Astrid Lozano

COSC 1337 (59606)

MIDTERM

**QUESTION 1. Strings and vectors (dna)**

**“dnavector.h”**

#include <vector>

#include <string>

std::vector<int> lettercount(std::string s);

**“dnavector.cpp”**

#include<vector>

#include<string>

#include "dnavector.h"

std::vector<int> lettercount(std::string s)

{

auto l = s.size() - 1;

std::vector<int> dnacount{};

int counta = 0;

int countt = 0;

int countc = 0;

int countg = 0;

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

{

if (s[i] == 'A')

counta += 1;

else if (s[i] == 'T')

countt += 1;

else if (s[i] == 'C')

countc += 1;

else if (s[i] == 'G')

countg += 1;

}

dnacount.push\_back(counta);

dnacount.push\_back(countt);

dnacount.push\_back(countc);

dnacount.push\_back(countg);

return dnacount;

}

**“main.cpp”**

#include<vector>

#include<string>

#include<iostream>

#include "dnavector.h"

int main()

{

std::string s = "AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC";

std::vector<int> dna = lettercount(s);

std::cout << "A: " << dna[0] << std::endl;

std::cout << "T: " << dna[1] << std::endl;

std::cout << "C: " << dna[2] << std::endl;

std::cout << "G: " << dna[3] << std::endl;

return 0;

}

**Test case**

#define CATCH\_CONFIG\_MAIN

#include #include "catch.cpp"

#include "dnavector.h"

TEST\_CASE("test dna string vector")

{

vector<int> count {1,2,21};

REQUIRE(lettercount("AGTCCT") == count);

}

**QUESTION #2**

**grade.h**

#include <vector>

#include <iostream>

void function(std::vector<int> grade);

class Countgrade

{

public:

Countgrade() = default;

Countgrade(std::vector<int> number) : grade(number) {}

void count(std::vector<int> grade);

private:

std::vector<int> grade{};

std::vector<int> cuenta{};

};

**Grade.cpp**

#include "grade.h"

#include <string>

#include <iostream>

#include <iomanip>

using std::setw; using std::endl; using std::cout;

void Countgrade::count(std::vector<int> grade)

{

int counta = 0;

int countb = 0;

int countc = 0;

int countd = 0;

int countf = 0;

auto s = grade.size();

for (auto i = 0; i < s; i++)

{

if (grade[i] >= 90)

counta += 1;

else if (grade[i] < 90 && grade[i] >= 80)

countb += 1;

else if (grade[i] < 80 && grade[i] >= 70)

countc += 1;

else if (grade[i] < 70 && grade[i] >= 60)

countd += 1;

else if (grade[i] < 60 && grade[i] >= 0)

countf +=1;

}

std::vector<int> cuenta = { counta, countb, countc, countd, countf };

cout << "grade" << setw(10) << "count" << endl;

cout << "A" << setw(10) << cuenta[0] << endl;

cout << "B" << setw(10) << cuenta[1] << endl;

cout << "C" << setw(10) << cuenta[2] << endl;

cout << "D" << setw(10) << cuenta[3] << endl;

cout << "F" << setw(10) << cuenta[4] << endl;

}

**Main.cpp**

#include "grade.h"

#include <vector>

#include <iostream>

#include <iomanip>

using std::endl;

int main()

{

std::vector<int> calificacion{ 95,62,86,74,52,98,93,94,83,85,81,75,79,77,72,67,63,61,58,58 };

Countgrade num;

num.count(calificacion);

}

QUESTION 3

**"receipt.h"**

#ifndef RECEIPT\_H

#define RECEIPT\_H

#include <iostream>

class Receipt

{

public:

Receipt() : amount(0) {};

Receipt(double a) : amount(a) {};

Receipt operator+=(Receipt& r);

friend std::ostream & operator<<(std::ostream& out, Receipt& r);

friend std::istream & operator>>(std::istream& in, Receipt& r);

void tally();

private:

double amount;

double taxes;

double tips;

double taxrate = .1;

double tiprate = .2;

double total{ 0 };

};

#endif //RECEIPT\_H

**“receipt.cpp”**

#include "receipt.h"

#include <iostream>

std::istream & operator>>(std::istream& in, Receipt& r)

{

std::cout << "enter the amount of your bill: " << std::endl;

in >> r.amount;

return in;

}

std::ostream & operator<<(std::ostream& out, Receipt& r)

{

out << "Bill: " << r.amount << std::endl;

out << "Tip: " << r.tips << std::endl;

out << "Tax: " << r.taxes << std::endl;

out << "Total: " << r.total;

return out;

}

Receipt Receipt::operator+=(Receipt& r)

{

taxes = taxrate \* amount;

tips = amount \* tiprate;

taxes += r.taxes;

tips += r.tips;

amount += r.amount;

total += r.amount + r.tips + r.taxes;

return Receipt(amount);

}

void Receipt::tally()

{

taxes = taxrate \* amount;

tips = amount \* tiprate;

total += amount + taxes + tips;

}

“main.cpp”

#include<vector>

#include<string>

#include<iostream>

#include "receipt.h"

using std::vector;

int main()

{

Receipt r, rr, rrr, result;

vector<Receipt> recibos{ r,rr,rrr };

int number;

std::cout << "how many receipts you want to include (up to three): " << std::endl;

std::cin >> number;

for (int j = 0; j < number; j++)

{

std::cin >> recibos[j];

recibos[j].tally();

result += recibos[j];

}

std::cout << result;

}

TEST CASE

#define CATCH\_CONFIG\_MAIN

#include #include "catch.cpp"

#include "receipt.h"

TEST\_CASE("test total amount is correct")

{

Receipts a(10), b(10), suma;

vector<Receipts> test{a,b};

test[0].tally();

test[1].tally();

suma += test[0];

suma += test[1];

REQUIRE(suma.total == 26);

}

**CLASS INHERITANCE & VIRTUAL FUNCTIONS**

#ifndef RECEIPT\_H

#define RECEIPT\_H

#include <iostream>

class Shape

{

public:

Shape() = default;

virtual void draw();

protected:

measure;

};

class Line : public Shape

{

public:

Line() = default;

void draw();

protected:

double length;

};

class Circle : public Shape

{

public:

Circle() = default;

void draw();

protected:

double radius;

};

#endif //HEADER\_H

**.cpp**

#include "Header.h"

#include <iostream>

void Shape::draw() {}

void Line::draw() {std::cout <<"line"; }

void Circle::draw() {std::cout <<"circle"; }

**main.cpp**

#include "Header.h"

#include <iostream>

int main()

{

Line line;

Shape& shape = line;

shape.draw();

Circle c;

Shape& shape\_c = c;

shape\_c.draw();

}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Shape |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  | #measure |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  | + virtual draw(): void |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | Line |  |  |  | Shape |  |
|  |  |  |  |  |  |  |
|  | # length: double |  |  |  |  |  |
|  |  |  |  |  | # radius: double |  |
|  |  |  |  |  |  |  |
|  | + draw(): void |  |  |  | + draw(): void |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**REFERECES**

**1.**

**.H**

#include <string>

#include <iostream>

void function(std::string& str\_by\_ref);

**.CPP**

#include "Header.h"

#include <string>

#include <iostream>

using std::string;

void function(std::string& str\_by\_ref)

{

auto s = str\_by\_ref.size();

string reverse(s, ' ');

int i = 0;

while (str\_by\_ref[i] != '\0')

{

reverse[i] = str\_by\_ref[s -1];

s = s - 1;

i = i + 1;

}

str\_by\_ref = reverse;

}

**MAIN**

#include "Header.h"

#include <string>

#include <iostream>

int main()

{

std::string str\_by\_ref = "qwerty";

std::cout << str\_by\_ref;

function(str\_by\_ref);

std::cout << str\_by\_ref;

}

**OUTPUT: qwerty ytrewq**

**2.**

**.H**

#include <string>

#include <iostream>

std::string function(std::string str\_by\_ref);

**.cpp**

#include "Header.h"

#include <string>

#include <iostream>

std::string function(std::string str\_by\_ref)

{

auto s = str\_by\_ref.size();

std::string reverse(s, ' ');

int i = 0;

while (str\_by\_ref[i] != '\0')

{

reverse[i] = str\_by\_ref[s -1];

s = s - 1;

i = i + 1;

}

return reverse;

}

**MAIN**

**TWO CASES:**

**If:**

**int main ()**

**{**

std::string str\_by\_ref = "qwerty";

std::cout << str\_by\_ref;

function(str\_by\_ref);

std::cout << str\_by\_ref; //just calling the string

}

**OUTPUT:**

**qwerty qwerty**

**if**

int main()

{

std::string str\_by\_ref = "qwerty";

std::cout << str\_by\_ref;

std::string s = function(str\_by\_ref);

std::cout << s; //calling the output of the function, displaying the return value

}

**Output is qwerty ytrewq**

3.

**.H**

#include <string>

#include <iostream>

void function(const std::string& str\_by\_ref);

**.cpp**

void function(const std::string& str\_by\_ref)

{

auto s = str\_by\_ref.size();

std::string reverse(s, ' ');

int i = 0;

while (str\_by\_ref[i] != '\0')

{

reverse[i] = str\_by\_ref[s];

s = s - 1;

i = i + 1;

}

str\_by\_ref = reverse;

}

No output, this error occurs:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Severity | Code | Description | Project | File | Line | Suppression State |
| Error | C2678 | binary '=': no operator found which takes a left-hand operand of type 'const std::string' (or there is no acceptable conversion) | Project1 | c:\users\5960606\source\repos\project1\project1\functions.cpp | 19 |  |

**Look at this code:**

class Shape

{

public:

virtual void draw() {}

};

class Line : public Shape

{

public:

void draw() { std::cout << "Line"; }

};

class Circle : public Shape

{

public:

void draw() { std::cout << "Circle"; }

};

int main()

{

Line line;

Shape& shape = line;

shape.draw();

Circle c;

Shape& shape\_c = c;

shape\_c.draw();

}

**What will the output be for line.draw() and shape\_c.draw()? Explain the rationale for the output.**

**Output is:**

**Line Circle**

**(the virtual function allows the draw methods of derived classes to override the draw method of shape class)**