OOP Assignment 3

Ian Whittemore

1700870

**Assignment 3 submission**

|  |  |  |
| --- | --- | --- |
| **Task** | **Mark allocated** | **Your mark** |
| **Question 1** | **30** |  |
| Class cpair design | **10** |  |
| Right use of constructor | **5** |  |
| Add function modification | **5** |  |
| The use of Add template function | **5** |  |
| Testing | **5** |  |
| **Question 2** | **70** |  |
| Section 1 | **30** |  |
| Section 2 | **10** |  |
| Section 3 | **10** |  |
| Section 4 | **10** |  |
| Testing | **10** |  |
| **Total** | **100** |  |

**Submission instructions:**

1. You must include this page as a first page in your submission.
2. You must insert all your code in this word document for the TWO questions.
3. You need to include screen shoots for your output as appropriate in your word document.
4. Each question should start in a new page.

**Question 1:**

The code in Figure 1 is representing a class definition of class “cpair” that has two integer numbers and an “Add” function that can receive two objects of the class “cpair” and return the sum of these two objects in a referenced parameter.

Answer the following:

1. Modify the class “cpair” and the function “Add” in Figure 1 to be a template class and function. This will make the function receives the cpair objects with different types (i.e. int, float, double, char, etc.).
2. Write a code that can represent the calling of the template function when used with different type of date.
3. Modify the Add function in figure 1 to represent operator overloading. Give a suitable example to call the function.

|  |  |
| --- | --- |
| #include "stdafx.h"  #include <iostream>  using namespace std;  class cpair  {  public:  cpair(int x=0, int y=0) {A=x; B=y;}  void print()  {cout << A <<" "<<B<<endl;}  int A, B;  } ; | void Add(cpair A1, cpair A2, cpair &R)  {  R.A= A1.A + A2.A;  R.B= A1.B + A2.B;  }  int \_tmain(int argc, \_TCHAR\* argv[])  {  cpair A1(4,5), A2(1,3), result;  Add(A1, A2, result);  result.print();  return 0;  } |

Figure 1

**Question 2:**

The Figures below represent three data structures **designed using the C++ STL** library.

**Section 1:**

Design a structure for each of the three structures shown in figure 1, 2, and 3 below.

**Section 2:**

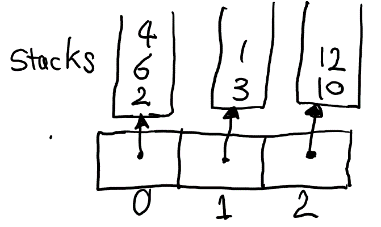
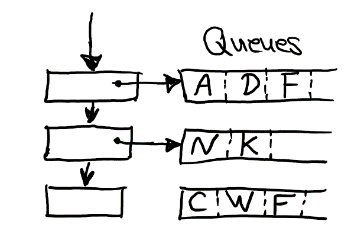
Compare the searching performance of the structures shown in Figure 3 and Figure 4.

**Section 3:**

Populate the data shown in each of the figures below as appropriate in their relevant structure.

**Section 4:**

Write a method for each structure to output the data to the console.



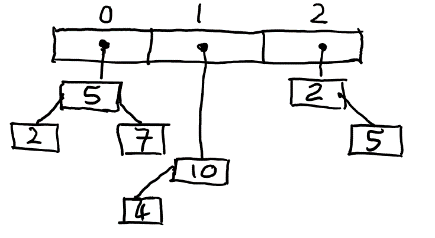




Figure 2 Figure 3 Figure 4

Question 1

#include "stdafx.h"

#include <iostream>

using namespace std;

template <class T>

class cpair

{

public:

    cpair(T x=0, T y=0) {A=x; B=y;}

    void print()

          {cout << A <<" "<<B<<endl;}

    T A, B;

       void Add(cpair a,cpair b);

          cpair operator+(cpair);

} ;

template <class T>

void cpair<T>::Add(cpair A1, cpair A2)

{

       A= A1.A + A2.A;

          B= A1.B + A2.B;

}

template <class T>

cpair<T> cpair<T>::operator+(cpair A1)

{

       cpair result;

       result.A = A + A1.A;

       result.B = B + A1.B;

       return result;

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

       cpair<int>A1(3.1,5.2),A2(1.2,3.6),R;

       R.Add(A1, A2);

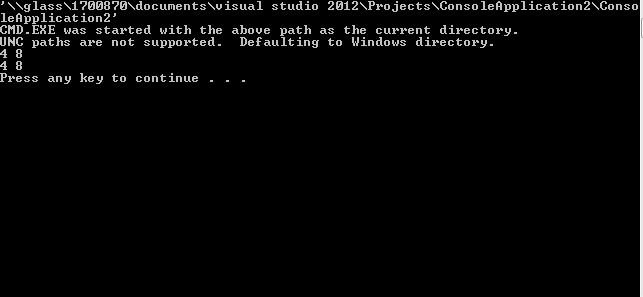
       R.print();

          R=A1+A2;

          R.print();

       return 0;

}



Question 2

#include "stdafx.h"

#include <iostream>

#include <string>

#include <time.h>

#include <math.h>

#include <set>

#include <map>

#include <algorithm>

#include <vector>

#include <stack>

#include <queue>

#include <list>

using namespace std;

template < class T>

void popElements(T&s)

{

       while (!s.empty()){

              cout<<s.top()<<" ";

              s.pop();

       }

}

template <class T>

void popElementsSet(set <T> &s)

{

       for (auto p=s.begin();p!=s.end();p++)

       {

              cout<<\*p;

       }

}

template < class T>

void popElementsQueue(T&s)

{

              for (auto h=s.begin();h!=s.end();h++)

       {

                        queue<char>queuechar1;

                        queuechar1=h;

       }

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

       vector<stack<int>>vecstack(3);

       list<queue<char>>queuechar;

          queue<char>queuechar1;

       queue<char>queuechar2;

          queue<char>queuechar3;

       vector<set<int>>veclist(3);

       int \*ptr;

    //Question2.1

      vecstack[0].push(3);

       vecstack[0].push(7);

       vecstack[0].push(2);

       vecstack[1].push(1);

       vecstack[1].push(4);

       vecstack[2].push(8);

       vecstack[2].push(9);

              popElements(vecstack[0]);

              cout<<endl;

              popElements(vecstack[1]);

              cout<<endl;

              popElements(vecstack[2]);

              cout<<endl;

       //Question 2.2

          queuechar1.push('W');

          queuechar1.push('T');

          queuechar1.push('S');

          queuechar2.push('L');

          queuechar2.push('N');

          queuechar3.push('P');

          queuechar3.push('Q');

          queuechar3.push('Z');

                queuechar.push\_back(queuechar1);

          cout<<endl;

          queuechar.push\_back(queuechar2);

          cout<<endl;

          queuechar.push\_back(queuechar3);

          cout<<endl;

          queuechar.pop\_back();

       //Question2.3

       veclist[0].insert(2);

       veclist[0].insert(3);

       veclist[0].insert(5);

       veclist[1].insert(7);

       veclist[1].insert(2);

       veclist[2].insert(5);

       veclist[2].insert(2);

          popElementsSet(veclist[0]);

          cout<<endl;

          popElementsSet(veclist[1]);

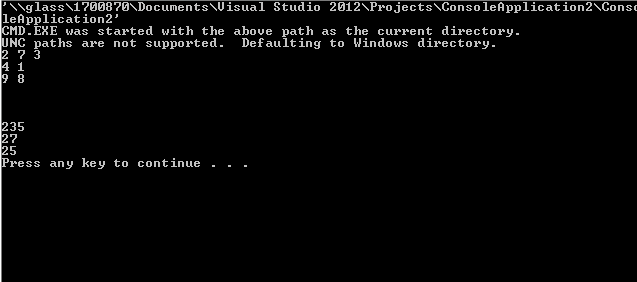
          cout<<endl;

          popElementsSet(veclist[2]);

          cout<<endl;

       return 0;

}



Section 2

The one that is better between figure 3 and 4 depends on what the user is using it for. Figure 3 is a list of queues and figure 4 is a vector of multiset. Figure 3 is known to be slow for looking up data, but this is useful if you want to find specific information and if it requires an unknown number of allocated spaces. Figure 4 is useful for large varieties of data, and will be able to have multiple fields allocated to the vector multiset, but needs a specific amount of allocated space, otherwise it will copy itself to expand itself but this creates a problem with performance if needed. So, the answer depends on what the user is performing.