Computer Science 7A – Spring '13 – Midterm 2 Name_____

Write all responses on separate paper. Do not use an electronic computer or calculator.

1. Suppose that `word` is the array declared as

```
char word[8] = {0};
```

   a. Compute the offset for `word[5]`, in bytes.
   b. If `word` is the address `0x3fffcb6`, what is the memory address of `word[7]`?
   c. What would be different if word were declared as shown below? Explain.

```
char* word = "";
```

   d. What would be the output of the following code fragment if the preprocessor directive

```
#define SIZE 65
```

   was included in the header?

```
char word[8] = {SIZE,SIZE+1,SIZE+2};
cout << "\n word = " << word << endl;
```

2. Suppose that `data7` is an array declared as

```
double data7[5][4] = {1,1,2,3,5,8,13,21,34};
```

   a. Compute the offset for the reference `data7[3][2]`. Assume a double requires 8 bytes.
   b. If `data7` is the address `0x3fff000`, what is the memory address of `a[3][2]`?
   c. What is the output of the code fragment below?

```
int main()
{
  double data7[5][4] = {1,1,2,3,5,8,13,21,34};
  for(int i = 0; i < 3; ++i)
  {
       for(int j = 0; j < 4; ++j)
          cout << "data7["<<i<<"]["<<j<<"]="<<data7[i][j]<<"
";
       cout << endl;
  }
}
```

3. Consider the string stored in contiguous memory, as shown to the right.
   a. Determine the integer representation of these 4 one-byte integers:
   b. Determine the word represented by these data. Note that the ASCII value of A is 65.
   c. Interpret the 4 bytes as 2 hexadecimal (base-16) numbers.

⋮
01010111
01000001
01010010
01000100
00000000

4. Write a function to take a cstring, reverse it and see if it's a palindrome (that is, a word that reads the same forwards and backwards like "TOOT").

5. Write a program that will read through all the words in "FourLetterWords.txt" and, if the word is a palindrome, print it out.

6. What will be the output of the following complete program?

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

using namespace std;

int main ()
{
   string str1 = "marma";
   string str2 = "lade";
   string str3;
   int  len;

   // copy str1 into str3
   str3 = str1;
   cout << "str3 : " << str3 << endl;

   // concatenates str1 and str2
   str3 = str1 + str2;
   cout << "str1 + str2 : " << str3 << endl;

   // total length of str3 after concatenation
   len = str3.size();
   cout << "str3.size() :  " << len << endl;

   return 0;
}
```

7. Write a program `wordMerge()` that will take two alphabetically correct words like "`adders`" and "`clop`" (that is, words whose letters are in alphabetical order) of any lengths and merge them so they form a new alphabetically correct word. So that, if the input to `wordMerge()` are the words "adders" and "clop" then the output is "`acddeloprs`."

Computer Science 7A – Spring '13 – Midterm 2 Solutions

1. Suppose that `word` is the array declared as

```
char word[8] = {0};
```

a. Compute the offset for `word[5]`, in bytes.

SOLN: Each char is one byte, so the offset would be 5.

b. If `word` is the address `0x3fffcb6`, what is the memory address of `word[7]`?

SOLN: 6+7=13, or, 'd' in hex, so the addresss is 0x3fffcbd

c. What would be different if word were declared as shown below? Explain.

```
char* word = "";
```

SOLN: Strangely, they are exactly the same.

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

int main()
{
      char word1[8] = {0};
      char* word2 = "";
      for(int i = 0; i < 10; ++i)
      {
            cout << endl << "word1[" << i << "] = "
                 << word1[7];
            cout << endl << "word2[" << i << "] = "
                 << word2[7] << endl;
      }
      cout << endl << "word1 = " << word1;
      cout << endl << "word2 = " << word2 << endl;

}
```

```
word1[0] =
word2[0] =

word1[1] =
word2[1] =

word1[2] =
word2[2] =

word1[3] =
word2[3] =

word1[4] =
word2[4] =

word1[5] =
word2[5] =

word1[6] =
word2[6] =

word1[7] =
word2[7] =

word1[8] =
word2[8] =

word1[9] =
word2[9] =

word1 =
word2 =
```

The program above produces the output shown to the right. There are zero warning, zero errors. Now, had my computer contained some garbage in that area, it might have looked more cluttered, but the declarations are exactly equivalent: the null terminator is the first character in the array of characters, so that's where the string ends.

d. What would be the output of the following code fragment if the preprocessor directive

```
#define SIZE 65
```

was included in the header?

```
char word[8] = {SIZE,SIZE+1,SIZE+2};
cout << "\n word = " << word << endl;
```

SOLN: word = ABC

Admittedly, "SIZE" is a bit misleading here. What's important is that the ASCII value for A is 65, for B it's 66 and C is 67.

2. Suppose that `data7` is an array declared as

```
double data7[5][4] = {1,1,2,3,5,8,13,21,34};
```

a. Compute the offset for the reference `data7[3][2].` Assume a double requires 8 bytes.
SOLN: This declaration allots memory for 20 doubles and if each double requires 8 bytes then the memory allocated is 160 bytes. While the actual memory is contiguous, the array indices allow it to be addressed in blocks. You can conceive of the memory as being in a rectangular array of 5 rows of 4, with 4*8 = 32 bytes each in each row. Thus data7[3][2] is the 3rd element in the 4th row, at an offset of 3*32 + 2*8 = 112 bytes.

b. If `data7` is the address `0x3fff000`, what is the memory address of `a[3][2]`?

SOLN: In hex, 112 = 7A. The address would be `0x3fff07a.`

c. What is the output of the code fragment below?

```
int main()
{
  double data7[5][4] = {1,1,2,3,5,8,13,21,34};
  for(int i = 0; i < 3; ++i)
  {
       for(int j = 0; j < 4; ++j)
          cout << "data7["<<i<<"]["<<j<<"]="<<data7[i][j]<<"
";
       cout << endl;
  }
}
```

SOLN:

```
data7[0][0]=1 data7[0][1]=1 data7[0][2]=2 data7[0][3]=3
data7[1][0]=5 data7[1][1]=8 data7[1][2]=13 data7[1][3]=21
data7[2][0]=34 data7[2][1]=0 data7[2][2]=0 data7[2][3]=0
Press any key to continue . . .
```

3. Consider the string stored in contiguous memory, as shown to the right.
   a. Determine the integer representation of these 4 one-byte integers:
   SOLN: 01010111_2=64+16+4+2+1=87, 01000001_2=64+1=65, 01010010_2=64+16+2=82, 01000100_2=64+4=68 and 00000000_2=0.
   b. Determine the word represented by these data. Note that the ASCII value of A is 65.
   SOLN: Assuming the null character is at the end, the word would be 87,65,82,68,0 = WARD, followed by '\0'
   c. Interpret the 4 bytes as 2 hexadecimal (base-16) numbers.
   SOLN: The 87 is shifted to $87*2^8$ and then added to the +65 yielding $87*2^8$+65=22337=010101101000001_2. But the way to convert to hex is go nibble by nibble 0101 = 5, 0111 = 7, 0100 = 4 and 0001 = 1 so $010101101000001_2 = 5741_{16}$ and 01010010010001002 = 5244_{16}.

4. Write a function to take a cstring, reverse it and see if it's a palindrome (that is, a word that reads the same forwards and backwards like "TOOT").

SOLN:

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

bool palindrome(char*);

int main()
{
   char word[] = "";
   cout << "\nEnter a word to test if it's a palindrome: " << endl;
   while(cin >> word)
      if(palindrome(word))
            cout << endl << word << " is a palindrome." << endl;
      else cout << endl << word << " is not a palindrome." << endl;
}

bool palindrome(char* w)
{
      int size = strlen(w);
      for(int i = 0; i <= size/2; i++)
            if(w[i]!=w[size-i-1]) return false;
      return true;
}
```

Here's some typical output:

```
Enter a word to test if it's a palindrome:
deified

deified is a palindrome.
dodobird

dodobird is not a palindrome.
tat

tat is a palindrome.
cat

cat is not a palindrome.
madam I'm adam

madam is a palindrome.

I'm is not a palindrome.

adam is not a palindrome.
```

5. Write a program that will read through all the words in “FourLetterWords.txt” and, if the word is a palindrome, print it out.
SOLN:
This program can use the function we wrote four #4.

```
//midterm 2 solutions

#include <iostream>
#include <fstream>
using namespace std;

bool palindrome(char*);

int main()
{
	char word[] = "AAAA";
	ifstream read("FourLetterWords.txt");
	while(read>>word)
		if(palindrome(word)) cout << word << " ";
}

bool palindrome(char* w)
{
	int size = strlen(w);
	for(int i = 0; i <= size/2; i++)
		if(w[i]!=w[size-i-1]) return false;
	return true;
}
```

Here is the output:
ABBA ACCA ANNA BOOB DEED ECCE ESSE GOOG KEEK KOOK NAAN NOON OPPO OTTO PEEP POOP SEES TOOT
Press any key to continue . . .

6. What will be the output of the following complete program?

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

using namespace std;

int main ()
{
   string str1 = "marma";
   string str2 = "lade";
   string str3;
   int  len;

   // copy str1 into str3
   str3 = str1;
   cout << "str3 : " << str3 << endl;

   // concatenates str1 and str2
   str3 = str1 + str2;
```

```
   cout << "str1 + str2 : " << str3 << endl;

   // total length of str3 after concatenation
   len = str3.size();
   cout << "str3.size() :  " << len << endl;

   return 0;
}
```

SOLN:

```
str3 : marma
str1 + str2 : marmalade
str3.size() :  9
Press any key to continue . . .
```

7. Write a program `wordMerge()` that will take two alphabetically correct words like "`adders`" and "`clop`" (that is, words whose letters are in alphabetical order) of any lengths and merge them so they form a new alphabetically correct word. So that, if the input to `wordMerge()` are the words "adders" and "clop" then the output is "`acddeloprs`."

SOLN:

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

bool alpha(string);
string wordMerge(string, string);

int main()
{
      string word1, word2, word3;
      int ctr1 = 0;
      cout << "\nEnter two alphabetically correct words: " << endl;
      do
      {
            cin >> word1 >> word2;
      } while(!(alpha(word1) && alpha(word2)));
      cout << "\nThe merged word is " << wordMerge(word1, word2)
           << endl;
}

bool alpha(string w)
{
      int len = strlen(w.c_str());
      for(int i = 0; i < len-1; ++i)
            if(!(w[i] <= w[i+1])) return false;
      return true;
}

string wordMerge(string w1, string w2)
{
```

```
      string w3 = "";
      int ctr1 = 0, ctr2 = 0,
          len1 = strlen(w1.c_str()), len2 = strlen(w2.c_str());
      while(ctr1 < len1 && ctr2 < len2)
            if(w1[ctr1]<=w2[ctr2])
                  w3.append(1,w1[ctr1++]);
            else
                  w3.append(1,w2[ctr2++]);
      if(ctr1<len1)
            for(int i = ctr1; i < len1; i++)
                  w3.append(1,w1[i]);
      if(ctr2<len2)
            for(int i = ctr2; i < len2; i++)
                  w3.append(1,w2[i]);
      return w3;
}
```

Here is the output:

```
Enter two alphabetically correct words:
access boot

The merged word is abcceoosst
```