1. **Explain: "ADT" Give an example of an ADT description of your own design.**

abstraction - hides the complexity of (implementation) We do not need how push() method was implemented.

stack ADT describe behavior

(array / Linked List)

Push () → adds to a stack

pop() → removes from

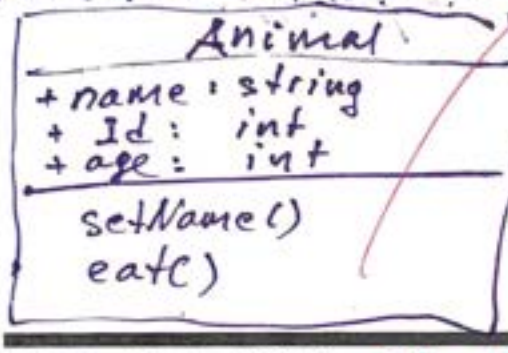
peek() → shows value of top of the stack

Binary tree could be implemented array

Graph could be implemented by 2d array aka matrix

2. **What is a "UML" diagram/representation of an ADT? (use your example from above)**

UML: ADT for Zoo System with class Animal, and instance of that class would be specific animal.



← class

← attributes (characteristics, field or properties) which describe each instance

← methods (operations, functions) specify behavior of a class.

3. **Explain the "STL"** STandart Template Library is quick (we do not need codings from scratch) and efficient (proven with time) way of coding. It consist of: containers it is how we store our data (DS): vector, array, stack, queque, list ... And it comes with functions how to traverse those containers (DS): begin(), end(). Also it comes with Algorithm on those containers: sort, search

4. **Explain "container class(es)" Provide example(s)** containers implemented as class templates

Containers is how we store our data

array
vector
stack
queue
list.

Elements can be deleted from vector but an array not

5. Describe at least two (2) advantages the C++ programmer has when using a VECTOR instead of an ARRAY, as their data structure.

Arrays can not be copy, but vector can

Vector occupy more memory, be they are dynam

Vector's length is dynamic, while array has fixed size, you specify it upon declaration int arr[100]

---

6. Create and design a new class of your own (entirely up to you, what it is, what it's called, what member data elements and functions it has). Create a vector of that class you created. Use it in some basic (at least) way that demonstrates it.
[YOU CAN USE THE SAME EXAMPLE YOU USED IN THE PROGRAMMING ASSIGNMENT]

```
class HappyDog {
private:
  int Monday;
public:
  int weekend();
  void weekday();
    vector<int> happydog;
};
```

data members (variables)

7]. If your goal is to minimize operations/steps needed, which would you rather perform: delete a single item from the middle of a static data structure like an integer array, or, delete a single item from the middle of a dynamic data structure like a linked list? Explain with diagrams and pictures the steps of removing an item from a linked list! Then, answer the following two questions:
(a) Is there a "cost" to using the dynamic data structure? And
(b) When would you suppose it makes sense to use a static data structure?

b) Array is more appropriate for storing
I when there is small or
fixed sized ~~of static data structu~~
II efficient access to of array's elements
by knowing it's index and
III less memory consumption

-3

8]. Name and explain three aspects/advantages of OOP.

Encapsulation - bring related things under one entity (class)
Abstraction - do not care about details, just do most important parts.
Polymorphism - from Greek "many shapes" we taking instances from inheritance can add specific functions that we need.
Inheritance - help us get rid off redundant (duplicate) code by creating instances from encapsulation object.

or any dynamic DS

7. ~~Arrays~~ Vector have ability to resize itself automatically while insertion and deletion

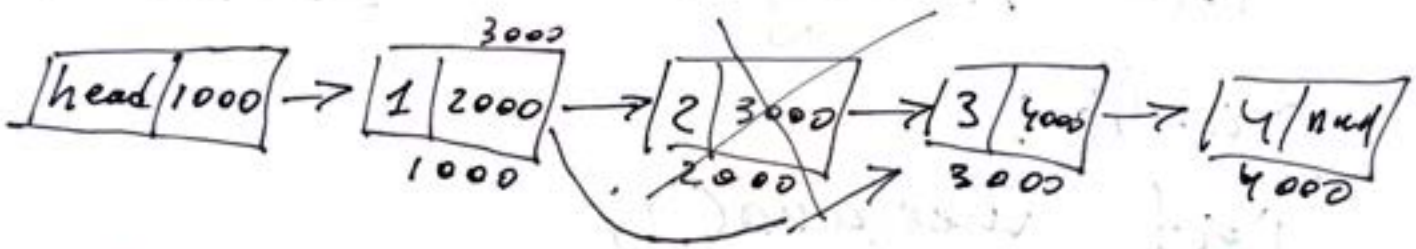
to delete: vector.erase(position) ← position of element to be removed

V.erase(V.begin() + 6)

return iterator points to ~~beginning~~ first element in vec.

Dynamic DS allows easy insertion or deletion.

1. we have to del link
2. then we have to delete that node from heap memory.



9. Back in the day: It was procedural programming: variables and functions like in first programming class in a college. Then you have a lot functions and if u need make changes in one it will break in functions related to that one. We call this "spaghetti code".

In OOP we combine and group related variables and functions working on those vars into an object. Those vars call properties and functions as methods

= encapsulation

Car
properties
Make
year
speed()
stop()

9]. For a large C++ project would you want to put all your code in one .cpp ? Why not ? What sort of files would be a part of your "project" and what would you find in each?

Back in the day, programmers used to do in a single file. But nowadays programmers do in separate files. It is better for efficiency, better to read and understand code. It will consists of main.cpp, another file.cpp for example for function, another one for Class.

10]. Show C++ code to recursively computer factorial of a number

```
def factorial():
    factorial(5)
        factorial(4)
            factorial(3)
                factorial(2)
                    factorial(1) - base case
```

11]. Explain why an iterative solution might be a better idea for factorial.

for small size

iterative

It will be faster and efficiently than recursive. NOT necessarily because use less memory, unlike recursive put data in stack

Then, name at least 2 examples of a coding task where going with a recursive solution does often seem like the better and more sensible choice.

Recursive is good for binary search. It is fastest way to find a number in a sorted array.

towers of Hanoi

10. Recursion is when function call itself or another function

```
void b { cout<<"bc"; b() }
int main() { b() }
```

you need base case is ending point for the function

Factorial: $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$ another words whatever # and all the way to 1.

```
int factorialFinder(int x) {
```

we need base case in order to stop this.

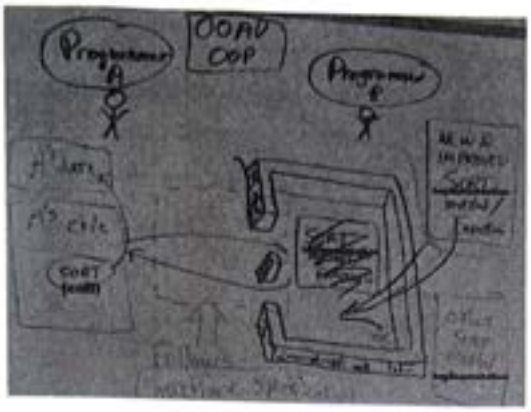
```
if (x==1) {
    return 1; ← when y return smth it is when y function ends.
} else {
    return x * factorialFinder(x-1)
                ↑
                5 · 4 ← what is it? no answer then it will take 4 · 3 same up till 1
}
}
```

```
int main()
{ cout<<factorialFinder(5)<<endl;
```

11) itirative & is loop.

```
int findFactorial(int x) {
    for (int i = x; i>1; i--)
        int product x = i
    return product
```

12) Recursive function are generally slower it require more memory



12]. Programmer A has code and data and wants to call a sort routine that Programmer B has written and is responsible for maintaining and updating. Which of the following is true in the realm of "Object Oriented Design and Programming: [choose one only!]

[ ] Programmer A should know <u>all of the exact details</u> of how the algorithm Programmer B is using for the sort operates, in detail.

[√] Programmer A and Programmer B should agree on the "<u>Interface Specification</u>" only and Programmer A and B need know **<u>no</u>** <u>details</u> of the other's code otherwise.

13]. Static Data Structures vs. Dynamic Data Structure:
I need to store a bunch of numbers –
- order doesn't particularly matter to me , but I want no blank/empty spaces in the middle of my data structure as I use it.

What's a static data structure I can use? Give pros and cons of going with that.

array

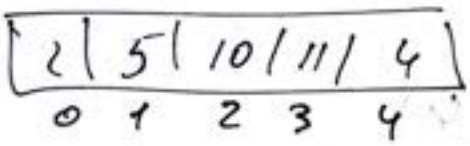
Drawbacks too little

-4

What's a dynamic data structure I can use ? Give the pros and cons of going with that.

Static Data Structure

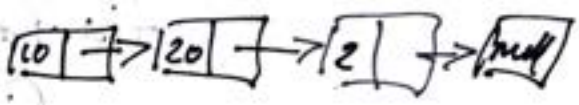
Array



- Provide easier access to element through index.
- can not change size once created
- not flexible

Dynamic DS

Linked list



- Not provide easier access (have to go throug nodes in order to reach desired)
- size can be updated
- flexible

bag.h

```
//the BAG class implementation
#ifndef _BAG_
#define _BAG_

typedef  int bag_type;

class Bag {

private:
   int      count;     // num
   bag_type data[20];  // dat

public:
         Bag();
   bool  insert(bag_type);
   bool  remove(bag_type);
   int   size();
   void  clear();
   bool  inbag(bag_type);
   int   howmany(bag_type);
};
#endif
```

bag.cpp

```
#include "bag.h"  //the BAG  IMPLEMENTATION code
#include <iostream>
Bag::Bag()                    // (CONSTRUCTOR)
{  count=0; }
//-----------------------------------------
void Bag::clear()             // CLEAR
{   count=0; }
//-----------------------------------------
bool Bag::insert(bag_type value)  // INSERT
{  bool reply;
   if(count <20 ) {
      data[count]=value;
      reply=true;
      count++; } else {reply=false;}
   return reply;}
//-----------------------------------------
bool Bag::inbag(bag_type value)   // INBAG
{  bool reply=false;
   int  index;
   for(index=0;index<count&& !reply;index++)
      if(data[index] == value) reply=true;
   return reply;}
//-----------------------------------------
int Bag::howmany(bag_type value)  // HOWMANY
{  int thismany=0;
   int index;
   for(index=0;index<count;index++)
      if(data[index]==value) thismany++;
   return thismany;}
//-----------------------------------------
bool Bag::remove(bag_type value)  // REMOVE
{  bool reply=false;
   int  index;
   if(howmany(value) == 0) return reply;
   reply=true;
   index=0;
   while(data[index] != value) index++;
   for(;index<count;index++)
      data[index]=data[index+1];
   count--;
   return reply;}
//-----------------------------------------
int Bag::size()               // SIZE
{   return count; }
```

main.cpp

```
// DRIVER program to test the BAG
#include <iostream>
#include <cstdlib>
#include "bag.h"
using namespace std;
int main(int argc, char *argv[])
{ Bag b;
  bag_type value;
  cout << "Bag\n";
  b.insert(4);
  do {value=rand()%6 +1;
  } while(b.insert(value));
  cout << b.size()<< " elements in bag\n";
  cout << b.howmany(4) << " fours\n";
  b.remove(4);
  cout << b.size()<< " elements in bag\n";
  cout << b.howmany(4) << " fours\n";
  cout << b.howmany(5) << " fives\n";
  while(b.inbag(5)) b.remove(5);
  cout << b.howmany(5) << " fives\n";
  return 0;
}
```

bagtest.cpp

```
// DRIVER program to test the BAG
#include <iostream>
#include <cstdlib>
#include "bag.h"
using namespace std;
int main(int argc, char *argv[])
{
  Bag b;
  bag_type value;
  cout << "Bag\n";
  b.insert(4);
  do {
     value=rand()%6 +1;
  } while(b.insert(value));
  cout << b.size()<< " elements in the bag\n";
  cout << b.howmany(4) << " fours " << endl;
  b.remove(4);
  cout << b.size()<< " elements in the bag\n";
  cout << b.howmany(4) << " fours \n";
  cout << b.howmany(5) << " fives \n";
  while(b.inbag(5)) b.remove(5);
  cout << b.howmany(5) << " fives \n";
  return 0;
}
```

14]. Explain in your own words why the project above might best be kept in separate files (as illustrated above) – even though in theory you could have all that in a single cpp and it would work just fine. Also explain what I am reminding the programmer of by circling the words "main" in the 2 files on the right.

We can not use two main in one project. It won't run.

If we will keep everything in one file it will be „spaghetti code" it will be harder to read or understand it, harder to debug.

15]. Presume your code ALREADY has a 1000 item data structure (integer array or integer vector) with 1000 random integers in it ALREADY IN SORTED ORDER.
[sound familiar? Yes - this question is based on your September 29 assignment . . .]

Show the C++ code that does these actions: (show only this part of the program)
Asks for a value to search for in the data structure. [user enters an integer value]
Performs an iterative binary search to find out if the value is in the data structure.
Report the index where found, if found.

-3

15ExtraCredit]. Show the C++ code that does these actions: (show only this part of the program)
Asks for a value to search for in the data structure. [user enters an integer value]
Performs a RECURSIVE binary search to find out if the value is in the data structure.
Report the index where found, if found.

```
int array[];
int sizeofarray;
get middle of the array pivot.
check if pivot is the looking num
if it larger then
compare from middle till end of ar
if it smaller then
compare from beggining till middle
```

+1

16]. This semester we looked at a C++ program and a set of associated header files which included:

listnode.h list.h stack.h queue.h

The list.h file contained the definition of a class called List that had these functions:

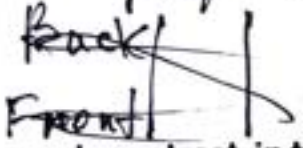
insertAtFront
insertAtBack
removeFromFront
removeFromBack

which are for manipulating data in the dynamic data structure type: linked lists.

insert at Front and remove From Front

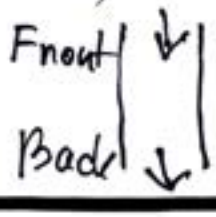
In a stack.h header file, two fundamental STACK SPECIFIC operations/functions should be defined. What are the Stack functions called? Secondly, for those 2 functions, which of the list.h functions named above would you call to do each of those stack operations? Why?

Stack use principle First come, last go. so we push function in to stack and pop function from a stack



In contrast, in the queue.h header file, which two QUEUE SPECIFIC operations would be defined for queue operations? Also, for those 2 functions, which of the list.h functions named above would you call to do those queue operations? Why?

Queue use principle first come first go



EXTRA CREDIT: In this space share 2 key things you know about "smart pointers"

- They take care of garbage collection.
- memory management

+2