Computer Programming Solutions 2004

Problem #1

```
#include <iostream>
#include <list>

using namespace std;

int main(int argc, char* argv[])
{
	int base10, base;

	cout<<"Integer (base10): ";
	cin>>base10;

	cout<<"Base: ";
	cin>>base;

	int digit;
	list<char> newNum;
	while(base10 > 0)
	{
		digit = base10 % base;
		if(digit < 10) {
			newNum.push_front(char('0' + digit)); }
		else {
			newNum.push_front(char('A' + digit - 10)); }

		base10 = (int)(base10 / base);
	}

	list<char>::iterator numIter = newNum.begin();
	for( ; numIter != newNum.end(); ++numIter) {
		cout<<*numIter; }
	cout<<endl;

	return 0;
}
```

Problem #2

```
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char* argv[])
{
	char num[256];
```

```
	int base;

	cout<<"Integer: ";
	cin>>num;

	cout<<"Base: ";
	cin>>base;

	string num_s(num);
	int place = 1, newNum = 0;
	char digit;
	string::reverse_iterator numIter = num_s.rbegin();
	for( ; numIter != num_s.rend(); ++numIter)
	{
		digit = *numIter;
		if(digit <= '9' && digit >= '0') {
			newNum += place * (digit - '0'); }
		else {
			newNum += place * (digit - 'A'); }

		place *= base;
	}
	cout<<newNum<<endl;

	return 0;
}
```

Problem #3

```
#include <iostream>
#include <cmath>

using namespace std;

int findPrimeFactor(int num)
{
	int sqrtValue = (int)sqrt((double)num) + 1;
	for(int i = 2; i<sqrtValue; i++)
	{
		if(i == num) {
			return num; }

		if(num % i == 0) {
			return findPrimeFactor(i); }
	}

	return num;
}

int main(int argc, char* argv[])
{
	int num;
```

```
	cout<<"Input: ";
	cin>>num;

	int factor;
	while(num > 1)
	{
		factor = findPrimeFactor(num);
		cout<<factor;
		num /= factor;
		if(num > 1) {
			cout<<"*"; }
	}
	cout<<endl;

	return 0;
}
```

Problem #4

```
#include <iostream>

using namespace std;

int main(int argc, char* argv[])
{
	double u[3], v[3];
	cout<<"A = ";
	cin>>u[0];
	cout<<"B = ";
	cin>>u[1];
	cout<<"C = ";
	cin>>u[2];
	cout<<"D = ";
	cin>>v[0];
	cout<<"E = ";
	cin>>v[1];
	cout<<"F = ";
	cin>>v[2];

	double dot, sub[3], cross[3], mult[3];

	sub[0] = u[0] - v[0];
	sub[1] = u[1] - v[1];
	sub[2] = u[2] - v[2];

	cross[0] =   sub[1]*v[2] - sub[2]*v[1];
	cross[1] = -(sub[0]*v[2] - sub[2]*v[0]);
	cross[2] =   sub[0]*v[1] - sub[1]*v[0];

	dot = u[0]*v[0] + u[1]*v[1] + u[2]*v[2];
```

```
	mult[0] = dot * cross[0];
	mult[1] = dot * cross[1];
	mult[2] = dot * cross[2];

	cout<<"("<<mult[0]<<", "<<mult[1]<<", "<<mult[2]<<")"<<endl;

	return 0;
}
```

Problem #5

```
#include <iostream>
#include <complex>

using namespace std;

int main(int argc, char* argv[])
{
	// This is actually pretty trivial with the stl
	double input_r, input_i;

	cout<<"A = ";
	cin>>input_r;

	cout<<"B = ";
	cin>>input_i;

	complex<double> x(input_r, input_i);

	cout<<"C = ";
	cin>>input_r;

	cout<<"D = ";
	cin>>input_i;

	complex<double> y(input_r, input_i);

	x = abs(x - y)*(x*x + y*y);
	cout<<x.real()<<"+"<<x.imag()<<"i"<<endl;

	return 0;
}
```

Problem #6

```
#include <iostream>
#include <cctype>

using namespace std;

int main(int argc, char* argv[])
{
```

```
	char input[256];

	cout<<"Input: ";
	cin.getline(input, 256);

	char token[256];
	int tokenElement, element = 0, tokenCount = 1;
	while(element < strlen(input))
	{
		if(input[element] == '\"')
		{
			element++;
			tokenElement = 0;
			while(input[element] != '\"') {
				token[tokenElement++] = input[element++]; }
			token[tokenElement] = '\0';
			cout<<"Token "<<(tokenCount++)<<":"<<token<<endl;
		}
		else if( isalnum(input[element]) )
		{
			tokenElement = 0;
			while( isalnum(input[element]) ) {
				token[tokenElement++] = input[element++]; }
			token[tokenElement] = '\0';
			cout<<"Token "<<(tokenCount++)<<":"<<token<<endl;
		}
		element++;
	}

	return 0;
}
```

Problem #7

```
#include <iostream>
#include <cmath>

using namespace std;

int main(int argc, char* argv[])
{
	int n;

	cout<<"n = ";
	cin>>n;

	double* a = new double[n];
	double* b = new double[n];

	for(int i = 0; i<n; i++)
	{
		cout<<"a"<<(i+1)<<" = ";
```

```
		cin>>a[i];
		cout<<"b"<<(i+1)<<" = ";
		cin>>b[i];
	}

	double xk;

	cout<<"xk = ";
	cin>>xk;

	double result = 0;

	for(int i = 0; i<n; i++) {
		result += a[i] * b[i] * pow(xk, b[i] - 1); }

	cout<<result<<endl;

	delete[] a;
	delete[] b;

	return 0;
}
```

Problem #8

```
#include <iostream>
#include <list>

using namespace std;

int main(int argc, char* argv[])
{
	int n;

	cout<<"Number of elements: ";
	cin>>n;

	double element;
	list<double> numbers;
	for(int i = 0; i<n; i++)
	{
		cin>>element;
		numbers.push_back(element);
	}

	double firstElement = numbers.front();

	numbers.sort();

	double mean, median = 0, prob, sum = 0, count = 0;

	list<double>::iterator numIter = numbers.begin();
```

```
	for(i = 0; numIter != numbers.end(); ++numIter, i++)
	{
		sum += *numIter;
		if(*numIter == firstElement) {
			count++; }

		// This is a little crude for finding the median
		// but it's the only way with a list
		if(n % 2 == 1 && i == n/2) {
			median = *numIter; }
		else if(i == n/2-1 || i == n/2){
			median += *numIter; }
	}

	if(n % 2 == 0) {
		median /= 2.; }

	prob = count/n;
	mean = sum/n;

	cout<<prob<<", "<<mean<<", "<<median<<endl;

	return 0;
}
```

Problem #9

```
#include <iostream>
#include <cmath>

#define PI 3.14159

using namespace std;

int main(int argc, char* argv[])
{
	double radius, height;

	cout<<"r = ";
	cin>>radius;

	cout<<"h = ";
	cin>>height;

	double surfaceArea = 2.*PI*radius*height + 2.*PI*radius*radius;

	double volume = PI*radius*radius*height;

	double prismSide = (2.*radius)/sqrt(2.);

	double diff = volume - prismSide*prismSide*height;
```

```
	cout<<surfaceArea<<", "<<diff<<endl;

	return 0;
}
```

Problem #10

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

#define NUM_CITIES 9

class Node
{
public:
	string getName() { return m_cityName; }

	bool findShortestPath(string dest, int curDist);

	void addPath(Node* city, int distance);

	Node(string name);

private:
	bool   m_beenHere;
	string m_cityName;
	vector<Node*> m_pathCity;
	vector<int>   m_pathDistance;
};

Node tampa("T");
Node atlanta("A");
Node dallas("Da");
Node newYork("NY");
Node chicago("C");
Node pheonix("P");
Node denver("De");
Node seattle("S");
Node sanFran("SF");

Node* allCities[NUM_CITIES] = { &tampa, &atlanta, &dallas, &newYork, &chicago,
                                &pheonix, &denver, &seattle, &sanFran };

int g_minDistance = INT_MAX;

int main(int argc, char* argv[])
{
	char input[3];
```

```
	cout<<"Origin: ";
	cin>>input;
	string origin(input);

	cout<<"Destination: ";
	cin>>input;
	string destination(input);

	tampa.addPath(&atlanta, 463);
	tampa.addPath(&dallas,  1181);
	atlanta.addPath(&newYork, 906);
	atlanta.addPath(&chicago, 780);
	atlanta.addPath(&seattle, 2713);
	atlanta.addPath(&dallas,  810);
	dallas.addPath(&pheonix, 1051);
	dallas.addPath(&chicago, 996);
	pheonix.addPath(&denver, 823);
	chicago.addPath(&seattle, 2058);
	seattle.addPath(&sanFran, 816);

	for(int i = 0; i<NUM_CITIES; i++)
	{
		if(allCities[i]->getName() == origin)
		{
			allCities[i]->findShortestPath(destination, 0);
			break;
		}
	}

	cout<<g_minDistance<<" mi"<<endl;

	return 0;
}

bool Node::findShortestPath(string dest, int curDist)
{
	if(m_beenHere == true) {
		return true; }

	if(m_cityName == dest)
	{
		if(curDist < g_minDistance) {
			g_minDistance = curDist; }

		return false;
	}

	m_beenHere = true;
	vector<Node*>::iterator cityIter = m_pathCity.begin();
	vector<int>::iterator   distIter = m_pathDistance.begin();

	for( ; cityIter != m_pathCity.end(); ++cityIter, ++distIter)
```

```
        {
                (*cityIter)->m_beenHere = (*cityIter)->findShortestPath(dest, curDist + *distIter);
        }
}

void Node::addPath(Node* city, int distance)
{
        m_pathCity.push_back(city);
        m_pathDistance.push_back(distance);

        city->m_pathCity.push_back(this);
        city->m_pathDistance.push_back(distance);
}

Node::Node(string name)
{
        m_beenHere = false;
        m_cityName = name;
}
```