found = 1;
           }
       }
       if (found == 0)
cout << "Input is not found in the list!" << endl;
}
```

- **reviousNode**
- nodePtr->next
- head->next
- **head**

Consider Program B: Blank #[4]:

Supply the code that is appropriate in the blank.



```
PROGRAM B
void FloatList::appendNode(float num)
{
   ListNode *newNode, *nodePtr;
   newNode = new ListNode;
   newNode->value = [1];
   newNode->next = [2];
   if ([3])
     head = newNode;
  else
   {
      nodePtr = head;
      while (nodePtr->[4])
         nodePtr = nodePtr->next;
      nodePtr->next = [5];
    }
cout << endl << "Input has been successfully Appended!" << endl;
}
```

- head
- nodePtr->next
- nodePtr
- **newNode**

Use of template will make the ADT flexible in terms of accepting values of different data types.

- **True**
- False

The list container, found in the Standard Template Library, is a template version of a doubly linked list.

- **True**
- False

A circular linked list has 2 node pointers.

- True
- **False**

It is is representation of the logical relationship existing between individual elements of data.

- Algorithm
- Data
- **Data Structure**

- Program

A Tree is unordered lists which use a ‘hash function’ to insert and search.

- True
- **False**

An ADT is a relational model, together with the various operations defined on the model.

- True
- **False**

Given: int *q; * q = 10; The statement cout << *q will have an output of 10.

- **True**
- False

In the declaration, int *p, q, only p is a pointer variable.

- **True**
- False

You can assign a name, definition, and then use the type name to declare variables using typedef keyword.

- **True**
- False

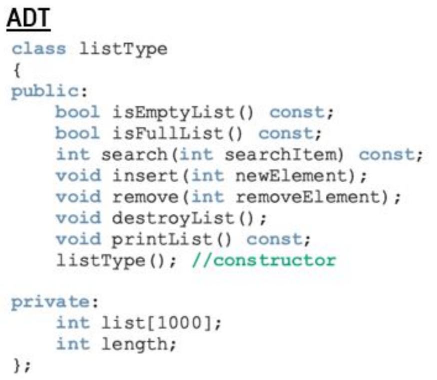
list.unique();
unique removes any element that has the same value as the element

- **True**
- **False**

Pointer is a non-primitive data structure.

- **True**
- **False**

Refer to ADT, How many operations can the ADT do?

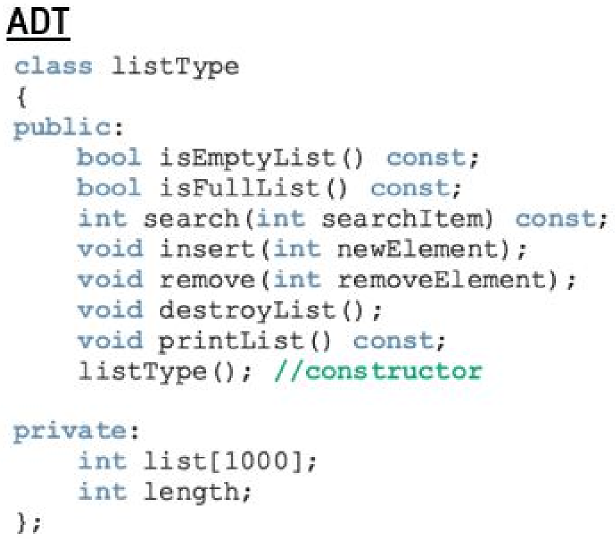


ADT

```
class listType
{
public:
    bool isEmptyList() const;
    bool isFullList() const;
    int search(int searchItem) const;
    void insert(int newElement);
    void remove(int removeElement);
    void destroyList();
    void printList() const;
    listType(); //constructor

private:
    int list[1000];
    int length;
};
```

- **7**
- 5
- 6
- 8

Refer to ADT: What is the data type of the elements that the ADT may accommodate?



ADT

```
class listType
{
public:
    bool isEmptyList() const;
    bool isFullList() const;
    int search(int searchItem) const;
    void insert(int newElement);
    void remove(int removeElement);
    void destroyList();
    void printList() const;
    listType(); //constructor

private:
    int list[1000];
    int length;
};
```

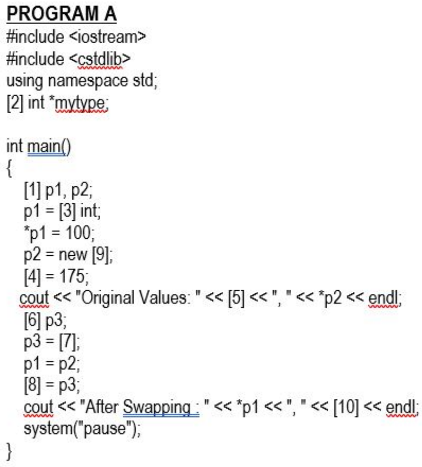
- float
- string
- **int**
- char

TRUE/FALSE: The * operator in front of a pointer variable produces the variable to which it points. When used this way, the * operator is called the dereferencing operator.

- False
- **True**

Consider Program A: Blank #[4]:

Supply the code that is appropriate in the blank.



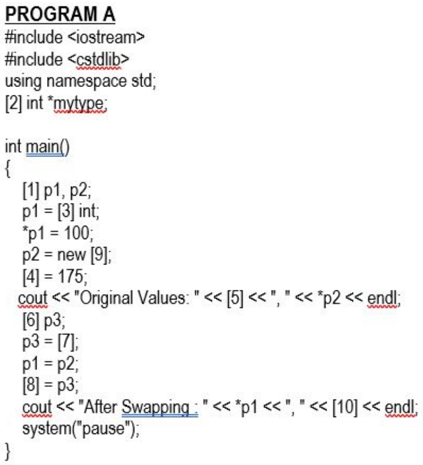
PROGRAM A

```
#include <iostream>
#include <cstdlib>
using namespace std;
[2] int *mytype;

int main()
{
   [1] p1, p2;
   p1 = [3] int;
   *p1 = 100;
   p2 = new [9];
   [4] = 175;
  cout << "Original Values: " << [5] << ", " << *p2 << endl;
   [6] p3;
   p3 = [7];
   p1 = p2;
   [8] = p3;
   cout << "After Swapping : " << *p1 << ", " << [10] << endl;
   system("pause");
}
```

- @p2
- p2
- *P2
- ***p2**

Consider Program A: Blank #[6]:

Supply the code that is appropriate in the blank.



```
PROGRAM A
#include <iostream>
#include <cstdlib>
using namespace std;
[2] int *mytype;

int main()
{
   [1] p1, p2;
   p1 = [3] int;
   *p1 = 100;
   p2 = new [9];
   [4] = 175;
  cout << "Original Values: " << [5] << ", " << *p2 << endl;
   [6] p3;
   p3 = [7];
   p1 = p2;
   [8] = p3;
   cout << "After Swapping : " << *p1 << ", " << [10] << endl;
   system("pause");
}
```

- int
- my type
- INT
- **mytype**

Traversing means visiting each node of a linked list one by one.

- **True**
- False

A linked list can grow or shrink in size as the program runs.

- **True**
- False

Queue adds anywhere, removes the highest priority.

- True
- **False**

Which of the following is the proper way of declaring a pointer variable of type char?

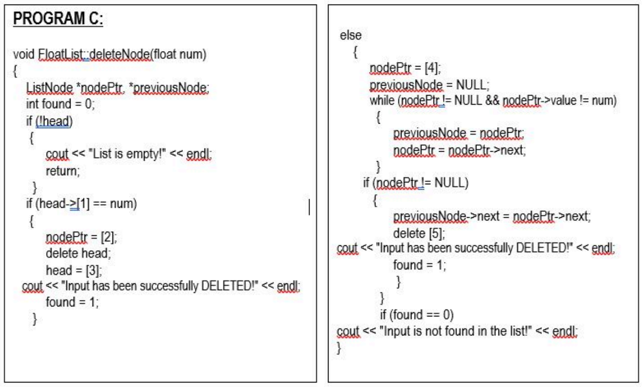
- **char *p, *q;**
- .*char p, q;
- &char p, q;
- .char &p, &q;

Remove operator eliminates a dynamic variable and returns the memory that the dynamic variable occupied to the freestore manager so that the memory can be reused.

- **True**
- False

Consider Program C: Blank #[2]:

Supply the code that is appropriate in the blank.



```
PROGRAM C:

void FloatList::deleteNode(float num)
{
   ListNode *nodePtr, *previousNode;
   int found = 0;
   if (!head)
    {
       cout << "List is empty!" << endl;
       return;
    }
   if (head->[1] == num)
    {
       nodePtr = [2];
       delete head;
       head = [3];
  cout << "Input has been successfully DELETED!" << endl;
       found = 1;
    }
```

```
else
   {
       nodePtr = [4];
       previousNode = NULL;
       while (nodePtr != NULL && nodePtr->value != num)
        {
           previousNode = nodePtr;
           nodePtr = nodePtr->next;
        }
     if (nodePtr != NULL)
       {
           previousNode->next = nodePtr->next;
           delete [5];
cout << "Input has been successfully DELETED!" << endl;
           found = 1;
            }
         }
         if (found == 0)
cout << "Input is not found in the list!" << endl;
}
```

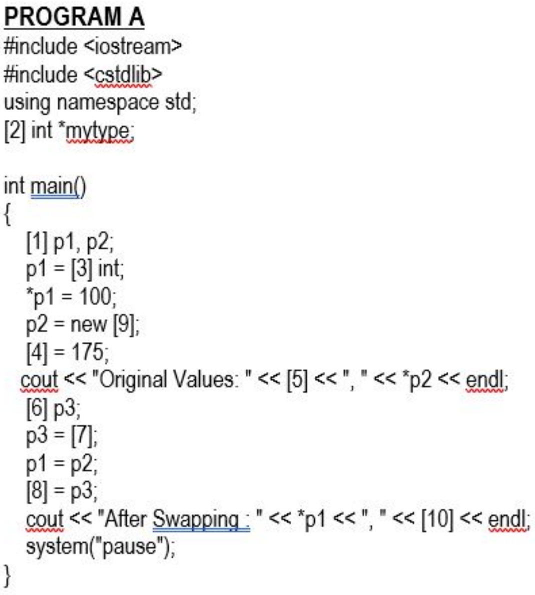
- head
- **nodePtr->next**
- nodePtr
- head->next

If pointer variable pointing to the dynamic variable that was destroyed and becomes undefined is called dangling pointers.

- **True**
- False

Consider Program A: Blank #[1]:

Supply the code that is appropriate in the blank.

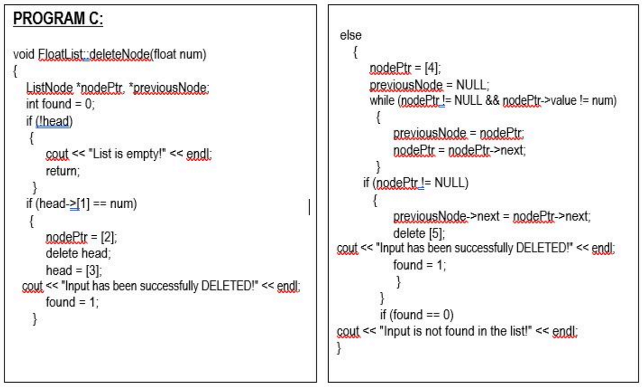


```
PROGRAM A
#include <iostream>
#include <cstdlib>
using namespace std;
[2] int *mytype;

int main()
{
   [1] p1, p2;
   p1 = [3] int;
   *p1 = 100;
   p2 = new [9];
   [4] = 175;
  cout << "Original Values: " << [5] << ", " << *p2 << endl;
   [6] p3;
   p3 = [7];
   p1 = p2;
   [8] = p3;
   cout << "After Swapping : " << *p1 << ", " << [10] << endl;
   system("pause");
}
```

- integer
- my type
- **int**
- mytype

Consider Program C: Blank #[1]:

Supply the code that is appropriate in the blank.



```
PROGRAM C:

void FloatList::deleteNode(float num)
{
   ListNode *nodePtr, *previousNode;
   int found = 0;
   if (!head)
    {
       cout << "List is empty!" << endl;
       return;
    }
   if (head->[1] == num)
    {
       nodePtr = [2];
       delete head;
       head = [3];
  cout << "Input has been successfully DELETED!" << endl;
       found = 1;
    }
```

```
else
   {
      nodePtr = [4];
      previousNode = NULL;
      while (nodePtr != NULL && nodePtr->value != num)
        {
           previousNode = nodePtr;
           nodePtr = nodePtr->next;
        }
     if (nodePtr != NULL)
        {
           previousNode->next = nodePtr->next;
           delete [5];
cout << "Input has been successfully DELETED!" << endl;
           found = 1;
            }
        }
        if (found == 0)
cout << "Input is not found in the list!" << endl;
}
```

- nodePtr
- next
- value
- **num**

Appending a node means adding a node at the start of the list.

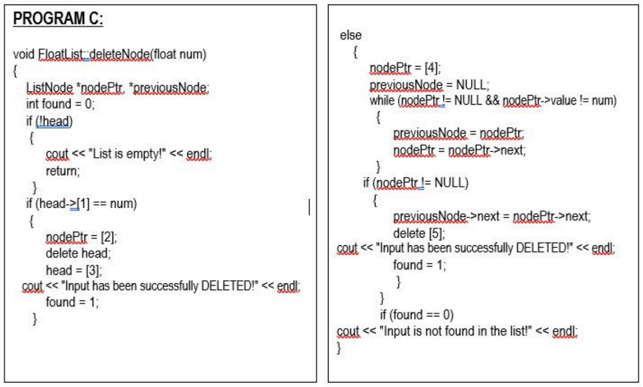
- True
- **False**

When you apply delete to a pointer variable, the dynamic variable to which it is pointing is destroyed.

- **True**
- False

Consider Program C: Blank #[3]:

Supply the code that is appropriate in the blank.



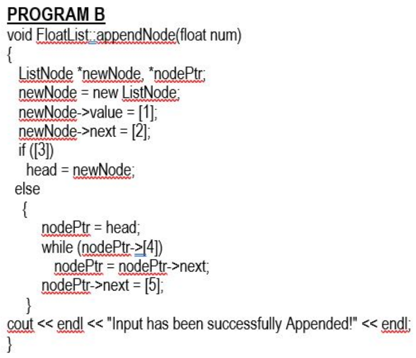
```
PROGRAM C:

void FloatList::deleteNode(float num)
{
   ListNode *nodePtr, *previousNode;
   int found = 0;
   if (!head)
    {
       cout << "List is empty!" << endl;
       return;
    }
   if (head->[1] == num)
    {
       nodePtr = [2];
       delete head;
       head = [3];
  cout << "Input has been successfully DELETED!" << endl;
       found = 1;
    }
```

```
else
   {
      nodePtr = [4];
      previousNode = NULL;
      while (nodePtr != NULL && nodePtr->value != num)
        {
           previousNode = nodePtr;
           nodePtr = nodePtr->next;
        }
     if (nodePtr != NULL)
        {
           previousNode->next = nodePtr->next;
           delete [5];
cout << "Input has been successfully DELETED!" << endl;
           found = 1;
            }
        }
        if (found == 0)
cout << "Input is not found in the list!" << endl;
}
```

- nodePtr->next
- nodePtr
- **head->next**
- previousNode

Consider Program B: Blank #[2]:

Supply the code that is appropriate in the blank.

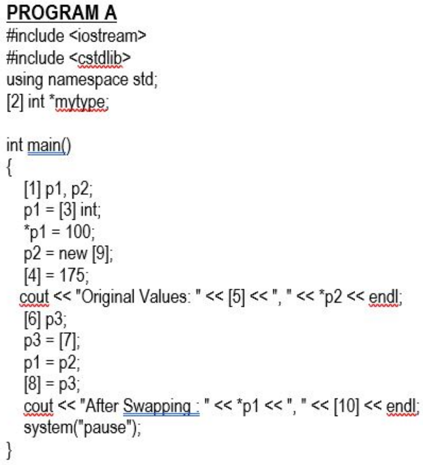


```
PROGRAM B
void FloatList::appendNode(float num)
{
   ListNode *newNode, *nodePtr;
   newNode = new ListNode;
   newNode->value = [1];
   newNode->next = [2];
   if ([3])
     head = newNode;
  else
   {
       nodePtr = head;
       while (nodePtr->[4])
          nodePtr = nodePtr->next;
       nodePtr->next = [5];
    }
cout << endl << "Input has been successfully Appended!" << endl;
}
```

- -1
- **NULL**
- 0
- null

Consider Program A: Blank #[5]:

Supply the code that is appropriate in the blank.



```
PROGRAM A
#include <iostream>
#include <cstdlib>
using namespace std;
[2] int *mytype;

int main()
{
   [1] p1, p2;
   p1 = [3] int;
   *p1 = 100;
   p2 = new [9];
   [4] = 175;
  cout << "Original Values: " << [5] << ", " << *p2 << endl;
   [6] p3;
   p3 = [7];
   p1 = p2;
   [8] = p3;
   cout << "After Swapping : " << *p1 << ", " << [10] << endl;
   system("pause");
}
```

- P1
- ***p1**
- *P1
- p1

FA1 - FROM ELLA

Primitive data structures are derived from non-primitive data structures.

- True
- **False**

True/False: Abstract Data Type (ADT) stores data and allow various operations on the data to access and change it.

- **True**
- False

FA1 - FROM DISCORD

The following are characteristics of non-primitive data types, except:

- **The non-primitive data structures emphasize on structuring of a group of homogeneous data items only.**
- The design of an effective data structure must take operations to be performed in that data structure.
- They are derived from primitive data types.
- They are more sophisticated data types.

This node is responsible to handle the data that will be added to the linked list.

- previous
- **newNode**
- head
- nodePtr

FA 2 FROM BILL

Below are characteristics of a dynamic stack, except:

- Can be implemented with a linked list
- **Can be implemented with a dynamic array**
- Shrink in size as needed
- Grow in size as needed

In a DYNAMIC STACK, the node that was POPPED is deleted.z

- **True**
- False

What happens to the value of the TOP during a PUSH operation in a STATIC STACK?

- decrements by 1
- becomes 0
- **increments by 1**
- becomes -1

In a DYNAMIC STACK, the node that was POPPED is deleted.

- **True**
- False

The following are 3 possible containers that can be used in implementing the STL Stack, except:

- **array**
- list
- vector
- deque

A static stack is implemented using arrays.

- **True**
- **False**

Below is a valid declaration of a dynamic stack implemented as a vector:

stack< int > iStack

- **True**
- **False**

Pop function will always retrieve the top.

- **True**
- False

The STL function top returns a reference to the element at the top of the stack.

- **True**
- False

The STL stack container may be implemented as a vector, a list, or a deque.

- **True**
- False

In a multi-user system, a queue is used to hold print jobs submitted by users , while the printer services those jobs one at a time.

- **True**
- False

Given:
Enqueue(1)
Enqueue(2)
Enqueue(3)
The value stored in front of the queue is 3.

- True
- **False**

A queue, however, provides access to its elements in first-in, first-out (FIFO) order.

- **True**
- False

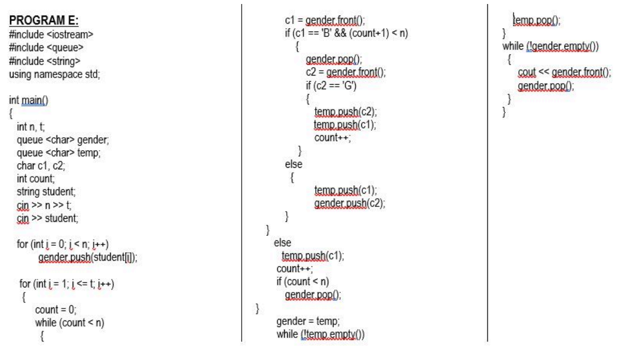
TRUE/FALSE: In a static queue, elements are being deleted when you dequeue.

- **True**
- **False**

As each new item is added to the queue, a new node is added to the rear of the list, and the rear pointer is updated to point to the new node.

- **True**
- False

Consider Program E: What is the **name of the queue that initially holds the input** of the user?



PROGRAM E:

```
#include <iostream>
#include <queue>
#include <string>
using namespace std;

int main()
{
  int n, t;
  queue <char> gender;
  queue <char> temp;
  char c1, c2;
  int count;
  string student;
  cin >> n >> t;
  cin >> student;

  for (int i = 0; i < n; i++)
        gender.push(student[i]);

   for (int i = 1; i <= t; i++)
   {
      count = 0;
      while (count < n)
       {
          c1 = gender.front();
          if (c1 == 'B' && (count+1) < n)
            {
               gender.pop();
               c2 = gender.front();
               if (c2 == 'G')
               {
                 temp.push(c2);
                 temp.push(c1);
                 count++;
              }
          else
           {
                 temp.push(c1);
                 gender.push(c2);
          }
      }
        else
          temp.push(c1);
         count++;
         if (count < n)
           gender.pop();
   }
         gender = temp;
         while (!temp.empty())
            temp.pop();
         }
         while (!gender.empty())
          {
            cout << gender.front();
            gender.pop();
          }
         }
```

- temp
- **gender**
- c1
- student

Programs that use the deque ADT must include the deque header file.
- **True**
- False

C++ has an existing queue container.
- **True**
- False

A deque is similar to an array, but allows efficient access to values at both the front and the rear.
- **True**
- **False**

Pop is a function in deque STL that removes the first element of the deque and discards it.
- True
- **False**

During a POP operation in a STATIC STACK, the elements are being moved one step up.
- True
- **False**

In a DYNAMIC STACK the pointer TOP stays at the HEAD after a PUSH operation.
- **True**
- **False**

What is the value of TOP when the STATIC STACK is FULL?
- <= to the stack size
- < to the (stack size-1)
- < to the stack
- **= to the (stack size-1)**

The initial value of index top in the static implementation of a stack is 0.
- True
- **False**

The STL empty function will yield a value of true if the stack has elements.
- True
- **False**

Invoking the STL function top will automatically retrieve the element and move the pointer.
- True
- **False**

The two primary queue operations are push and pop.
- True
- **False**

Given:
Enqueue(1)
Enqueue(2)
Enqueue(3)
Dequeue()
The value stored in front of the queue is 1.
- True
- **False**

In a static queue, what are the values of front and rear when a full queue is dequeued?
- **front=-1, rear=stackSize-1;**
- front=stackSize-1; rear=-1;
- **front=stackSize-1; rear=stackSize-1;**
- **front=0; rear=stackSize-1;**

Consider Program E: What is the output of the following inputs:

3 3
BGB

- **GBB**
- BGB
- BBG
- None of the Options

The queue container adapter can be built upon vectors, lists, or deques.
- **True**
- False

The queue ADT is like the the stack ADT: it is actually a container adapter.
- **True**
- False

The following are stack operations except:
- **Clear**
- Push
- Pop

- IsEmpty

The following are 3 possible containers that can be used in implementing the STL Stack, except:

- list
- deque
- vector
- **array**

In a dynamic stack, the pointer TOP is like the HEAD which always point to the first element of the linked list.

- **True**
- False

Below is a valid declaration of a dynamic stack implemented as a list:

stack< int, list<int> > iStack;

- **True**
- False

TRUE/FALSE: To dequeue means to insert an element at the rear of a queue.

- True
- **False**

TRUE/FALSE: The pointer FRONT moves every time a new value is enqueued.

- True
- **False**

A static queue does not need the isFull operation anymore.

- True
- **False**

A static queue is fix in size.

- **True**
- False

Consider Program E: True/False: Assigning of one queue to the other is allowed.

- False
- **True**

The queue version of push always inserts an element at the front of the queue.

- True
- **False**

A deque (pronounced "deck" or "deek") is a double-ended queue.

- **True**
- False

Which of the following is an application of a stack?

- Printer Spooler
- CPU Scheduling
- **Calculator**
- Sending of Network Packets

In a dynamic implementation of stack, the pointer TOP has an initial value of NULL.

- **True**
- False

The STL function push retrieves an element at the top of the stack.

- True
- **False**

A stack container that is used to adapt to different containers, it is often referred to as a container adapter.

- **True**
- False

TRUE/FALSE: The elements in a queue are processed like customers standing in a grocery check-out line: the first customer in line is the first one served.

- **False**
- **True**

A dynamic queue starts as an empty linked list.

- **True**
- False

queue int x;

Is a valid declaration of a queue container in C++.

- True
- **False**

A queue has top and bottom pointers.

- True
- **False**

Consider Program E: What is the output of the following inputs:

5 1
BGGBG

- GBBG
- **GBGGB**
- BGGBG
- GBBGGG

When you perform enqueue in a dynamic queue, both the front and rear pointers move.

- True
- **False**

Front is a function in deque STL that returns a reference to the first element of the deque.

- **True**
- False

Consider Program E: What are the valid characters being processed by the program?

- **B, G**
- G, H
- B, C
- A, B

Using pop function automatically moves the top pointer to the next node without deleting the memory used.

- True
- **False**

Dynamic Stacks can be implemented using linked list.

- **True**
- False

In a dynamic stack, pointer TOP points to a fixed value in the linked list and does not move.

- True
- **False**

In a static stack, the variable stackSize will handle the total capacity of the stack.

- **True**
- False

A dynamic queue makes use of an array for implementation.

- True
- **False**

As each item is dequeued, the node pointed to by the front pointer is deleted, and front is made to point to the next node in the list.

- **True**
- False

FA2 - FROM NICOLE

During a POP operation in a STATIC STACK, the elements are being moved one step up.

- True
- **False**

In a DYNAMIC STACK, the node that was POPPED is deleted.

- **True**
- False

The manner in which a stack behaves?

- FILO
- **LIFO**
- FIFO
- LILO

In a dynamic stack, pointer TOP points to a fixed value in the linked list and does not move.
Group of answer choices

- True
- **False**

In a DYNAMIC STACK the pointer TOP stays at the HEAD after a PUSH operation.

- True
- **False**

In a dynamic implementation of stack, the pointer TOP has an initial value of NULL.

- **True**
- False

Using pop function automatically moves the top pointer to the next node without deleting the memory used.

- True
- **False**

The STL empty function will yield a value of true if the stack has elements.

- True
- **False**

The STL function push retrieves an element at the top of the stack.

- True
- **False**

A stack container that is used to adapt to different containers, it is often referred to as a container adapter.

- **True**
- False

Given:
Enqueue(1)
Enqueue(2)
Enqueue(3)
Dequeue()
The value stored in front of the queue is 1.

- True
- **False**

A queue has top and bottom pointers.

- True
- **False**

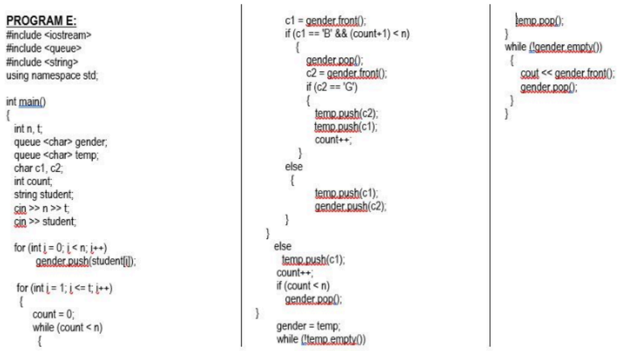
The two primary queue operations are push and pop.

- True
- **False**

Consider Program E: What is the output of the following inputs:

4 1

GGGB



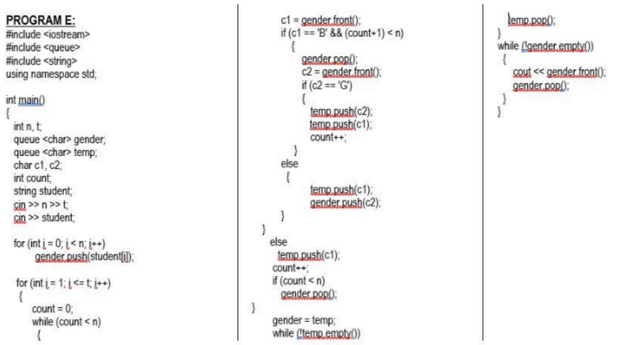
```
PROGRAM E:
#include <iostream>
#include <queue>
#include <string>
using namespace std;

int main()
{
  int n, t;
  queue <char> gender;
  queue <char> temp;
  char c1, c2;
  int count;
  string student;
  cin >> n >> t;
  cin >> student;

  for (int i = 0; i < n; i++)
       gender.push(student[i]);

   for (int i = 1; i <= t; i++)
   {
      count = 0;
      while (count < n)
       {
              c1 = gender.front();
              if (c1 == 'B' && (count+1) < n)
                {
                   gender.pop();
                   c2 = gender.front();
                   if (c2 == 'G')
                   {
                     temp.push(c2);
                     temp.push(c1);
                     count++;
                  }
              else
               {
                     temp.push(c1);
                     gender.push(c2);
              }
          }
        else
          temp.push(c1);
         count++;
         if (count < n)
           gender.pop();
     }
         gender = temp;
         while (!temp.empty())
            temp.pop();
         }
         while (!gender.empty())
          {
            cout << gender.front();
            gender.pop();
          }
         }
```

- **GGGB**
- BGGG
- GGG
- GGBG

As each new item is added to the queue, a new node is added to the rear of the list, and the rear pointer is updated to point to the new node.

- **True**
- False

Consider Program E: What are the valid characters being processed by the program?



```
PROGRAM E:
#include <iostream>
#include <queue>
#include <string>
using namespace std;

int main()
{
  int n, t;
  queue <char> gender;
  queue <char> temp;
  char c1, c2;
  int count;
  string student;
  cin >> n >> t;
  cin >> student;

  for (int i = 0; i < n; i++)
       gender.push(student[i]);

   for (int i = 1; i <= t; i++)
   {
      count = 0;
      while (count < n)
       {
              c1 = gender.front();
              if (c1 == 'B' && (count+1) < n)
                {
                   gender.pop();
                   c2 = gender.front();
                   if (c2 == 'G')
                   {
                     temp.push(c2);
                     temp.push(c1);
                     count++;
                  }
              else
               {
                     temp.push(c1);
                     gender.push(c2);
              }
          }
        else
          temp.push(c1);
         count++;
         if (count < n)
           gender.pop();
     }
         gender = temp;
         while (!temp.empty())
            temp.pop();
         }
         while (!gender.empty())
          {
            cout << gender.front();
            gender.pop();
          }
         }
```

- **B, G**
- B, C
- A, B
- G, H

Pop is a function in deque STL that removes the first element of the deque and discards it.

- True
- **False**

Front is a function in deque STL that returns a reference to the first element of the deque.

- **True**
- False

The queue container adapter can be built upon vectors, lists, or deques.

- **True**
- False

The queue ADT is like the the stack ADT: it is actually a container adapter.

- **True**
- False

Which of the following is an application of a stack?

Calculator

CPU Scheduling

Printer Spooler

Sending of Network Packets

In a dynamic stack, the pointer TOP is like the HEAD which always point to the first element of the linked list.

Group of answer choices

True

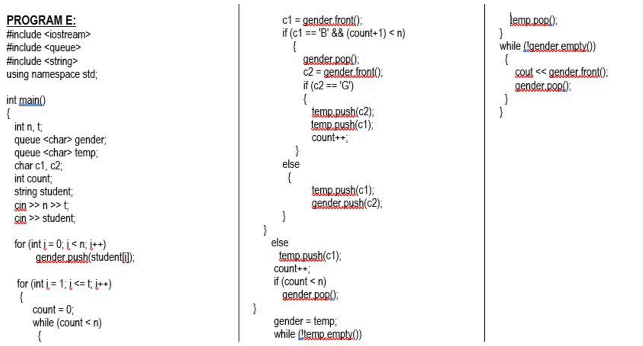
False

TRUE/FALSE: The elements in a queue are processed like customers standing in a grocery check-out line: the first customer in line is the first one served.

True

False

Consider Program E: What is the data type of the user input?



```
PROGRAM E:
#include <iostream>
#include <queue>
#include <string>
using namespace std;

int main()
{
  int n, t;
  queue <char> gender;
  queue <char> temp;
  char c1, c2;
  int count;
  string student;
  cin >> n >> t;
  cin >> student;

  for (int i = 0; i < n; i++)
        gender.push(student[i]);

   for (int i = 1; i <= t; i++)
   {
      count = 0;
      while (count < n)
       {
          c1 = gender.front();
          if (c1 == 'B' && (count+1) < n)
            {
              gender.pop();
              c2 = gender.front();
              if (c2 == 'G')
              {
                temp.push(c2);
                temp.push(c1);
                count++;
              }
          else
           {
                temp.push(c1);
                gender.push(c2);
          }
       }
        else
         temp.push(c1);
        count++;
        if (count < n)
          gender.pop();
     }
        gender = temp;
        while (!temp.empty())
          temp.pop();
        }
        while (!gender.empty())
         {
           cout << gender.front();
           gender.pop();
         }
        }
```

*char

char

string

int

Consider Program E: What are the valid characters being processed by the program?

B, G

B, C

A, B

G, H

Consider Program E: True/False: Assigning of one queue to the other is allowed.

True

False

Consider Program E: What is the output of the following inputs:

5 2

BGGBG

GGBGB

BBGGG

GGGBB

GBGBG

Consider Program E: What variable holds the length of the string that should be accepted?

count

n

t

input

queue int x;

Is a valid declaration of a queue container in C++.

Group of answer choices

True

False

Invoking the STL function top will automatically retrieve the element and move the pointer.

Group of answer choices

True

False

A dynamic queue starts as an empty linked list.

Group of answer choices

True

False

The queue version of push always inserts an element at the front of the queue.

Group of answer choices

True

False

What is the value of TOP when the STATIC STACK is FULL?

< to the (stack size-1)

< to the stack

= to the (stack size-1)

<= to the stack size

The initial value of index top in the static implementation of a stack is 0.

True
False

Below is a valid declaration of a dynamic stack implemented as a list:

stack<int, list<int> > iStack;

True
False

TRUE/FALSE: To dequeue means to insert an element at the rear of a queue.
True
False

A dynamic queue makes use of an array for implementation.
True
False

By default, the queue container uses list as its base

True
False

In a static stack, the variable stackSize will handle the total capacity of the stack
True
False

Dynamic Stacks can be implemented using linked list.
True
False

The following are stack operations except:

IsEmpty

Pop

Push

Clear

TRUE/FALSE: The pointer FRONT moves every time a new value is enqueued
False
True

A static queue does not need the isFull operation anymore.
True
False

Below is a valid declaration of a dynamic stack implemented as a vector: stack< int > iStack

- True
- **False**

FA 3 - FROM BILL

How many base case are there in the recursive function below?

```
void count(int n)
{
 if (n<0)
      cout << "n must be positive.." << endl;
else if (n>10)
    cout << "done counting..." << endl;
else
  {
    cout << n << endl;
    count (n+1);
   }
}
```

- 0
- 1
- 3
- **2**

When a recursive function directly calls itself, this is known as direct recursion.

- **True**
- False

A recursive function should be designed to stop making recursive calls when it reaches its

- last parameter
- **base case**
- return statement
- closing curly brace

The depth of the recursion is the number of times a recursive call is made until it reaches a base case.

- **True**
- False

The ________ of recursion is the number of times a recursive function calls itself.

- type
- level
- **depth**
- breadth

A recursive function is designed to terminate when it reaches its base case.

- **True**
- False

The function if/else statement in a recursive function controls the repetition.

- **True**
- False

When a function A calls a function B, which in turn calls A, we have

- function call cycling
- direct recursion
- **indirect recursion**
- perfect recursion

Nodes without children are called leaves.

- **True**
- False

The mother node of a tree structure is called the head.

- True
- **False**

A group of disjoint trees is called a paradise.

- True
- **False**

The height of a tree is its depth + 1.

- **True**
- False

Binary tree are called "trees" because they resemble an upside-down tree.

- **True**
- False

In certain types of binary trees, the number of leaves can be greater than the number of nodes.

- True
- **False**

The smallest number of levels that a binary tree with three nodes can have is two.

- **True**
- False

Visiting all nodes of a binary tree in some methodical fashion is known as

- **traversing the tree**
- branching out along the tree
- walking through the tree
- climbing the tree

In a binary search tree, all nodes to the right of a node hold values greater than the node's value.

- **True**
- False

Inorder, preorder, and postorder traversals can be accomplished using

- **recursion**.
- no parameters.
- no pointers.
- no arguments.

Values are commonly stored in a binary search tree so that a node's ________ child holds data that is less than the ________ data, while the node's data is less than the data in the other child.

- right, left child's
- right, node's
- **left, node's**
- left, right child's

Output will be the same if you use inorder, postorder, or preorder traversals to print the values stored in a binary tree.

- True
- **False**

Indirect recursion means that a function calls itself several times.

- True
- **False**

Recursive algorithms tend to be less efficient than iterative algorithms.

- **True**
- False

What will be the depth of the recursion when I invoke the function call count(5)?

```
void count(int n)
 {
   if (n<0)
     cout << "n must be positive.." << endl;
   else if (n>10)
    cout << "done counting..." << endl;
  else
   {
     cout << n << endl;
     count (n+1);
    }
}
```

- 5
- 10
- 4
- **6**

The speed and amount of memory available to modern computers diminishes the performance impact of the overhead of recursion so much that for many applications, this overhead is not noticeable.

- **True**
- False

A recursive function is designed to terminate when it reaches its base case.

- **True**
- False

The base case of a recursive function

- is 0.
- **depends on the problem being solved.**
- is depth / 2.
- is 1 / (depth * 3.1415).

In determining level of the tree, count will start at 1.

- **True**
- **False**

The root is the predecessor of all nodes below it.

- **True**
- **False**

The main difference between a binary tree and a linked list is that

- **nodes in a binary tree have two successors instead of one.**
- recursion is useful on binary trees, but not on linked lists.
- a binary tree can be empty, but a linked list cannot.
-) a linked list can be empty, but a binary tree cannot.

If a node has no successor, the corresponding pointer is set to

- point to its parent node.
- the root of the tree.
- **NULL.**
- a leaf.

A node that has no children is a

- root node
- pure binary node
- **leaf node**
- head node

The shape of a binary search tree is

- always triangular.
- always balanced.
- determined by the programmer.
- **determined by the order in which values are inserted.**

A recursive function that does not correctly handle its base case may

- read the NULL terminator and stop
- **cause an infinite chain for recursive calls**
- return 0 and stop
- return FALSE and stop

The function

```
  int fact(int k)

  {

    return k*fact(k-1);

    if (k==0) return 1;

  }
```

- works for all non-negative values of k, but not for negative numbers.

- computes the factorial on an integer k passed to it as parameter.
- **does not correctly handle its base case.**
- **Falsereturns the value 1 if it is passed a value of 0 for the parameter k.**

Any algorithm that can be coded with recursion can also be coded using a loop.

- **True**
- False

The programmer must ensure that a recursive function does not become

- a prototyped function
- a static function
- a dynamic function
- **trapped in an infinite chain of recursive calls**

A Recursive Call is a function call in which the function being called is the same as the one making the call.

- **True**
- False

Tree is a non-linear data structure.

- **True**
- False

A program keeps track of a binary tree using a pointer to

- **one of its leaves**
- **the node in the tree holding the biggest value**
- **the node in the tree holding the smallest value**
- **none of the above**

Every node in a binary tree can have pointers to its end nodes.

- **its left and right child.**
- binary nodes.
- its left and right parent.

Binary trees are commonly used to organize key values that index database records.

- **True**
- False

Deleting a node from a binary search tree node

- is easiest when the node is the root.
- is hardest when the node has one child.
- **is hardest when the node has two children.**
- is hardest when the node is a leaf.

Binary search trees are commonly used

- in linear data communication processes.
- with arrays of integers.
- **in database applications.**
- none of the choices

In a binary search tree where all the stored values are different, the node holding the largest value cannot have two children.

- **True**
- False

It is a powerful technique that can be used in the place of iteration.

- Repetition
- **Recursion**
- Looping
- Recurrence

A kind of recursion that occurs when function A calls function B, which in turn calls function A is called a Direct Recursion.

- True
- **False**

Suppose that a recursive function with integer parameter n has a base case of 0, and for each non-base case, the function makes a recursive call with argument n+1. If the function is initially called with an actual argument of n = 3, the function call will

- return after a chain of 3 recursive calls.
- **cause an infinite chain of recursive calls.**
- return after a chain of 4 recursive calls.
- return after a chain of 2 recursive calls.

A tree is a collection of nodes and this collection may be empty.

- **True**
- False

The maximum number of children that exists for a node is called as degree of a node.

- **True**
- False

Implementing a binary tree in a class requires a structure for representing the nodes of the binary tree, as well as a pointer to the structure as a class member. This pointer will be set to

- the leftmost child node.
- **the root of the tree.**
- the deepest leaf node.

- the first leaf node.

A binary tree can be created using a structure containing a data value and

- two data nodes.
- **two pointers, one for the left child and one for the right child.**
- Falsea pointer to the last child node.
- a pointer to the first child node.

One method of traversing a binary search tree is postorder traversal.

- **all of the choices**
- inorder traversal.
- preorder traversal.

Binary search trees may be implemented as templates, but any data types used with them should support the _________ operator.

- ==
- **all of the choices**
- >
- <

Recursion can be use to:
compute factorials

- **Both A & B**
- program things that cannot be programmed without recursion
- find the greatest common divisor of 2 integer (GCD)

The solvable known problem in a recursive function is called the inductive step.

- True
- **False**

The immediate predecessor of a node in a tree is called the root.

- True
- **False**

An operation that can be performed on a binary search tree is

- **insertion of new value.**
- removing a value stored in the tree.
- all of the choices
- Falsesearching the tree for the occurrence of a given value.

The case for which the solution is expressed in smaller version of itself.

- Terminal Case
- Base Case
- **Recursive Case**
- Iterate Case

The role of recursive functions in programming is to break complex problems down to a solvable problem.

- **True**
- False

All problems that require repetitions must be implemented using recursion.

- True
- **False**

There exists a binary tree with a hundred nodes, but only one leaf.

- **True**
- False

A child node that has no parent is

- an orphan node
- a rootless node
- **none of the above**
- a leaf node

The height of a binary tree describes how many levels there are in the tree.

- **True**
- False

When a binary tree is used to facilitate a search, it is referred to as a

- **binary search tree.**
- binary ordered deque.
- binary queue.
- sort algorithm.

The solvable problem is known as the base case.

- **True**
- False

A subtree is the collection of some node, together with all its descendants.

- **True**
- False

A binary tree node with no parent is called the

- head pointer
- binary node
- **root node**
- pointer node

A tree with a height of three must have

- three subtrees.
- Falseone root and three nodes with two children each.
- six nodes.
- **three levels.**

The width of a tree is the largest number of nodes at the same level.

- **True**
- False

The preorder method of traversing a binary tree involves processing the root node's data, traversing the left subtree, and then traversing the right subtree.

- **True**
- False

A strong reason to use a binary search tree is aesthetics and program design.

- **to expedite the process of searching large sets of information.**
- it is more flexible than the unary search tree.
- to enhance code readability.

The ________ in a binary tree is analogous to the head pointer in a linked list.

- null pointer
- **root pointer**
- leaf pointer
- binary pointer

The inorder method of traversing a binary tree involves traversing the left subtree, processing the data in the root, and then traversing the right subtree.

- **True**
- False

When an application begins searching a binary search tree, it starts at

- the rightmost child of the root node.
- the middle node, halfway between the root and the longest branch.
- **the root node.**
- the outermost leaf node.

FA 4 - FROM BILL

A cycle is a sequence of vertices v1,v2,. . .vk such that consecutive vertices vi and vi+1 are adjacent
Group of answer choices

True

False

An edge e = (u,v) is a pair of vertices
Group of answer choices

True

False

A complete graph is a graph in which all pairs of vertices are adjacent
Group of answer choices

True

False

When the edges in a graph have no direction, the graph is called undirected
Group of answer choices

True

False

Length of a path is nothing but the total number of vertices included in the path from source to destination node.
Group of answer choices

True

False

Graphs are used in the analysis of electrical circuits, finding the shortest route, project planning, linguistics, genetics, social science
Group of answer choices

True

False

In a undirected graph. elements in the set of edges are ordered pairs.
Group of answer choices

True

False

Simple graph has loops and parallel edges.
Group of answer choices

True

False

A comparison-based sorting algorithm usually take at most n-1 passes to sort the data.
Group of answer choices

True

False

Sorting is used to arrange names and numbers in meaningful ways.
Group of answer choices

True

False

Searching is a process that organizes a collection of data into either ascending or descending order.
Group of answer choices

True

False

Heapify picks the largest child key and compare it to the parent key.
Group of answer choices

True

False

You can perform heap sort even to a binary tree that does not follow the heap property.
Group of answer choices

True

False

In sorting a max heap, the first element that will be transferred to the sorted list is the smallest value.
Group of answer choices

True

False

It will take 3 comparisons to to find the value 7 in the list [1,4,8,7,10,28]?
Group of answer choices

True

False

Binary search uses decrease and conquer technique.

Group of answer choices

True

False

Given an array arr = {5,6,77,88,99} and key = 88 and using Binary Search, it will take 3 comparisons to find the key.
Group of answer choices

True

False

In Binary Search, if search item is less than middle element, restrict the search to the right half of the list
Group of answer choices

True

False

In sequential search, the search stops when:

record with matching key is found
or when search has examined all records without success.
Group of answer choices

True

False

Special member that uniquely identifies the item in the data set is called a primary key
Group of answer choices

True

False

A directed graph is connected if, for any pair of vertices, there is a path between them.
Group of answer choices

True

False

Any edge may be connected to any other, these connections are called vertices.
Group of answer choices

True

False

There are 4 vertices indicated in the set of edges below:

E= { (a,b), (a,c), (a,d), (b,e), (c,d), (c,e), (d,e)}

Group of answer choices

True

False

In Insertion Sort, we find the smallest element from the unsorted sublist and swap it with the element at the beginning of the unsorted data.
Group of answer choices

True

False

The Quick Sort Algorithm consists of 3 steps: Divide, Iteration and Conquer.
Group of answer choices

True

False

In quick sort everything that is less than the pivot element goes to the right.
Group of answer choices

True

False

The basis for dividing the unsorted list in merge sort is defined by the chosen pivot element.
Group of answer choices

True

False

Linear Search is more efficient than Binary Search.
Group of answer choices

True

False

A Linear search algorithm requires data to be ordered.
Group of answer choices

True

False

A graph is suppose to be weighted if its every edge is assigned some value which is greater than or equal to zero.
Group of answer choices

True

False

If e = (u, v) then e is incident on u and v
Group of answer choices

True

False

Insertion sort is a sequence of interleaved insertion sorts based on an increment sequence.
Group of answer choices

True

False

Insertion sort is a simple sorting algorithm that is appropriate for small inputs.
Group of answer choices

True

False

Linear Search can be performed both in numbers and letters.
Group of answer choices

True

False

Given an array arr = {5,6,77,88,99} and key = 5 and using Binary Search, it will take 1 comparison to find the key.
Group of answer choices

True

False
The indegree of a node is the total number of edges going out from that node Group of answer choices True **False**
A graph without cycle is called acyclic Graph. A tree is a good example of acyclic graph. Group of answer choices **True** False

FA 4 FROM NICOLE

There are **4 vertices** indicated in the set of edges below:

E= { (a,b), (a,c), (a,d), (b,e), (c,d), (c,e), (d,e)}

- True
- **False**

In a undirected graph. elements in the set of edges are ordered pairs.

- True
- **False**

In Insertion Sort, we find the smallest element from the unsorted sublist and swap it with the element at the beginning of the unsorted data.

- True
- **False**

Heapify picks the largest child key and compare it to the parent key.

- **True**
- False

In Binary Search, if search item is less than middle element, restrict the search to the right half of the list

- True
- **False**

Given an array arr = {5,6,77,88,99} and key = 5 and using Binary Search, it will take **1 comparison** to find the key.

- True
- **False**

Given an array arr = {5,6,77,88,99} and key = 88 and using Binary Search, it will take **3 comparisons** to find the key.

- True
- **False**

Binary search uses decrease and conquer technique.

- True
- **False**

Graphs are used in the analysis of electrical circuits, finding the shortest route, project planning, linguistics, genetics, social science

- **True**
- False

Any edge may be connected to any other, these connections are called vertices.

- True
- **False**

A **complete graph** is a graph in which all pairs of vertices are adjacent

- **True**
- False

A graph is suppose to be weighted if its every edge is assigned some value which is greater than or equal to zero.

- **True**
- False

A directed graph is connected if, for any pair of vertices, there is a path between them.

- **True**
- False

A graph consists of a number of data items, each of which is called a vertex.

- **True**
- False

In Insertion Sort, we find the smallest element from the unsorted sublist and swap it with the element at the beginning of the unsorted data.
True
False

In sorting, the list is divided into two parts: sorted and unsorted.
True
False

Merge Sort makes use of the technique divide and conquer.
True
False

Linear Search is more efficient than Binary Search.
True
False

In sequential search, the search stops when: **record with matching key is found or when search has examined all records without success.**

- **True**
- False

A Linear search algorithm requires data to be ordered.

- **True**
- False

Special member that uniquely identifies the item in the data set is called a primary key

- True
- **False**

The indegree of a node is the total number of edges going out from that node

- True
- **False**

If e = (u, v) then e is incident on u and v

- **True**
- False

Shell Sort is the most common sorting technique used by card players.

- True
- **False**

A heap is a binary tree. Group of answer choices

- **True**
- False

Both merge sort and quick sort requires recursion for its implementation.

- **True**
- False

FA 4 FROM GWEN

In a undirected graph. elements in the set of edges are ordered pairs.

- True
- **False**

There are 4 vertices indicated in the set of edges below: E= { (a,b), (a,c), (a,d), (b,e), (c,d), (c,e), (d,e)}

- True
- **False**

Simple graph has loops and parallel edges.

- True
- **False**

A graph is suppose to be weighted if its every edge is assigned some value which is greater than or equal to zero.

- **True**
- **False**

If e = (u, v) then e is incident on u and v

- **True**
- False

Any edge may be connected to any other, these connections are called vertices.

- True
- **False**

A graph consists of a number of data items, each of which is called a vertex.

- **True**
- False

The indegree of a node is the total number of edges going out from that node

- True
- **False**

Searching is a process that organizes a collection of data into either ascending or descending order.

- True
- **False**

In Insertion Sort, we find the smallest element from the unsorted sublist and swap it with the element at the beginning of the unsorted data.

- True
- **False**

Shell Sort is the most common sorting technique used by card players.

- True
- **False**

You can perform heap sort even to a binary tree that does not follow the heap property.

- **True**
- False

The basis for dividing the unsorted list in merge sort is defined by the chosen pivot element.

- True
- **False**

In sorting a max heap, the first element that will be transferred to the sorted list is the smallest value.

- True
- **False**

Binary search uses decrease and conquer technique.

- True
- **False**

Linear Search can be performed both in numbers and letters.

- **True**
- False

Given an array arr = {5,6,77,88,99} and key = 88 and using Binary Search, it will take 3 comparisons to find the key.

- True
- **False**

Given an array arr = {5,6,77,88,99} and key = 5 and using Binary Search, it will take 1 comparison to find the key.

- True
- **False**

It will take 3 comparisons to to find the value 7 in the list [1,4,8,7,10,28]?

- True
- **False**

A Linear search algorithm requires data to be ordered.

- **True**
- **False**

A heap can either be a max or min heap.

- **True**
- False

Cycle is simple path in which the first and last vertices are the different4

- True
- **False**

FA 4 FROM ELLA

A directed graph is connected if, for any pair of vertices, there is a path between them.

Group of answer choices
True
False

A graph without cycle is called acyclic Graph. A tree is a good example of acyclic graph.

Group of answer choices
True
False

When the edges in a graph have no direction, the graph is called **undirected**

Group of answer choices
[✓ True]
False

The indegree of a node is the total number of edges going out from that node

Group of answer choices

True

[✓ False]

An edge e = (u,v) is a pair of vertices

Group of answer choices

[✓ True]

False

Graphs are used in the analysis of electrical circuits, finding the shortest route, project planning, linguistics, genetics, social science

Group of answer choices
[✓ True]
False

If *e* = (*u*, *v*) then *e* is incident on *u* and v

Group of answer choices
True
False

Insertion sort is a sequence of interleaved insertion sorts based on an increment sequence.

Group of answer choices
True
False

In Bubble Sort, each time an element moves from the unsorted part to the sorted part one sort pass is completed.

Group of answer choices

True

False

Sorting is used to arrange names and numbers in meaningful ways.

Group of answer choices

True

False

A heap is a binary tree.

Group of answer choices

True

False

In quick sort everything that is less than the pivot element goes to the right.

Group of answer choices

True

False

Merge Sort makes use of the technique divide and conquer.

Group of answer choices

True

False

Given an array arr = {5,6,77,88,99} and key = 5 and using Binary Search, it will take **1 comparison** to find the key.

Group of answer choices

True

False

In sequential search, the search stops when:
record with matching key is found
or when search has examined all records without success.

Group of answer choices

True

False

In Binary Search, if search item is less than middle element, restrict the search to the right half of the list

Group of answer choices

True

False

Linear Search is more efficient than Binary Search.

Group of answer choices

True

False

Binary search uses decrease and conquer technique.

Group of answer choices

True

False

Special member that uniquely identifies the item in the data set is called a **primary key**

Group of answer choices

True

False