Eric Roberts
CS 106B

Handout #23
February 2, 2015

Section Handout #4
Pointers and Memory

Problem 1. Differentiating the stack and the heap

Using the heap-stack diagrams from Chapter 12 as a model, draw a diagram showing how memory is allocated just before the following program returns:

```
int main() {
   int *a1 = new int[2];
   a1[0] = 1776;
   a1[1] = 2001;
   double a2[3] = { 1.61803, 2.71828, 3.14159 };
   double *dp = a2;
   return 0;          ←Diagram memory at this point
}
```

In this example, you should make two diagrams: one that uses explicit addresses and one that uses arrows to represent pointers. In this problem, you should assume that the type **`int`** requires four bytes, the type **`double`** requires eight bytes, all heap addresses begin with the hexadecimal digit **`1`**, and all stack addresses begin with the hexadecimal digit **`F`**.

1a) Diagram using explicit addresses:

heap | *stack*

1b) Diagram using arrows:

heap | *stack*

Problem 2: Pointer arithmetic

In the memory diagrams you drew for problem 1, what would be the value of each of the following expressions:

2a) `a1[1]`

2b) `a2[2]`

2c) `*a1`

2d) `*dp`

2e) `*(dp + 1)`

2f) `&a2[3] - dp`

Problem 3: Pointer tracing

What output is generated by the following program:

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

char *enigma(char *cp);
void mystery(char *cp);

int main() {
   char s1[4] = { 'H', 'H', 'H', '\0' };
   char *s2 = enigma(s1);
   *s2-- = '!';
   cout << s1 << endl;
   mystery(s2);
   *--s2 -= 5;
   cout << s1 << endl;
   return 0;
}

char *enigma(char *cp) {
   char ch1 = *cp++;
   char ch2 = (*cp)++;
   if (ch1 == ch2) cp++;
   return cp;
}

void mystery(char str[]) {
   for (int i = 0; str[i] != '\0'; i++) {
      str[i] = '+';
   }
}
```

4. Heap-stack diagrams

Using the heap-stack diagrams from Chapter 12 as a model, draw a diagram showing how memory is allocated for each of the following problems. You need not include explicit addresses in your diagram, but must indicate—either through addresses or arrows—where reference values point in memory. For extra practice, you might try drawing the heap-stack diagrams in both the explicit address and arrow forms.

4a.

```
enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES };

struct Card {
   int rank;
   Suit suit;
};

Card createCard(int rank, Suit suit);

int main() {
   Card *hand = new Card[2];
   hand[0] = createCard(9, DIAMONDS);
   hand[1] = createCard(7, HEARTS);
   return 0;
}

Card createCard(int rank, Suit suit) {
   Card card;
   card.rank = rank;
   card.suit = suit;
   return card;
}
```

←Diagram at this point on the second call to this function

heap | *stack*

4b.

```
class Matrix {
public:

   Matrix(int size) {
      this->size = size;
      data = new double *[size];
      for (int i = 0; i < size; i++) {
         double *row = new double[size];
         data[i] = row;
         for (int j = 0; j < size; j++) {
            row[j] = (i == j) ? 1.0 : 0.0;
         }
      }
   }

   int getSize() {
      return size;
   }

   int get(int row, int col) {
      return data[row][col];
   }

private:
   int size;
   double **data;

};

int main() {
   Matrix m(2);
   for (int i = 0; i < m.getSize(); i++) {
      for (int j = 0; j < m.getSize(); j++) {
         cout << " " << m.get(i, j);
      }
      cout << endl;
   }
   return 0;
}
```

←Diagram memory on the last time this point is reached.

heap

stack