**Logo

Description automatically generated San Francisco Bay University**

**CS360 - Programming in C and C++**

**2022 Summer Final Exam**

**Student Name: Student ID:**

1. Various situations require that pairs of numbers be treated as a unit. For example, each screen coordinate has an *x* (horizontal) component and a *y* (vertical) component. Represent such a pair of numbers as a structure called pair that comprises two *int* member variables.

Now, assuming you want to be able to store *pair* variables on a stack, that is, you want to be able to place a pair (which contains two integers) onto a stack using a single call to a *push()* function with a structure of type *pair* as an argument, and retrieve a pair using a single call to a *pop()* function, which will return a structure of type *pair*. Start with the *Stack2* class in the following program, and from it, please derives a new class called *pairStack*. This new class need contain only two members: the overloaded *push()* and *pop()* functions. The *pairStack::push()* function will need to make two calls to *Stack2::push()* to store the two integers in its pair, and the *pairStack::pop()* function will need to make two calls to *Stack2::pop()* (although not necessarily in the same order).

*// staken.cpp*

*// overloading functions in base and derived classes*

*#include <iostream>*

*using namespace std;*

*#include <cstdlib> //for exit()*

*////////////////////////////////////////////////////////////////*

*class Stack{*

*protected: //NOTE: can’t be private*

*enum { MAX = 3 }; //size of stack array*

*int st[MAX]; //stack: array of integers*

*int top; //index to top of stack*

*public:*

*Stack() //constructor*

*{ top = -1; }*

*void push(int var) //put number on stack*

*{ st[++top] = var; }*

*int pop() //take number off stack*

*{ return st[top--]; }*

*};*

*////////////////////////////////////////////////////////////////*

*class Stack2 : public Stack*

*{*

*public:*

*void push(int var) //put number on stack*

*{*

*if(top >= MAX-1) //error if stack full*

*{ cout << "\nError: stack is full"; exit(1); }*

*Stack::push(var); //call push() in Stack class*

*}*

*int pop() //take number off stack*

*{*

*if(top < 0) //error if stack empty*

*{ cout << "\nError: stack is empty\n"; exit(1); }*

*return Stack::pop(); //call pop() in Stack class*

*}*

*};*

*////////////////////////////////////////////////////////////////*

*int main(){*

*Stack2 s1;*

*s1.push(11); //push some values onto stack*

*s1.push(22);*

*s1.push(33);*

*cout << endl << s1.pop(); //pop some values from stack*

*cout << endl << s1.pop();*

*cout << endl << s1.pop();*

*cout << endl << s1.pop(); //oops, popped one too many...*

*cout << endl;*

*return 0;*

*}*

1. Start with the *Distance* class in the following program. Using a loop similar to that in the *DISKFUN* example as below, get a number of *Distance* values from the user, and write them to a disk file. Append them to existing values in the file, if any. When the user signals that no more values will be input, read the file and display all the values.

// englcon.cpp

// constructors, adds objects using member function

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

class Distance //English Distance class

{

  private:

    int feet;

    float inches;

  public: //constructor (no args)

    Distance() : feet(0), inches(0.0)

    { }

    //constructor (two args)

    Distance(int ft, float in) : feet(ft), inches(in)

    { }

    void getdist() //get length from user

    {

      cout << "\nEnter feet: "; cin >> feet;

      cout << "Enter inches: "; cin >> inches;

    }

    void showdist() //display distance

    { cout << feet << "\'-" << inches << '\"'; }

    void add\_dist( Distance, Distance ); //declaration

};

//--------------------------------------------------------------

//add lengths d2 and d3

void Distance::add\_dist(Distance d2, Distance d3)

{

  inches = d2.inches + d3.inches; //add the inches

  feet = 0; //(for possible carry)

  if(inches >= 12.0) //if total exceeds 12.0,

  { //then decrease inches

    inches -= 12.0; //by 12.0 and

    feet++; //increase feet

  } //by 1

  feet += d2.feet + d3.feet; //add the feet

}

////////////////////////////////////////////////////////////////

int main()

{

  Distance dist1, dist3; //define two lengths

  Distance dist2(11, 6.25); //define and initialize dist2

  dist1.getdist(); //get dist1 from user

  dist3.add\_dist(dist1, dist2); //dist3 = dist1 + dist2

  //display all lengths

  cout << "\ndist1 = "; dist1.showdist();

  cout << "\ndist2 = "; dist2.showdist();

  cout << "\ndist3 = "; dist3.showdist();

  cout << endl;

  return 0;

}

// diskfun.cpp

// reads and writes several objects to disk

#include <fstream> //for file streams

#include <iostream>

using namespace std;

////////////////////////////////////////////////////////////////

class person //class of persons

{

  protected:

    char name[80]; //person’s name

    int age; //person’s age

  public:

    void getData() //get person’s data

    {

      cout << "\n Enter name: "; cin >> name;

      cout << " Enter age: "; cin >> age;

    }

    void showData() //display person’s data

    {

      cout << "\n Name: " << name;

      cout << "\n Age: " << age;

    }

};

////////////////////////////////////////////////////////////////

int main()

{

  char ch;

  person pers; //create person object

  fstream file; //create input/output file

  //open for append

  file.open("GROUP.DAT", ios::app | ios::out |

  ios::in | ios::binary );

  do //data from user to file

  {

    cout << "\nEnter person’s data:";

    pers.getData(); //get one person’s data

    //write to file

    file.write( reinterpret\_cast<char\*>(&pers), sizeof(pers) );

    cout << "Enter another person (y/n)? ";

    cin >> ch;

  }

  while(ch=='y'); //quit on ‘n’

    file.seekg(0); //reset to start of file

  //read first person

  file.read( reinterpret\_cast<char\*>(&pers), sizeof(pers) );

  while( !file.eof() ) //quit on EOF

  {

    cout << "\nPerson:"; //display person

    pers.showData(); //read another person

    file.read( reinterpret\_cast<char\*>(&pers), sizeof(pers) );

  }

  cout << endl;

  return 0;

}

1. Start with the *safearay* class in the following example. Make this class into a template, so the safe array can store any kind of data. In *main()*, create safe arrays of at least two different types, and store some data in them.

// arrover3.cpp

// creates safe array (index values are checked before access)

// uses overloaded [] operator for both put and get

#include <iostream>

using namespace std;

#include <cstdlib> //for exit()

const int LIMIT = 100; //array size

////////////////////////////////////////////////////////////////

class safearay{

  private:

    int arr[LIMIT];

  public:

    int& operator [](int n) //note: return by reference

    {

      if( n< 0 || n>=LIMIT )

      { cout << "\nIndex out of bounds"; exit(1); }

      return arr[n];

    }

};

////////////////////////////////////////////////////////////////

int main(){

  safearay sa1;

  for(int j=0; j<LIMIT; j++) //insert elements

    sa1[j] = j\*10; //\*left\* side of equal sign

  for(int j=0; j<LIMIT; j++) //display elements

  {

    int temp = sa1[j]; //\*right\* side of equal sign

    cout << "Element " << j << " is " << temp << endl;

  }

  return 0;

}