Name______________ ✓ ______________

CS 31
Fall 2010
Midterm Exam
November 3, 2010

Problem #	Possible Points	Actual Points
1	10	10
2	10	10
3	10	10
4	10	10
5	10	10
6	10	10
7	15	15
8	10	10
9	15	10
TOTAL	100	~~[illegible]~~ 95 ✓

STUDENT ID #: [redacted]

SIGNATURE: [redacted]

ENROLLED IN LECTURE (circle one): 1 [✓ 2] 3

CLOSED BOOK, TWO 8.5"x11" SHEETS
NO ELECTRONIC DEVICES

ENJOY!

1. **[10 points in all]**

[5 points] Which of these statements is true about the following program?

a. It is a well-formed C++ program with well-defined behavior.

[✓ b.] It has no error that would prevent compilation, but when executed has undefined behavior or does something to cause it to end prematurely.

c. It has at least one error that will prevent compilation from succeeding.

Circle a, b, or c; if you circle b or c, briefly state the problem:

when k is s.size()-1, s.at(k+1) is out of range.

```
#include <iostream>
#include <string>
using namespace std;
bool isSorted(string s)
{
    for (int k = 0; k != s.size(); k++)
        if (s.at(k) > s.at(k+1))
            return false;
    return true;
}

int main()
{
    if (isSorted("effort"))
        cout << "The string is sorted" << endl;
    else
        cout << "The string is out of order" << endl;
}
```

[5 points] Which of these statements is true about the following program?

[✓ a.] It is a well-formed C++ program with well-defined behavior.

b. It has no error that would prevent compilation, but when executed has undefined behavior or does something to cause it to end prematurely.

c. It has at least one error that will prevent compilation from succeeding.

Circle a, b, or c; if you circle b or c, briefly state the problem:

```
#include <iostream>
using namespace std;
int main()
{
    int n;
    for (int k = 4; k >= 0; k--)
    {
        if (k < 2)
            cout << n << endl;
        else if (k != 3)
            cout << (1.0 / k) << endl;
        else
            n = 5;
    }
}
```

2. **[10 points] What is the output produced by the following program fragment?**

```
int n = 10;
while (n/2 > 2)
{
     cout << n << endl;
     n--;
     if (n % 2 == 0)     // a reminder: remainder
          n -= 2;
}
cout << "Finish: n = " << n << endl;
```

```
10
9
6
Finish: n= 5
```

3. **[10 points] Write a single statement that declares an array of 10 strings. Name the array `arr`. Then write a single statement that sets the value of the last element of `arr` to `"We're happy!"`. Then write a loop that sets every element of `arr` (other than the last element) to the value of the element after it, but with "^_^" appended. (After this loop has completed, the second-to-last element of `arr` will be `"We're happy!^_^"`, the third-to-last element will be `"We're happy!^_^^_^"`, etc.). You may assume that `#include <string>` and `using namespace std;` have appeared previously. Related reminder: If `s` and `t` are strings, then to store in `t` the result of appending `"ABC"` to `s`, you can say `t = s + "ABC";`**

```
string arr[10];
arr[9]= "we\'re happy!"
         nice
for (int k=8; k >=0, k--)
 arr[k]= arr[k+1] + "^_^";
```

✓

4. **[10 points in all] Recall from Project 3 that a *path segment* is one of the letters N, E, S, W, in upper or lower case, followed by one digit character or by two digit characters. A *path* is a sequence of zero or more path segments.**

The following program fragment is supposed to read in ten strings. For each string that is a well-formed path, the program is supposed print how many path segments in that path that have two digit characters following the letter. Assume the program contains a correct implementation of the `isPathWellFormed` function, which returns `true` if and only if its argument is a well-formed path.

```
#include <iostream>
#include <string>
#include <cctype>
using namespace std;
...
string p;
int longSegmentCount = 0;
for (int n = 0; n < 10; n++)
{
    cout << "Enter a path: ";
    getline(cin, p);
    if (isPathWellFormed(p))
    {
        for (int k = 2; k < p.size(); k += 2)
        {
            if (isdigit(p.at(k)))
            {
                longSegmentCount++;     1
                k++;
            }
        }
        cout << longSegmentCount << endl;
    }
    else
        cout << "That path is not well-formed" << endl;
}
```

0 1 2 . 3 4
DDD D
w 1 s . 10

k=4

k=5

k = ~~0~~7

Long seg. 1

k < 10

~~0~~5 k=7

2

A sample transcript *should* start out

```
Enter a path: w1s10
1
                0 1 2 3 4 5 6 7 8 9
Enter a path: n12s3e10w2
2
Enter a path: n12
1
Enter a path: ne12
That path is not well-formed
Enter a path:
0
Enter a path: s3w2
0
... etc. ...
```

4. (continued)

[5 points] The program does not work as intended. If the first two strings entered as input to the program are the two in the sample transcript on the previous page, what are the first lines the program actually writes (other than the `Enter a path` prompts)? (If the program worked as intended, you'd answer

```
1
2
```

but the program doesn't work as intended.)

1
3 ✓

[5 points] What small change to the program will fix the problem? (You may either clearly indicate the change below, or rewrite a portion of the code.)

```
string p;
~~int longSegmentCount = 0;~~
for (int n = 0; n < 10; n++)
{         int  longSegmentCount = 0          ✓
    cout << "Enter a path: ";
    getline(cin, p);
    if (isPathWellFormed(p))
    {
        for (int k = 2; k < p.size(); k += 2)
        {
            if (isdigit(p.at(k)))
            {
                longSegmentCount++;
                k++;
            }
        }
        cout << longSegmentCount << endl;
    }
    else
        cout << "That path is not well-formed" << endl;
}
```

5. [10 points] Convert this switch statement to code that produces exactly the same output without using a switch statement. (We'll give you a break by reminding you to read the switch statement carefully.)

```
char legInjuryCode;
 ... assume some intervening code here that gives legInjuryCode a value ...
switch (legInjuryCode)
{
  case 'C':
  case 'S':
     cout << "Muscle cramp or sprain" << endl;
     break;
  case 'B':
     cout << "Bone break" << endl;
  case 'L':
     cout << "Laceration" << endl;
     break;
  default:
     cout << "No problem" << endl;
     break;
}
```

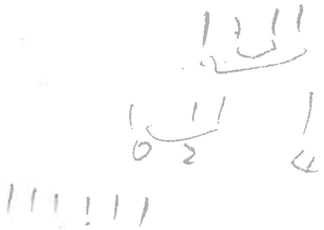
```
if ( legInjuryCode == 'C' || legInjuryCode == 'S' )
      cout << "Muscle cramp or sprain" << endl;
else if ( legInjuryCode == 'B' )
     cout << "Bone break" << endl << "Laceration" << endl;        ✓
else if ( legInjuryCode == 'L' )
     cout << "Laceration" << endl;

else
     cout << "No problem" << endl;
```

6. **[10 points] The following function is supposed to reverse the order of the characters in a string. For example, if `str` is a string with the value `"ten"`, then `flip(str)` changes its value to `"net"`, while if its value were `"regal"`, then `flip(str)` changes its value to `"lager"`, or if its value were `" @# $"`, then `flip(str)` changes its value to `"$ #@ "`. If its value were `"rotator"`, then `flip(str)` results in its still being `"rotator"`. The function works correctly for these and many other strings, but has a serious flaw — there are many strings for which it does *not* correctly reverse the string.**

e = 0.

```
void flip(string& s)
{
   if (s.size() == 0)
      return;
   int b = 0;
   int e = s.size() - 1;
   while (b != e)
   {
      char ch = s.at(b);
      s.at(b) = s.at(e);
      s.at(e) = ch;
      b++;
      e--;
   }
}
```



Briefly state the common characteristic of all and only those strings that this function doesn't correctly reverse. (Examples of not-necessarily-correct answers to this problem would be "All strings starting with M" or "All strings whose length is a prime number" or "All strings consisting of three of the same character, such as xxx or 666")

All strings whose length is even number.

What small change to the program will fix the problem? (You may either clearly indicate the change below, or rewrite a portion of the code.)

```
void flip(string& s)
{
   if (s.size() == 0)
      return;
   int b = 0;
   int e = s.size() - 1;
   while (b != e)
   {
      char ch = s.at(b);
      s.at(b) = s.at(e);
      s.at(e) = ch;
      b++;
      e--;
   }
}
```

while (b!=e && b!=e-1)

✓

7. [15 points] A *self-indexed array* of size n is an integer array each of whose elements is an integer between 0 and n-1, inclusive. Here's an example:

```
int arr[10] = { 5, 4, 7, 3, 1, 9, 0, 5, 1, 2 };
```

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]	arr[7]	arr[8]	arr[9]
5	4	7	3	1	9	0	5	1	2

We can consider each value in a self-indexed array to specify the index of some (usually different) element in the array. Define the *successor* of any element k to be the element whose index is arr[k]. For example, the successor of element 0 is element 5, since the value of arr[0] is 5. The successor of element 5 is element 9.

If we start at some element, we can form a *path* by following the chain of successors starting from that element. For example, if we start at element 2, we see that its successor is element 7. The successor of element 7 is element 5. The successor of element 5 is element 9. The successor of element 9 is element 2. If a path eventually cycles like this back to the element we started with, we call it a *circuit*.

On the other hand, if we start at element 8, we find its successor is element 1, whose successor is element 4, whose successor is element 1. This path cycles between elements 1 and 4, but it is not a circuit, because the cycle does not include the starting element of 8.

On the next page, write a function named `hasCircuit` that takes three parameters:

An array of integers named `a`, a self-indexed array;
An integer named `n`, the number of elements in the array;
An integer named `start`, the starting element to check for a circuit

The function returns the boolean value `true` if there is a circuit in the array starting at element `start`, or `false` otherwise. You may assume (and thus don't have to check) that `a` is indeed a self-indexed array of size `n`, that `n` is positive, and that start is between 0 and `n-1`, inclusive. When your function returns, `a` must contain the same values it did when the function was entered.

Here's an example of how the function can be used:

```
int arr[10] = { 5, 4, 7, 3, 1, 9, 0, 5, 1, 2 };

if (hasCircuit(arr, 10, 1)) // the cycle is 1->4->1->...
     cout << "Circuit!";    //  so this will print Circuit!

if (hasCircuit(arr, 10, 6)) // 6->0->5->9->2->7->5->...
     cout << "Circuit!";    //  so this will not print
                            //  Circuit!, since the cycle
                            //  does not include 6
```

7. (continued) Write your `hasCircuit` function here. (You do not have to write a main routine or #include directives.) Hint: No cycle in a self-indexed array can be longer than the number of items in the array.

```
bool  hasCirchit ( int a[], int n, int start)
{        int k= start;

         for( int i=0; i<n; i++)
         { k = a[k];
           if (k== start)
             return true; }

      return false;

   }.
```

8. [10 points] What is the ninth digit of your UCLA student ID number? 3___
What output is produced by the following program if its input is the ninth digit of your UCLA student ID number?

```
3   40
3   40
3
4
go
```

```
#include <iostream>
#include <string>
using namespace std;

void doe(int go, int beau);
void whoa(int& blow);
void though(string sew);

int main()
{
   int go;
   cin >> go;        // you enter the ninth digit of
                     // your student id
   int blow = 40;
   doe(go, blow);
   doe(go, blow);
   whoa(go);
   whoa(go);
   though("go");
}

void doe(int go, int beau)
{
   cout << go << " " << beau << endl;
   go++;
   beau = 80;
}

void whoa(int& blow)
{
   cout << blow << endl;
   blow++;
}

void though(string sew)
{
   cout << sew << endl;
}
```

9. [15 points] Let's define a *run* as a sequence of consecutive identical values. For example, the sequence 5 8 8 2 7 7 7 7 8 8 8 3 3 3 3 has a run of length 1 with the value 5, a run of length 2 with the value 8, a run of length 1 with the value 2, a run of length 4 wih the value 7, a run of length 3 with the value 8, and a run of length 4 with the value 3. The longest run is of length 4 (there are two of them); the earliest appearing one of those two longest runs has the value 7.

Write a function named `longestRun` that takes three parameters:

an array of integers named `a`
an integer named `n`, the number of interesting elements in `a`
a reference to integer named `value`

The function returns an int, the length of the longest run in the first `n` elements of array `a`. The function sets `value` to the value in that longest run; if there is more than one run of the longest length, the function sets value to the value in the earliest appearing longest run. The function does not change any of the values in the array. Here's an example of how the function can be used:

```
int main()
{
     int data[15] = {
          5, 8, 8, 2, 7,  7, 7, 7, 8, 8,  8, 3, 3, 3, 3
     };
     int v;
     int len = longestRun(data, 15, v);
     cout << len << ' ' << v << endl;  // writes  4 7

     len = longestRun(data, 5, v);
     cout << len << ' ' << v << endl;  // writes  2 8

     len = longestRun(data, 2, v);
     cout << len << ' ' << v << endl;  // writes  1 5
}
```

On the next page, write the `longestRun` function. You may assume that `n` will always be positive and that the array passed in to `a` will have at least `n` elements. (In other words, you don't have to do any error checking.)

10

9. (continued) Write your `longestRun` function here.

```
int   longestRun (int a[], int n, int& value)
{    int max =1;     value = a[0]
     for (int k=0; k<n-1; k++)
     {   int run =1;
     if (a[k]==a[k+1])
          run ++;

          if (run > max)
     { max = run;
          value = a[k]; }
     }

     return max;

}
```

keep resetting run to 1

Can't go passed #2

5