**TASK**

**Deadline Thursday (8th Feb, 2018)**

1. Submit home tasks of Lecture no 5
2. Write c++ code to print elements of 1D array using pointer.

Int array[5]={1,2,3,4,5}

1. Output?

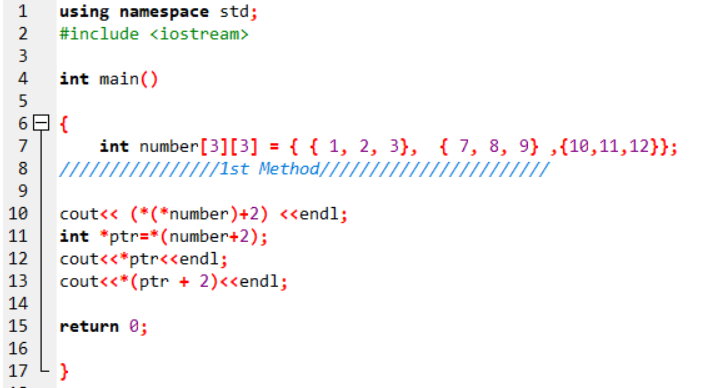
int number[2][6] = { { 1, 2, 3, 4, 5, 6 }, { 7, 8, 9, 10, 11, 12 }};

cout<<\*(\*(number + 1) + 2);

int \*ptr= \*(number +1);

cout<<\*ptr;

1. Output??



1. Output??

int \*ptrr [2]={matrix[0],matrix[1]….. matrix[r ]};

int \*\*ptrc=ptrr;

cout<<ptrc;

coutt<<\*\*ptrc;

Cout<<\*(ptrc+1)

1. Write a C++ code to add all the contents of 2nd row of a 4 x 5 array using **pointer notation only** in a variable **sum**.
2. **Use pointers and addresses to modify the code so x is passed by reference (pass address of x in function) instead and is squared. This will involve changes to the square function that *does not* involve changing void to int and giving square a return statement.**

#include <stdio.h>

void square(int num)

{ num = num \* num; }

int main()

{ int x = 4;

square(x);

cout<< x;

return 0; }

1. Answer the following questions:
2. What types of pointers are num, odd and prime?
3. After execution of first 7 lines, is q++ a valid statement? Explain in detail why or why not.
4. Can we increment the value of every element of num using p pointer. Explain in detail why or why not. (Note: Do not use num to increment in the array)
5. Name the pointers in line 5, 6 and 7.
6. p and r both points to a constant integer. Similarly, p can point to the constant r is pointing but vice versa is not possible. Why?
7. p can point to the variable var, but after pointing we cannot change the value of var using p. Explain why?

#include <iostream>

using namespace std;

int main()

{

1) int var = 10;

2) const int odd[5] = { 1, 3, 5, 7, 9 };

3) int num[5] = { 1, 2, 3, 4, 5 };

4) int prime[5] = { 2, 3, 5, 7, 11 };

5) int const\* p = num;

6) int\* const q = prime;

7) int const\* const r = odd;

8) return 0;

}

1. Write a c++ code that will copy from one string to another string using pointers

char\* str1 = “Practice makes perfect";

char str2[80]; //empty string

copystr(str2, str1); //copy str1 to str2

Void copystr(char\* dest, const char\* src)

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

//Sort an array A using insertion sort. Notice it is to be passed by reference.

void sort(/\* what's the argument data type?\*/ A, int n){

int tmp;

int i;

int j;

for(i=1; i<n; i++){

tmp = A[i];

j=i-1;

//For each element in A, search for where it belong in the subarray preceeding it's current location

while(tmp<A[j] && j>=0){

A[j+1]=A[j];

j-=1;

}

A[j+1]=tmp;

}

}

int main(){

//Allows use to generate random numbers

srand(time(NULL));

//Read a user input integer and store it in n

int n;

cout<<”Enter an integer n: ";

cin>>n;

int array[n];

//Assign each element in the array a random number between 0 and 31,999

int i;

for (i=0; i<n; i++){

array[i]=rand()%32000; //This line assigns random numbers

}

//Prints out the elements of the unsorted array

int x;

cout<<”The unsorted array is:”;

for (x=0; x<n; x++){

cout<<array[x;

}

//Calls the sort function to sort the array

sort(array,n);

//Print out the elements of the now (supposedly) sorted array.

cout<<”The sorted array is: ";

for (x=0; x<n; x++){

cout<<array[x];

}

return 0;

}

In sort.c, I've implemented a basic implementation of insertion sort (not too efficient, but a very simple sorting algorithm). Look at and understand the code (read comments), and put the proper argument data type for the sort function's first argument. Compile and run the code to make sure it works (it sorts the numbers).

Now, replace all array index access (places where I access the array with [], such as in A[i]) in the entire program by using pointer addition instead.