selection sort code searches an array looking for the smallest element in the array. Then, the

smallest element is swapped with the first element of the array. The process is repeated for the

sub-array beginning with the second element of the array. Each pass of the array results in one

element being placed in its proper location. When the sub-array being processed contains one

element, the array is sorted. Write C++ code for this selection sort and output must be stored in

“output.txt” file (having all passes

selection sort code searches an array looking for the smallest element in the array. Then, the

smallest element is swapped with the first element of the array. The process is repeated for the

sub-array beginning with the second element of the array. Each pass of the array results in one

element being placed in its proper location. When the sub-array being processed contains one

element, the array is sorted. Write C++ code for this selection sort and output must be stored in

“output.txt” file (having all passes

# Department of Computing

**CS250: Data Structure and Algorithms**

**Class: BSCS 9AB**

**Course Instructor: Dr. Yasir Faheem**

# Lab 1: Pointers & Array-based Implementation of Lists

**Part One: Revision of Pointers in C++**

**Introduction**

This lab is about pointers, memory occupied by variables, and dynamic vs static memory allocation.

**Objectives**

This lab will revise the concepts learnt by students in the previous semesters.

**Tools/Software Requirement**

Visual Studio C++, Eclipse C++ or any other IDE.

**Description**

Pointers are used to point towards a particular memory address. In this lab, we will use the pointers and perform tasks with the help of them.

**Lab Tasks**

You are required to upload the lab tasks on the LMS and the name of that tasks must be in this format YourFullName\_reg#.cpp

Remember to comment your code properly. Inappropriate or no comment will result in the deduction of marks.

**Task 1:** Write output of the following C++ codes in your document without executing it.

**Example code a)**

int a;

int b;

int \*p=&a; // p assigned a address

int \*q=&b; // q assigned b address

a=20; // a assigned value 20

b=35; // b assigned value 35

p=q; // p assigned the address stored in q(same address as b)

\*p=83; //( the value changed from 35 to 83, as same address of p,q and b)

cout<<"a : "<<a<<" b: "<<b<<endl;

cout<<\*p<<" "<<\*q<<endl;

**Output:**

a : 20 b: 83

83 83

**Example code b)**

int x[4] = {0,4,6,9};

int \*p, a=3;

p=x;

(\*p)++; //0 to 1

cout<<\*p<<endl; //1

cout<<\*(p+1)<<endl; //4

p++;

\*p=\*p+a; // \*p=4+3

cout<<\*p<<endl; /7

p=p+2; //What is happening here? the current address of p is incremented by 2 places from index 1 to 3.

cout<<\*p<<endl; //9

**Output:**

1

4

7

9

<newline>

**Example code c)**

int a, \*p, \*q;

int arr[4]= {0};

p=arr; // assign index 0 of arr to p

q=p; // assign adress stored in p to q

\*p=4;

for(int i=0; i<3; i++){

a=\*p;

p++;// incremed index of array arr, having same address as p.

\*p=(a+i);

}

for (int j=0; j<4; j++){

cout<<\*q<<" ";// 4 4 5 7

q++; // incrementing index position of array, as address of q and p is same.

}

**Output:**

4 4 5 7

**Task 2:**

int a=5, b=10;

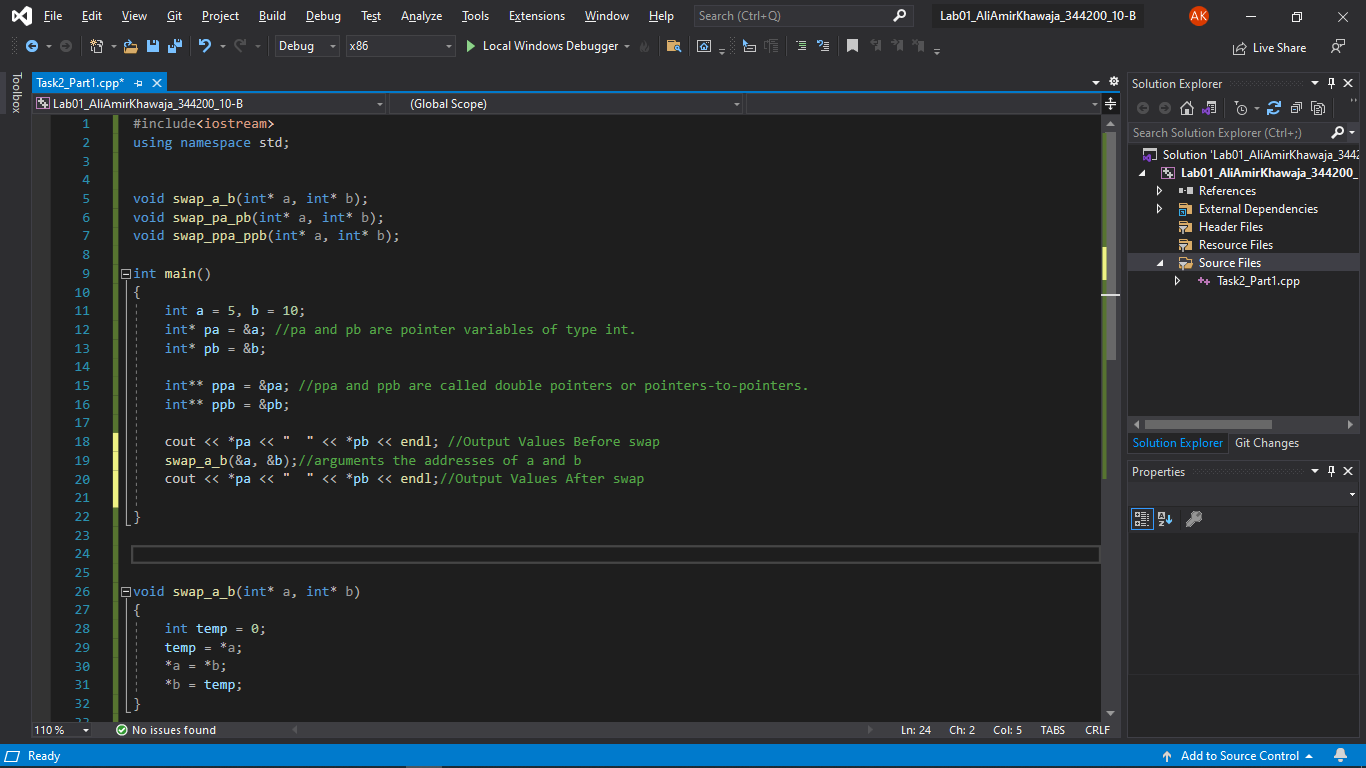
int \*pa=&a; //pa and pb are pointer variables of type int.

int \*pb=&b;

int \*\*ppa=&pa; //ppa and ppb are called double pointers or pointers-to-pointers.

int \*\*ppb=&pb;

1. Write code of a function that swaps values of variables a and b. Input to the function should be the address of both the variables.



Text

Description automatically generated

1. Write code of a function that swaps values of pointer variables pa and pb. Input to the function should be the address of both the pointer variables.

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

1. Write code of a function that swaps values of the variables a and b using pointer-to-pointer variables ppa and ppb.

A screenshot of a computer

Description automatically generated with medium confidence

Text

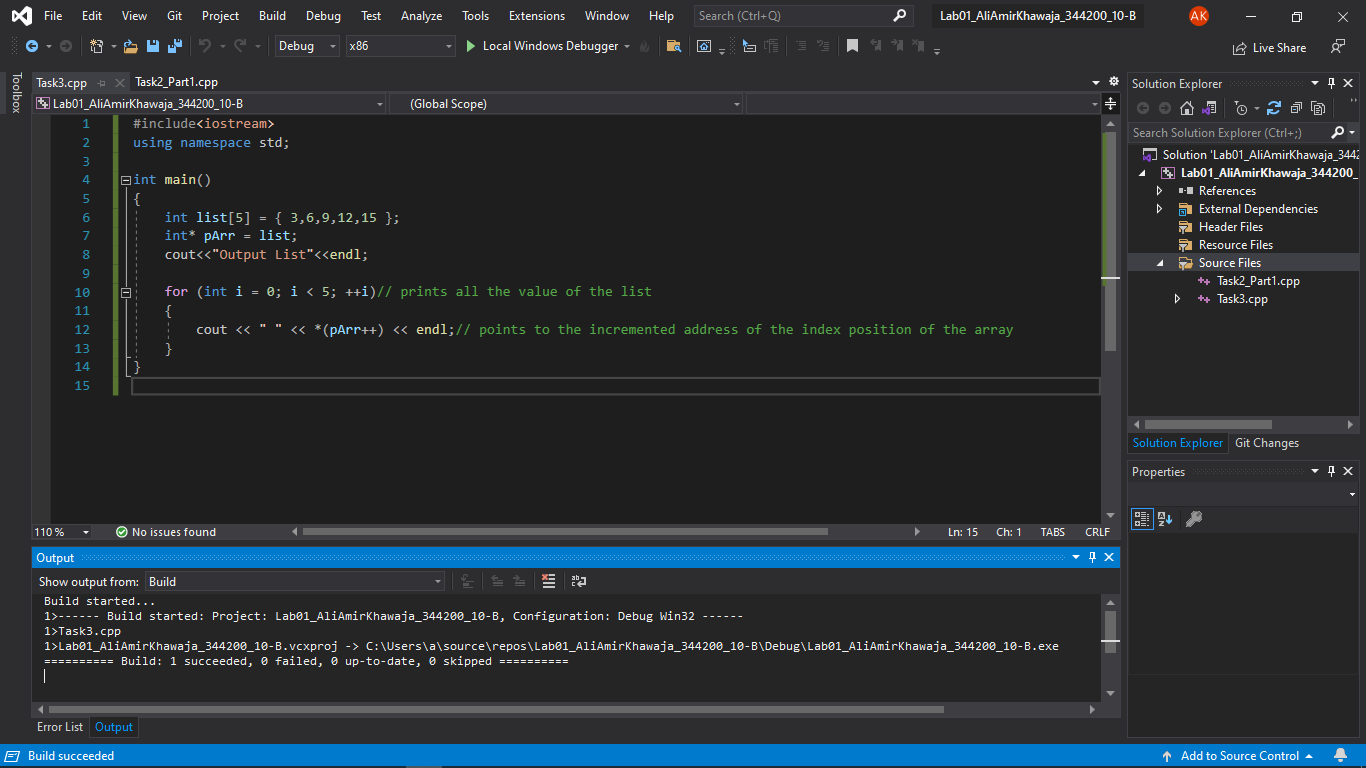
Description automatically generated

**Task 3:**

int list[5]={3,6,9,12,15};

int \*pArr= list;

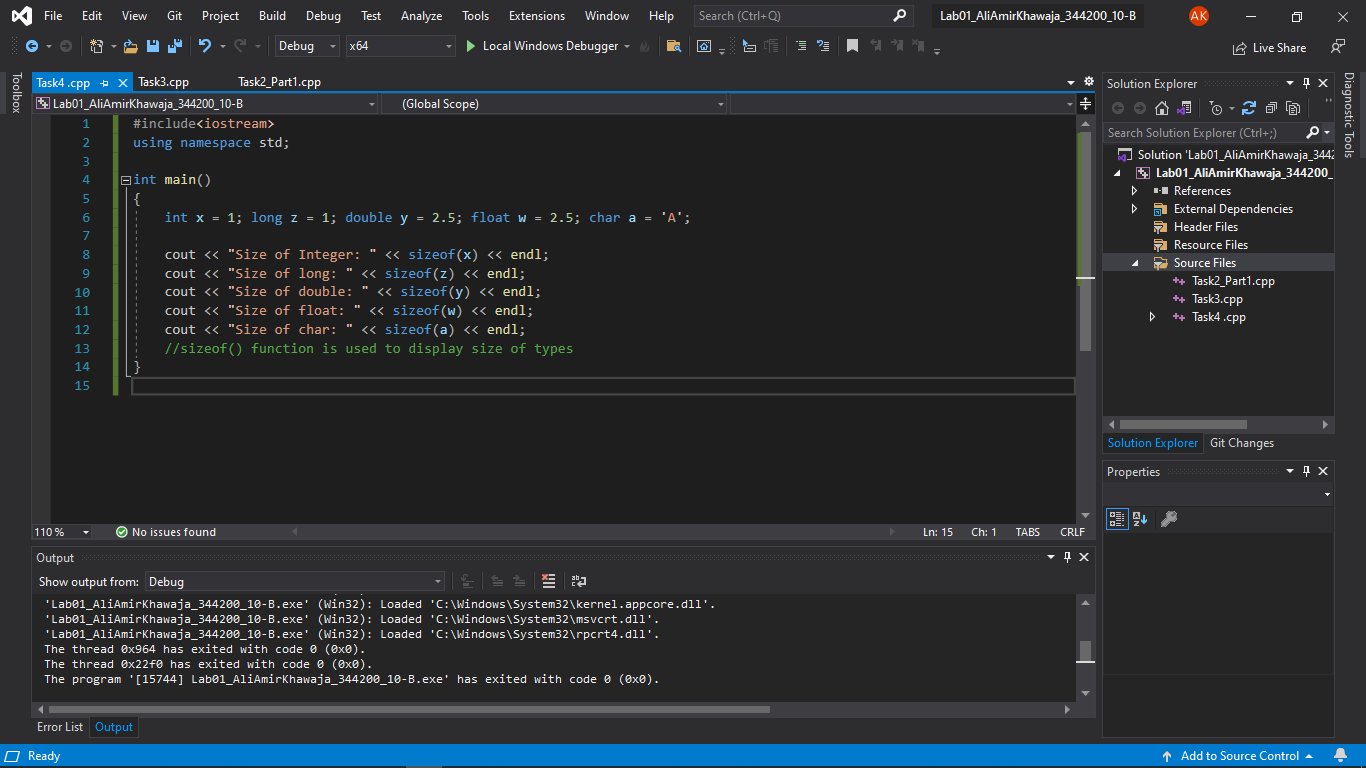
Your task is to write a piece of code that prints all values stored in the array **list** using only pointer variable pArr. Do not use the conventional way of printing values by numbering indexes.

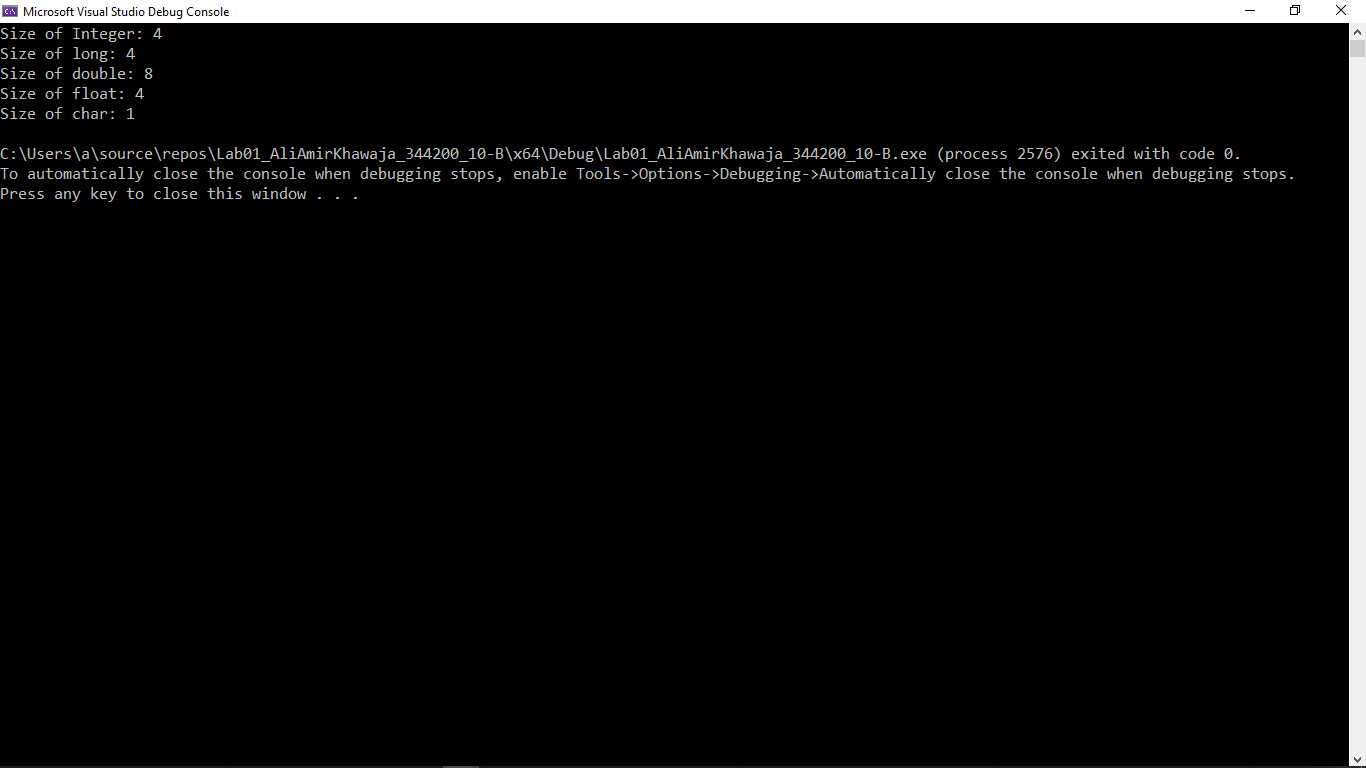


Text

Description automatically generated

**Task 4:** Write code to find the memory in bytes occupied by int, long, double, float and char.





**Recap: Static vs Dynamic Arrays**

Consider the two variants of declaring arrays below. Memory for the first variant gets allocated on the activation stack. The lifetime of an array created using method-A depends on its scope. If it is defined globally, its life is equal to the lifetime of the application. If it is declared in a function, memory for it gets allocated on the stack when the function gets called. It gets deallocated when the function call terminates. All the data related to the function call including the array gets removed from the stack. On the other hand, memory for the array created using new operator gets allocated on the heap at runtime. The lifetime of such an array is at max equal to the execution time of the application. If the array is not required during the execution of the program, the memory allocated to it can be freed using **delete []** command.

**Method A:**

const int size=5;

int x[size];

for (int i = 0; i < size; i++)

{

//cout << "x[" << i << "] = ";

x[i] = i + 1;

}

**Method B**

int size; // Note that size variable is const in variant A whereas it isn’t in variant B. Find out the logic behind it.

cout << "Enter size of array: ";

cin >> size;

int \*x = new int[size];

for (int i = 0; i < size; i++)

{

//cout << "x[" << i << "] = ";

x[i] = i + 1;

}

**Part Two: Array-based Implementation of List Data Structure**

**Introduction**

We have studied List as Abstract Data Type in the class. We have discussed various operations that can be performed on a list. The objective of this lab is to implement an array-based list. First, you shall implement static array-based list which is not resizable. Then, you shall implement a dynamic version of an array-list; you shall allocate memory for array at runtime using the **new** operator.

Your task is to implement all the following operations.

1. bool IsEmpty() operation. It returns true value if the list is empty. Otherwise, false.
2. bool isFull() function. It returns true value if the list is full. False otherwise.
3. InsertSorted(value) a value at its logical position in a list.
4. CreateRoom(int pos); If a new value has to be inserted into a list at position pos, this function creates room for it, by shifting all values from position pos till last one index to the right.
5. FillGap(int pos); If an existing value has to be deleted from position pos, this function fills the gap by shifting all values from pos till last element to the left side.
6. DynamicExpansion( ); If an array-list is already full, this function dynamically creates a new array of size greater than the current size of the array, copies all contents from the older array to the new one, updates the pointer which stores the address of an array, and lastly deletes the older array using **delete** ptr[] command.
7. DynamicReduction(); If the size of a list falls below 50% capacity of the array after a deletion operation, this function dynamically creates a new array of size smaller than the current size of the existing array, copies all contents from the older array to the new one, updates the pointer that stores the address of an array, and lastly deletes the older bigger array using **delete** ptr[] command.

**Task 1: Implement a version of an array-list in which the array is static.**

struct staticArrayList{

int array[size];

bool isEmpty();

bool isFull();

void insertAtposition(int value, int position);

void insertSorted(int value);

void printList();

void DeleteValue(int value);

void DeleteValueAtPosition(int pos); // deletes a value if a user enters a valid position (1 <= pos <= size of a list.)

**Note:** The size of a list and that of an array are different things. Moreover, an element at position pos in a list is stored at index pos-1 in an array.

}

Code:

A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence

Output:

A picture containing shape

Description automatically generated A picture containing text

Description automatically generated

**Task 2: Implement a version of an array-list in which the array is dynamic**. Memory for an array of capacity *size*can be allocated at runtime using new operator. It can be deallocated using **delete** operator. Moreover, array-list can be resized if it becomes full, or the number of elements in it fall below a certain threshold. In addition, in this version of the program you shall assume that we are maintaining a list sorted in the ascending order; InsertAtEnd() and InsertAtPosition() functions are no longer required in this version of the program. You may implement a new function named InsertSorted(newValue). It should find the logical position of the newValue in the list, create room for it, if required, by calling the makeRoom(pos) function, and then insert it.

struct DynamicArrayList{

int \*pArray[];

const int size;

pArrayList = new int [size];

int array[size];

bool isEmpty();

bool **isArrayFull();**

void insertAtposition(int value, int position);

void insertSorted(int value);

void printList();

void DeleteValue(int value);

void DeleteValueAtPosition(int pos); // deletes a value if a user enters a valid position (1 <= pos <= size of a list)

void DynamicExpansion(); //Increases size of the array when the list becomes full.

void DynamicReduction(); //decreases size of the array when the size of a list becomes less than 50% after deletion operation

}

Code:

A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Output

Text

Description automatically generated