Answered by: Yuzhou Shao(Ken)

Note: The Original Question texts are changed into Blue Color

Besides, the answers are mainly written in Black(also, some diagram or annotations are in other colors).

# Interview questions

## Queues and stacks

\* Describe the difference between a stack and a queue. (5 points)

|  |  |
| --- | --- |
| Stack | Queue |
| The stack elements are Last in First Out(LIFO) | The Queue elements are First in First Out(FIFO) |
| Insertion Operation is called Push Operation | Insertion Operation is called Enqueue Operation |
| Deletion Operation is called Pop Operation | Deletion Operation is called Dequeue Operation |
| Push and Pop Operation takes place from one end of the stack | Enqueue and Dequeue Operation takes place from a different end of the queue |
| The most accessible element is called Top and the least accessible is called the Bottom of the stack | The insertion end is called Rear End and the deletion end is called the Front End. |
| Only one pointer is used for performing operations | Two pointers are used to perform operations |
| Empty condition is checked using  Top==-1 | Empty condition is checked using  Front==-1||Front==Rear+1 |
| Full condition is checked using  Top==Max-1 | Full condition is checked using  Rear==Max-1 |
| There are no variants available for stack | There are 3 types of variants i.e. circular queue, double-ended queue and priority queue. |
| Can be considered as a vertical collection visual | Can be considered as a horizontal collection visual |
| Used to solve the recursive type problems | Used to solve the problem with sequential processing |
| Simple Implementation | Complex implementation in comparison to stack |

To be more specific, **Stack** is a linear data structure that follows the specific order to perform the operations. In stack data structure, there is only one sequence to access the element. In the stack, we insert the element from one end with push operation and delete the element from the same end using pop operation. The end of the stack used to perform all the operations is called the top of the stack. Therefore, a stack follows the LIFO(Last in First Out) principle, which means the element that is inserted last will be the first element to come out of the stack. The most crucial thing to remember in the stack data structure is that it merely stores the elements of the same data type .

4

3

2

1

0

push

pop

**Condition to Check if Stack is Empty**

Int Empty()

{

if(top==1)

return 1;

else return 0;

}

**Condition to Check if Stack is Full**

Int Full()

{

If(top==MAX-1)

return 1;

else

return 0：

}

**Queue** is a linear data structure in which we can insert the element from one side of the list and delete the element from the other side of the list. The end of the list from where the elements are inserted is called the rear end and the end from where the elements are deleted is called the front end. Consequently, the queue data structure follows the FIFO(First in First Out) principle, which means the element inserted first from the rear end will be the first element to be deleted from the front end. The insertion technique in the queue data structure is called enqueue operation. While performing operations in the queue, there are 2 pointers, front pointer and rear pointer, where the front pointer is used to point the element that is added first in the queue. Also, just like stack data structure, we can only store elements of the same data type in the queue data structure.

0 1 2 3 4 5

1

10

8

6

3

Rear

Front

**Condition to Check if Queue is Empty**

Int Empty()

{ if(front==-1||front==rear+1)

return 1;

else

return 0;

}

**Condition to Check if Queue is Empty**

Int Empty()

{

If(front==-1||front==rear+1)

return 1;

else

return 0;

}

\* Describe a typical programming use for a queue. (5 points)

Queue is applied when the program is not required to be processed instantly, but have to be processed in First In First Out order like Breadth First Search. This property of Queue makes it also useful in following kind of scenarios.

1. When a resource is shared among multiple consumers, such as CPU scheduling, Disk Scheduling.
2. When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Example include IO Buffers, pipes, file IO, etc.
3. In Operating Systems:

Semaphores

First Come First Served Scheduling(FCFS), such as FIFO queue.

Spooling in printers

Buffer for devices like keyboard

1. In Networks:

Queues in routers/switches

Mail Queues

1. Variations: (Deque, Priority Queue, Doubly Ended Priority Queue)

Beside them, to me more specific, the different types of queues and its usages can be classified as below:

**Circular Queue** is a linear data structure in which the operations are performed based on First In First Out(FIFO) principle and the last position is connected back to the first position to make a circle(also called as “Ring Buffer”). Most of the implementations are as below scenarios:

Memory Management: The unused memory locations in the case of ordinary queues can be utilized in circular queues.

Traffic system: In a computer- controlled traffic system, circular queues are used to switch on the traffic lights one by one repeatedly as per the time set.

CPU Scheduling: As mentioned in the beginning, Operating Systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.

**Input restricted Queue**: In this type of Queue, the input can be taken from a single side only which is rear deletion of element can be done from both side(front and rear). This kind of Queue does not follow First In First Out(FIFO)

Enqueue(inserting)

Dequeue

Dequeue(deleting)

This queue is used in the cases where the consumption of the data needs to be in FIFO order but and if there is a need to remove the recently enqueued data for some causes and one such case can be irrelevant data, performance issue, and so on.

**Output restricted Queue:** In this type of Queue, the input can be taken from both sides(rear and front) and deletion of the element can be done from only one side(front).

Enqueue(inserting)

Dequeue

Enqueue

It is mainly used while the inputs have some priority order to be executed and the input can be placed even in the first place so that it is executed first.

**Double ended Queue** is another queue data structure in which the insertion and deletion operations are performed at both the ends(front and rear). That means, we can insert at both front and rear positions and can delete from both front and rear positions.

Dequeue

Enqueue

Dequeue

Enqueue

Since Deque supports both stack and queue operations, it can be sued as both. The Deque data structure supports clockwise and anticlockwise rotation in 0(1) time which can be useful in certain applications. Also, the problems where elements need to be removed and or added both ends can be efficiently solved using Deque.

**Priority Queue:** As briefly mentioned before,a priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority. There are 2 types of Priority Queues below:

Ascending Priority Queue: Element can be inserted arbitrarily but only smallest element can be removed. For instance, imagine there is an array with elements of 4,2,8 in the same order. Therefore, while inserting the elements, the insertion will be in the same sequence. However, the order will be 2,4,8 while deleting.

Descending priority Queue: Element can be inserted arbitrarily but only the largest element can be removed first from the given Queue. For instance, if there is an array with elements of 4,2,8 in the same order. Therefore, while inserting the elements, the insertion will be in the same sequence. However, the order will be 8,4,2 while deleting.

The priority queue is primarily used in the CPU scheduling algorithms.

## Linked lists

\* Use C++ to show the structure of an element in a doubly linked list in which each element holds a single character. Describe briefly what any pointers point to. (4 points)

**Doubly Linked list** is a container storing data. It is one of the linked list, where data is stored in a discontinuous memory space. Each data node contains the item, the pointer of the previous node, and the pointer of the next node.

Consequently, Doubly Linked list can delete a specific node with the given time complexity of O(1). It takes O(n) time for querying a specific node. Therefore, in the case of a less random query, while inserting or deleting items at the middle of the code is universally required. Doubly Linked List is helpful as a double container for storing data in this scenario. It’s structure is as below diagram.

null node node node null

next

item

prev

next

item

prev

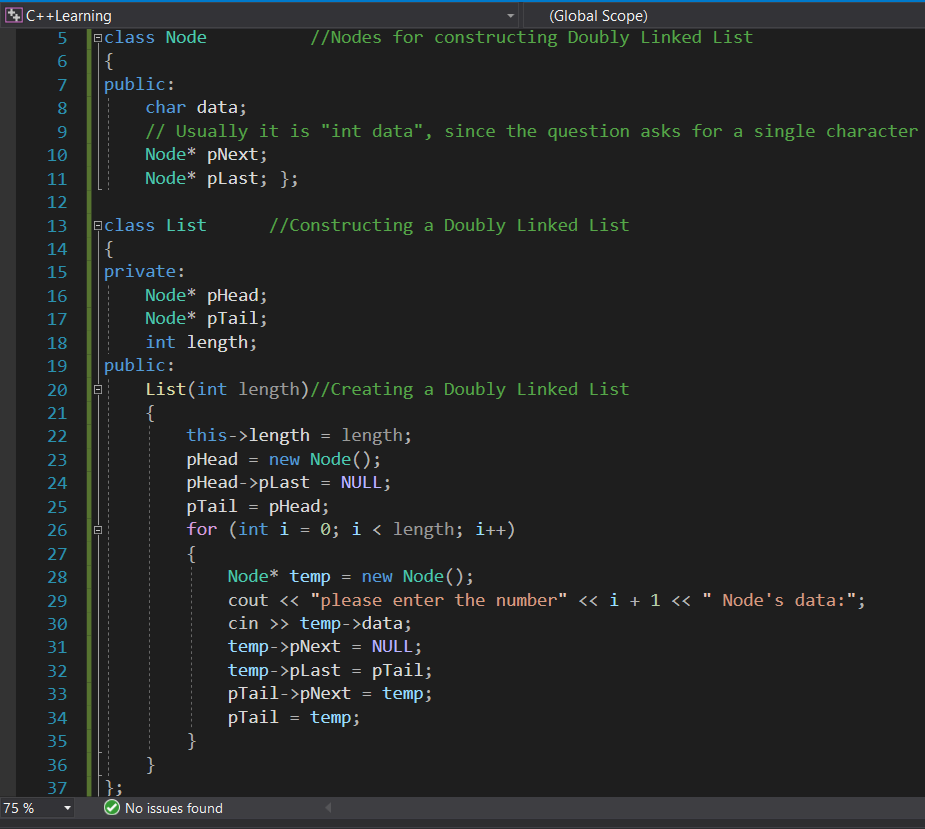
next

item

prev

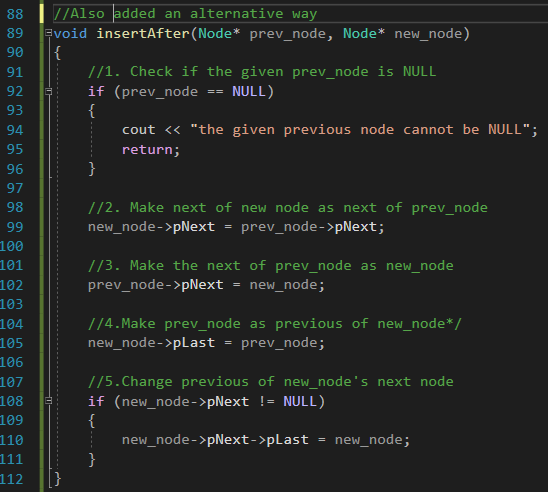
head tail

Below is an example of a snippet of C++ code for Doubly Linked List

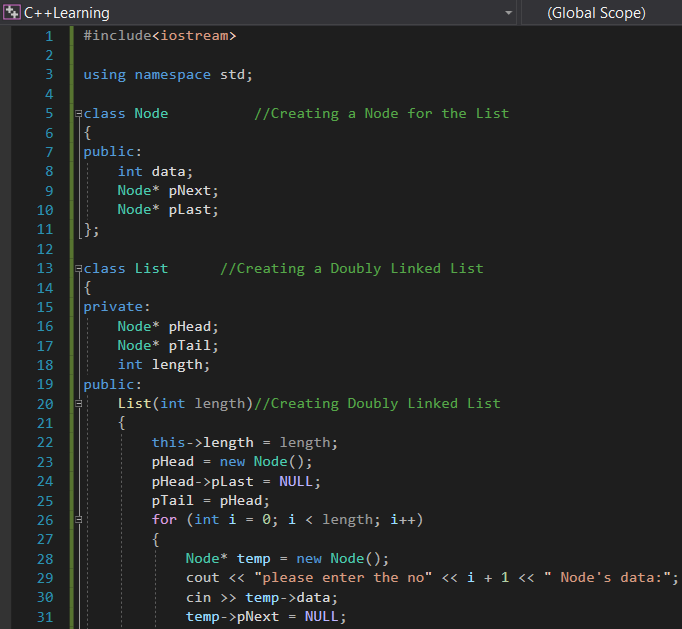


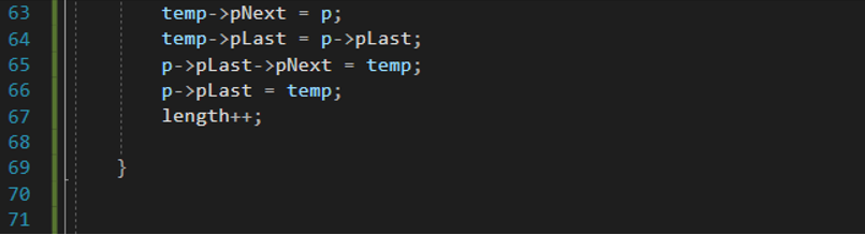
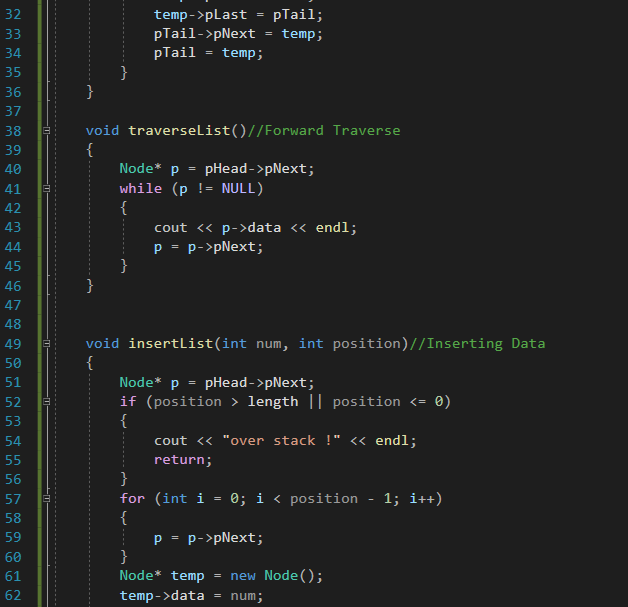
\* Use C++ to show the operations necessary to insert a list element given a pointer to the new element and a pointer to the element after which the new element needs to be inserted. (6 points)

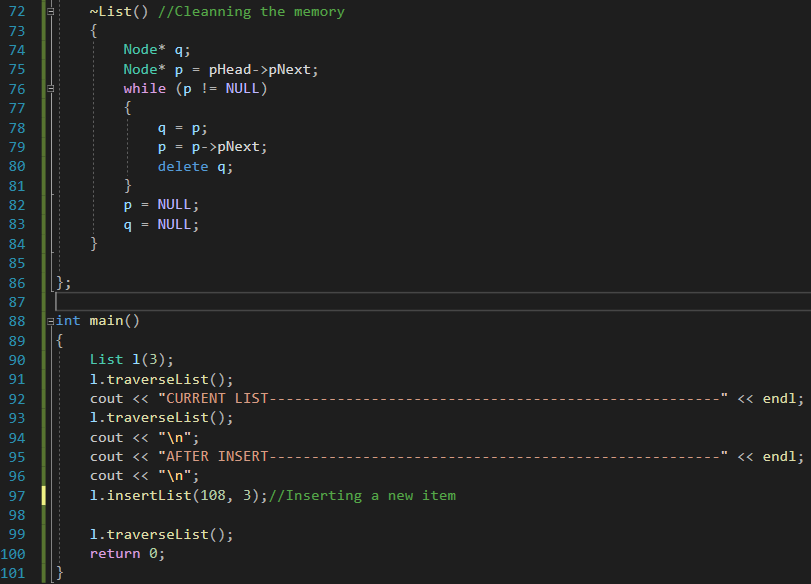
In briefly, the function can be as below.



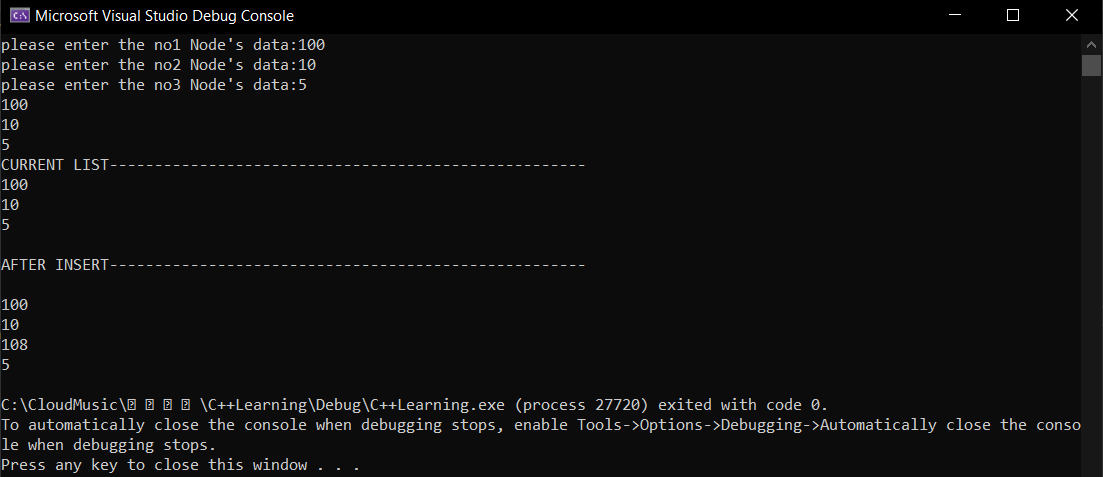
While to be more specific, below is another example, while from line 48 to line 70, the function demos how insertion works with the pointers.







Inside main function, in line 97, 108 is the number and 3 is the position to be inserted.



Above is the output where I took 100, 10, and 5 as the input items, and 108 was inserted at the given order.

## Binary trees

\* Use C++ to show the structure of an element of a sorted, binary tree in which each element contains a single character. Describe briefly what any pointers point to. (4 points)

\* Use C++ to show the operations necessary to locate the element in the sorted binary tree which matches a given character. (6 points)

Generally, a Binary tree is a hierarchical data structure in which every node has 2 children(i.e. left child and right child), as each node has 2 children hence the name “Binary”. Root node is the topmost node of the tree.

In this representation, array structure is used to implement the tree. Size of array

is equal to the total nodes in the tree, index of root node is 0. If a node is

at ‘i’ location, then its left child is at ‘2i’ index and right child at ‘2i+1’

location in the array. Visual Representation.

9

8

6

5

7

4

3

2

1

1 2 3 4 5 6 7 8 9 10 11 12 13

Linked-list Structure

**Left Data Right**

A

C

B

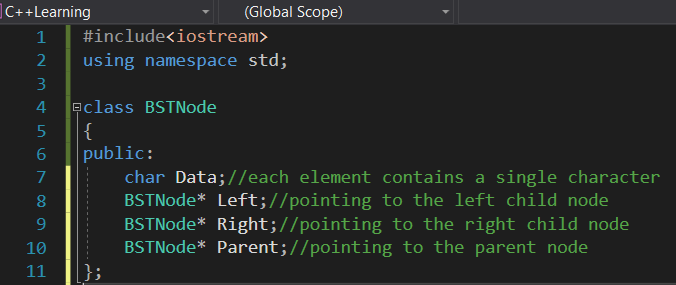
F

E

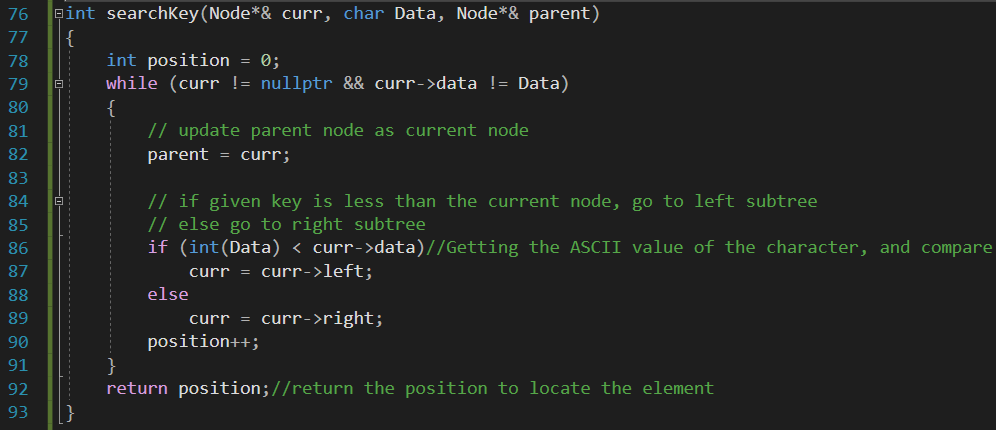
D

Therefore, we can implement it using C++ code as below:

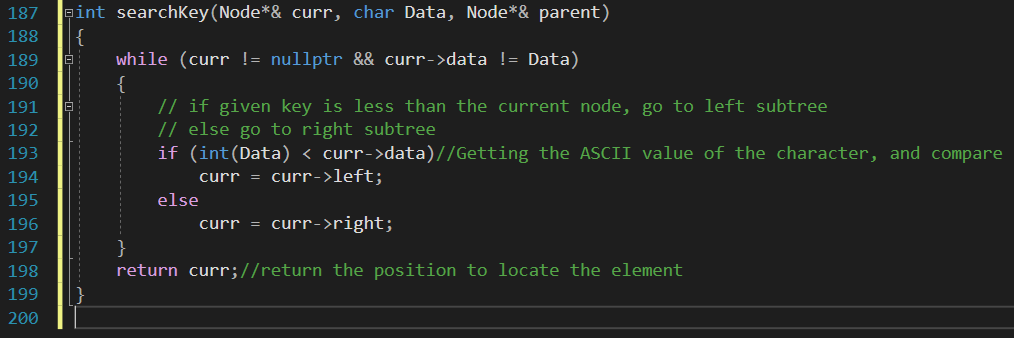
**For the former question above of Binary Trees Section:**



**For the latter question above of the Section of Binary Trees:**



There is also another alternative approach, slightly different.



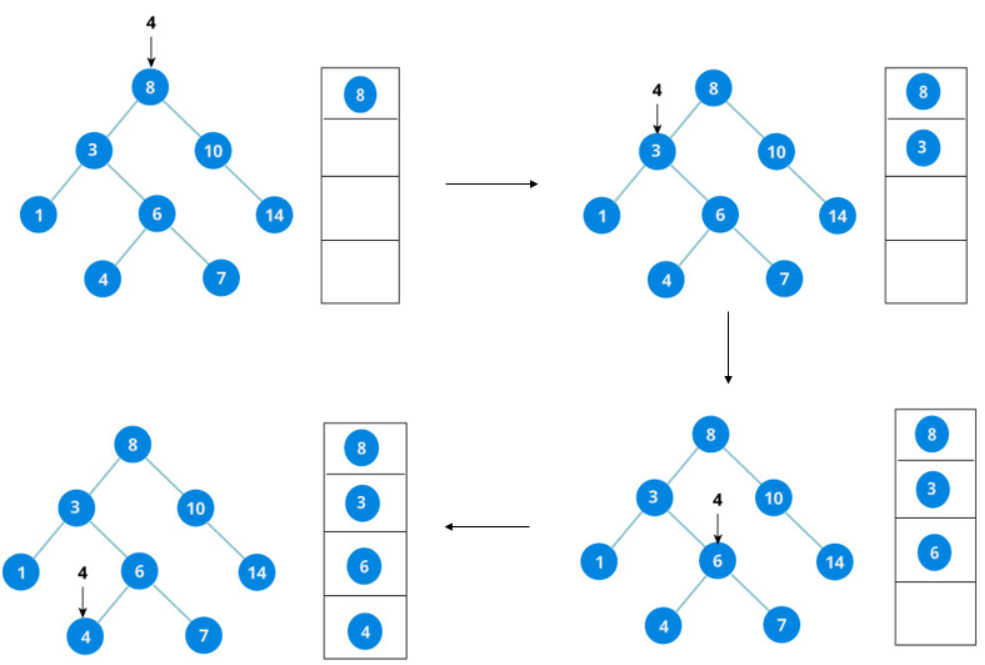


Image referenced from iq.opengenus.org

The visualization of Binary tree Searching Concept can be as above diagram.

## C++ specifics

\* Compare the following function and macro definitions. In what cases will they produce different results and/or side effects? (5 points)

```

int square(int val) { return val\*val; }

#define square(val) (val\*val)

```

Since we know that int is merely 32 bits long. Unexpected results will be produced in the first code when val\*val produces a result which is more than 32 bits long. Because the return value is int, the result will be chopped to fit an int and returned.

In case of the second one, we can assign new variable with any numeric datatype for macro, making it more flexible than function. For instance, we can apply float, long, long long int, etc.

However, there are big advantages of Macros. Furthermore, the comparison between macro definition(2nd line) and function(1st line) are as below:

|  |  |
| --- | --- |
| **#define square (val) (val\*val)** | **int square(int val) { return val\*val; }** |
| No type Checking is done in Macro | Type Checking is Done in Function |
| Using Macro increases the code length | Using Function keeps the code length unaffected |
| Use of macro can lead to side effect at later stages | Functions do no lead to any side effect in any case |
| Speed of Execution using Macro is Faster | Speed of Execution using Function is Slower |
| Before Compilation, macro name is replaced by macro value | During function call, transfer of control takes place |
| Macros are useful when small code is repeated many time | Function are useful when large code is to be written |
| Macro does not check any Compile - Time Errors | Function checks Compile - Time Errors |

## ConvertStringToInt

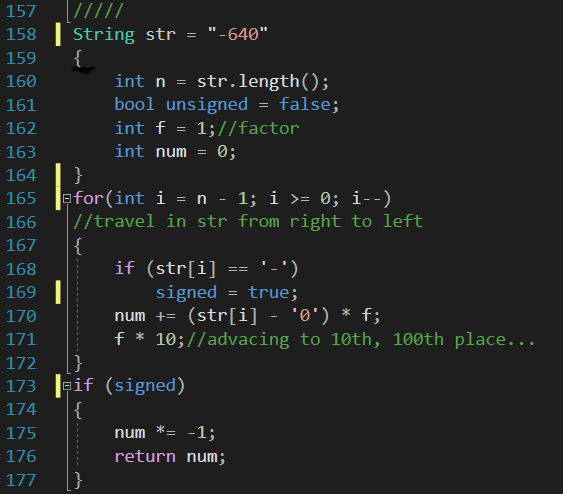
\* Write a C++ function which converts a character string into a signed integer without using any library functions. Assume that the string contains a valid integer, and no white space. (10 points)

```

// example of how the function you write would be used

int val = ConvertStringToInt("-640"); // val would end up with the integer value -640 in it

```



The Key Concept is ‘5’ – ‘0’ = 5

assign value of 5 assign value of 0

Similarly we can get ‘17’ – ‘0’ = 17.

Furthermore, a string is an array of char. So I check if the 1st char is negative and store it’s value in an int variable. Then for the rest, ascii table can be used to convert from characters to int. And multiply by the array 10^(length-index), -1 if it’s negative. Then keep adding the result in a variable.

## C++ specifics

Given the following C++ String class and its defined operations, list the String operations that are called from function Test() in the order they are called. Assume the four unimplemented functions are implemented elsewhere. (5 points)

```

class String

{

public:

String() { m\_str = new char[1]; \*m\_str = 0; }

~String();

String(const char\* str);

String(const String& other);

String& operator = (const String& other);

private:

char\* m\_str;

};

String Func(String str1)

{

String str2;

str2 = str1;

return str2;

}

void Test()

{

String str3("Hello");

str3 = Func(str3);

}

```

* Depend on the condition, Whether Text() is called first from main() is not specified.

In Text() function

1. str3 is constructed with constructor number 2, string(const char\*str);
2. when “Hello” is passed to Func() , str1 is constructed with constructor 3 or 2, where 3 print from constructor and check.
3. str2 is default constructed with constructor number 1, string(){m.str….}
4. In Func now str2= str1, where assignment operator= is called from str2
5. Step 4 one more assignment operator on str3… last line in Test call

\* Write the assignment operator for the String class defined above. Keep in mind the memory management assumptions implied by the implementation of the default constructor defined above. (10 points)

```

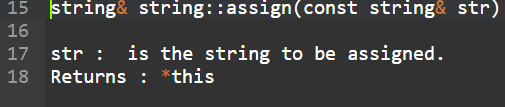
String& String::operator= (const String& other) {

}

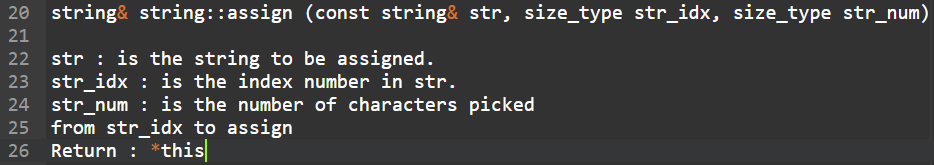
```

Usually, as the assignment operator for above String class, the member function assign() is used for the assignments, it assigns a new value to the string, replacing its current contents. (However, if it is in Java, there is Virtual Machine which will handle the memory automatically.)

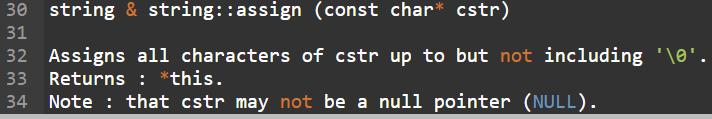
Case1: Assign the value of string str.



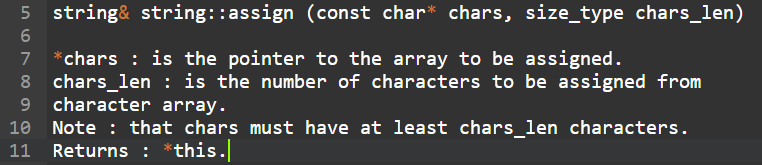
Case2: Assigns at most\_str\_num characters of str starting with index str\_idx. It throws out\_of\_range if str\_idx>str.size().



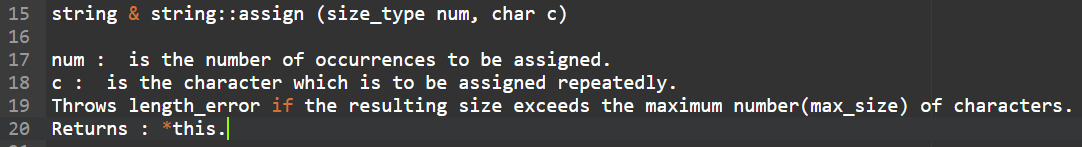
Case3: Assign the characters of the C-string cstr. It throws length\_error if the resulting size exceeds the maximum number of characters.



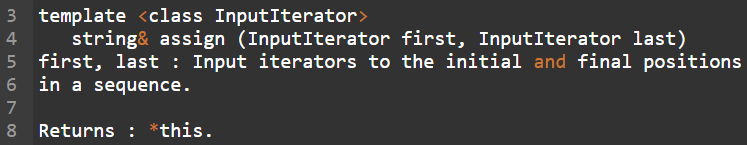
Case4: Assigns chars\_len characters of the character array chars. It throws length\_error if the resulting size exceeds the maximum number of characters.



Case5: Assigns num occurrences of character c. It throws length\_error if num is equal to string::npos



Case6: Assigns all characters of the range[beg, end]. It throws length\_error if range outruns the actual content of string.



## 2D and 3D vectors

\* Given a 2D vector A in the x-y plane of length |A| and angle theta to the x-axis, give the equations for the x and y components of A. (4 points)

|A|

y

According to the question, a diagram can be drawn as right.

For each end of the vector, we can draw one dash line parallel to

θ

x-axis and another parallel to y-axis respectively.

Therefore, the x component of A is cos θ \* |A| ,

0

x

the y component of A is sin θ \* |A|

\* Given the x,y,z components of a 3D vector A, give the equation for the angle between the vector and the x-y plane. (6 points)

z-axis

Based on the given information, I have draw a sketch as right.

Where each dash line is parallel to x-axis, y-axis or z-axis respectively.

A

A is the end point of vector and A’ is the projection of A in x-y plane.

Consequently, AA’ is perpendicular to plane x-y, ∠AOA’ is the angle we

z

are finding. Let’s set it as θ. So that we have:

x-axis

θ

= = =

O

Therefore, θ= which is the angle between the

y

vector and the x-y plane.

y -axis

x

## Physics

\* Given a 3D point starting at position P1 and moving with constant velocity vector V, write an equation for the position P2 of the point after elapsed time t. (5 points)

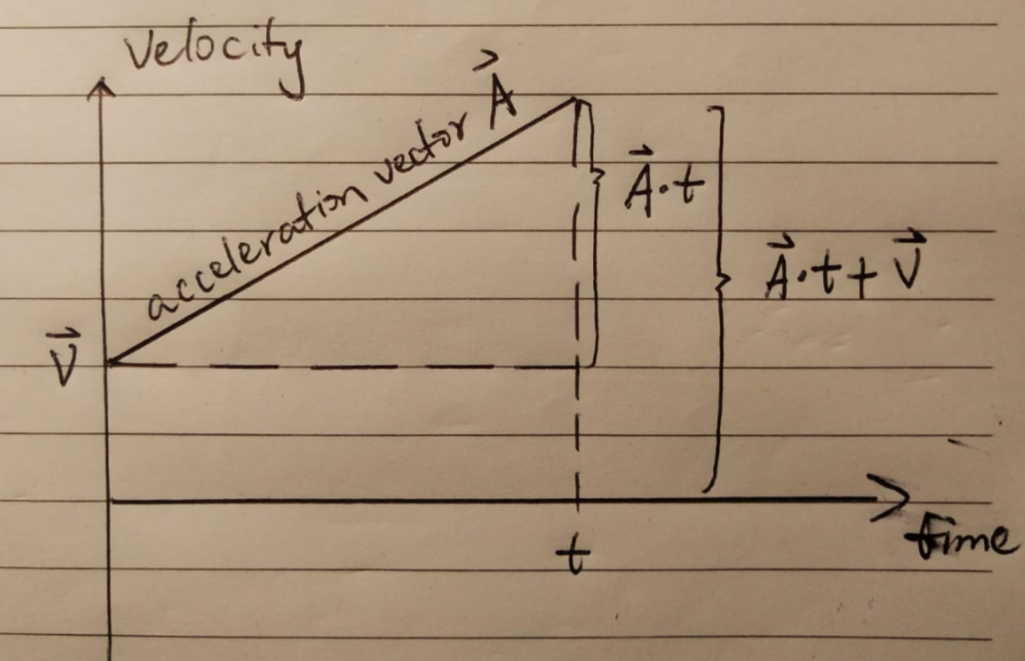
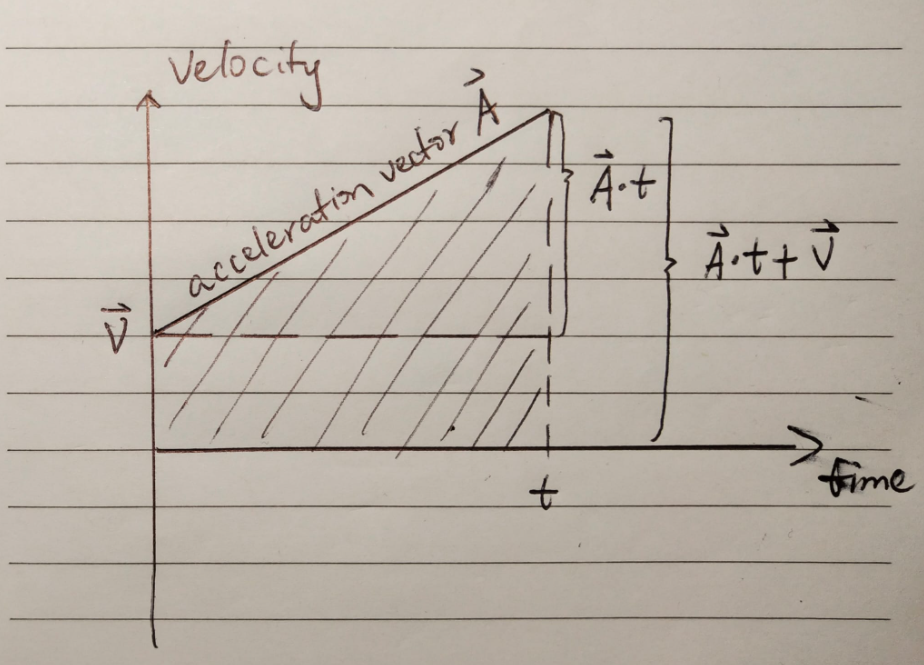
\* Given a 3D point starting at position P1 and moving with initial velocity vector V and constant acceleration vector A, write an equation for the position P2 of the point after elapsed time t. (7 points)

Note: Since it is a 3D point, P1 should contains 3 parameters . Similarly, P2 should be

While velocity vector contains the direction of the speed and do the operations with 3D coordinate automatically, therefore we can write to indicate it, where it should be .

For the 1st question above, the answer is:

For the 2nd question above, we can draw the diagram as follow, where the shadow aera (down right) of the trapezoid stands for the displacement of the given 3D point.

0

0

Therefore, the answer is:

## What does it do?

\* Examine the following function. What does it accomplish? (8 points)

```

int foo(int val)

{

int n=0;

while (val)

{

val &= val-1;

n++;

}

return n;

}

```

There is some bitwise Logical rule that would explain what this does in general. Briefly, bitwise operations evaluate 2 integral values in binary(base 2) form. They compare the bits are corresponding positions and then assign values based on the comparison.

While int is a function declaration, as above code, it performs bitwise and val and val-1.

Then counts the number of times this operation is possible till val becomes 0.

## Palettes

\* Describe what is meant by a "4-bit palette image". (4 points)

A 4-bit palette image is simply one in which each pixel is represented by 4 bits. A 4-bit image can contain 16(24) colors, each pixel having a numerical value between 0 and 15. The color palette for a 4-bit image will therefore normally have 16 entries between 0-15. As a 1-bit image might be called a 2-color image, a 4-bit image is also called a 16-color image.

4-BIT RGB PALETTE indicates 1 red bit, 2 green bits, 1 blue bit. Similar in concept to the Uzebox palette, which also uses bit combinations for the RGB color channels, in that case adding up to 8 bits in total, rather than 4.



Image referenced from https://lospec.com/

\* Describe what is meant by a "24-bit true-color image". (4 points)

A color image is typically represented by a bit depth ranging from 8 to 24 or higher. With a 24-bit image, the bits are often divided into 3 groupings: 8 for red, 8 for green, and 8 for blue. Combinations of those bits are used to represent other colors. A 24-bit image offers 2^24 (i.e. 16.7 million) color values, which is sufficient to cover the full range of human color perception fairly well. Increasingly scanners are capturing 10 bits or more per color channel and often outputting 8 bits to compensate for “noise” in the scanner and present an image that more closely mimics human perception.

It is common for each component intensity to be stored as an 8-bit integer, and so each pixel requires 24 bits to completely and accurately specify its color. Image formats that store a full 24 bits to describe the color of each and every pixel are therefore known as 24-bit color images. The term 24-bit is also used to describe monitor displays that use 24 bit per pixel in their display memories, and which are hence capable of displaying a full range of colors.

However, the disadvantages of 24-bit images mainly is that it requires 3 times as much memory, disk space and processing time to store and manipulate 24-bit color images as compared to 8-bit color images. In addition, there is often not much point in being able to store all those different colors if the final output device(e.g. screen or printer) can only actually produce a fraction of them. Since it is possible to use colormaps to produce 8-bit color images that look almost as good, at the time of writing 24-bit displays are relatively little used. However, it is to be expected that as the technology becomes cheaper, their use in image processing will grow.

\* How many shades of gray can be displayed in a 24-bit color image? (2 points)

Shades of gray means variations of gray or grey include achromatic grayscale shades, which lie exactly between white and black, and nearby colors with low colorfulness.

As mentioned in the answer of the last question. Using 24 bits to encode color information allows Eqn:eqn24b1 different colors to be represented, and this is sufficient to cover the full range of human color perception fairly well.

## General engineering

\* Describe the major benefits of data-oriented design. (5 points)

The main advantage of an object-oriented system is that the class tree is dynamic and can grow. Developers in an object-oriented environment fosters the growth of the class tree by defining new, more specialized classes to perform the tasks that the required applications supposed to perform. Creating additional applications will require no more than assembling classes from the class library.

To be more specific, in Object-Oriented Programming, splitting a process between multiple cores can be a painful process due to synchronization errors caused by different threads trying to concurrently access the same data. Having the threads wait for their turn to access the data results in a lot of idling, and the returns in terms of performance increase can be unsatisfactory. DOD simplifies parallelization. Because data is processed in groups, it is easy to split the data between different threads and the code will not be running into synchronization related problems.

Another strong point of Data-Oriented Design is the possibility for optimized cache utilization. In modern hardware, a key point to achieving great performance is to order the data in memory so that it can be efficiently used over and over again. If the data is laid contiguously in the memory, the data can be processed with near perfect cache usage resulting in superb performance. While optimizing the algorithms used to transform the data is certainly important, by looking back a bit further to how the data is being handled can yield even greater results in the large perspective.

Another already discussed advantage of using Data-Oriented composition is its modularity. While it does not bring any extra performance to the program, it makes the development process significantly easier to manage. Keeping the functions small and avoiding dependencies between different parts of the code keeps the codebase from branching out, which improves the readability of the code, and makes updating and rewriting it significantly easier. While modularity is not only limited to Data-Oriented Design, it is a major factor to take into account, and thus worth mentioning.

The last major benefits of DOD is how easy it makes it to test the code. Because all the functions are focused on directly transforming the data, unit tests simply have to take in some kind of input data, perform the transformation and see if the output is as expected. No need to worry about dependencies between different parts of the code.

\* Describe the advantage of Entity-Component-Systems over object-oriented class hierarchies. (5 points)

The major advantage of entity-component-systems over object-oriented programming are how they can be optimized to fit a program’s needs. The user is free to implement memory management by themselves, and decide whether they want to go for a simplistic implementation which may not be very cache effective, or if they want to go for a more complex, highly optimized system, which uses memory to the fullest, but may also suffer from fragmenting data tables.

\*Data Structures for effective Memory Management

CPU must read and write the data in increasing order. Therefore, for entity-component-system data structures, important things to prioritize are:

Data is being stored in RAM as contiguously as possible.

All entity system processors(systems) process their data in the order it is in Random Access Memory.

Therefore, below are the methods of implementing the actual data structures:

Approach 1: BigArray per ComponentType

Approach 2: Massive Interleaved Array

Approach 3:Seperated Access and Component Arrays.

Approach 4: More Explicit Structure and More Tables

\*Multi-Threading

Distributing a software to multiple threads has obvious performance benefits over having it run on a single thread. Concurrent operations drastically improve throughput times, responsiveness increases on both application and server side.

An entity only consists of an ID for accessing components. The idea is to have no game code(behavior) inside of the components. The components don’t have to be located physically together with the entity, but should be easy to find and access using the entity. It is a common practice to use a unique ID for each entity. This is not a requirement, but it has several advantages:

\*The entity can be referred using the ID rather than a pointer. This is more robust, as it would allow for the entity to be destroyed without leaving dangling pointers.

\*It helps for saving state externally. When the state is loaded again, there is no need for pointers to be constructed.

\*Data can be shuffled around in memory as required.

\*Entity IDs can be used when communicating over a network to uniquely identity the entity.

Some of these advantages can also be achieved using smart pointers.

The normal way to transmit data between systems is to store the data in components, and then have each system access the component sequentially.

# Shader optimization

\* Describe a simple optimization for the following shader (5 points)

```

float3 diff; // 0 or 1

float4 someValue;

if (diff.x == 1.0 || diff.y == 1.0 || diff.z == 1.0)

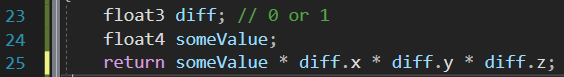
return someValue;

else

return float4(0.0);

```

My Optimized Solution is as below.



First of all, we should try to avoid the “if” “else” loop in Shader unless there are no alternative ways to change the performances. As the code given in the question, the return value should either be someValue itself or 0. Therefore, multiplication is an excellent method to optimize it, as in line 25 above.