Announcements
Office Hours this week
   Friday:  2:30 — 3:15 pm
   No Office Hours Monday, Tuesday next week

Today
   Swapping data
   Memory allocation
Swapping data
(covered in reading, but you may have missed it)
Problem: Given 2 integers, stored in variables x and y,
write code to swap the values stored in these variables
Example: if x is 3 and y is 7, write code so that at the end,
x is 7 and y is 3

Approach 1
x = y;
y = x;

Approach 2
int temp = x;
x = y;
y = temp;

Which of these approaches work properly?
A. Both 1 and 2
B. 1, not 2
C. 2, not 1
D. Neither 1 nor 2
Consider a 1-dimensional array as a mechanism to store many data elements. What positive properties do arrays have?

1. The size of arrays is flexible; one can add space to arrays, as needed, after an array variable is declared.
2. Given an index i and array a, array element a[i] can be accessed very efficiently.
3. Passing an array to a procedure is efficient: Only an address need be provided; the elements of the array are not copied.

A. 1, 2, and 3 are all true
B. 2 and 3 are true; 1 is not
C. 1 and 3 are true; 2 is not
D. 1 is true; 2 and 3 are not
E. some other combination is true/false
Suppose I want to read an integer n from the terminal, declare an array of size n, and then read n characters into that array.

Two approaches are proposed:

1. int i, n;
   scanf ("%d", &n);
   char a [n];
   for (i = 0; i < n; i++)
      scanf ("%c", &a[i]);

2. int i, n;
   scanf ("%d", &n);
   char *a = malloc (n*sizeof (char));
   for (i = 0; i < n; i++)
      scanf ("%c", &a[i]);

Which, if any, of these approaches works?

A. Both  B. 1, not 2  C. 2, not 1  D. Neither
Suppose I want to do the same reading of n char values, but I want to do that in a function readString, called in main.

Three possible approaches for the code in main are proposed:

1. char str [];
   str=readString();
2. char str [1000];
   str=readString();
3. char str [] =readString();
4. char * str;
   str=readString();

A. Both 1, 2   B. 1, not 2   C. 2, not 1   D. Neither
A. Both 3, 4   B. 3, not 4   C. 4, not 3   D. Neither
Suppose I want to write a function that will read an integer \( n \) from the terminal, declare a char array of size \( n \), and then read a string of \( n \) characters (adding the null at the end). The function should return the string (actually a pointer to the string), so a call to the function would be

\[
\text{char } * \text{ str } = \text{readString} ()
\]

Two approaches are proposed:

1. \[
\text{char } * \text{ readString} () \{
    \text{int } i, n; \\
    \text{scanf } ("%d", &n); \\
    \text{char } a [n+1]; \\
    \text{for } (i = 0; i < n; i++) \text{scanf } ("%c", &a[i]); \\
    a[n] = 0; \text{return } a;
\}
\]

2. \[
\text{char } * \text{ readString} () \{
    \text{int } i, n; \\
    \text{scanf } ("%d", &n); \\
    \text{char } a [n+1] = \text{malloc}((n+1)\text{sizeof(char)}); \\
    \text{for } (i = 0; i < n; i++) \text{scanf } ("%c", &a[i]); \\
    a[n] = 0; \text{return } a;
\}
\]

Which, if any, of these approaches works?

A. Both  B. 1, not 2  C. 2, not 1  D. Neither
A program is to read a line of input, as long as the user wishes. After reading the line, the program is to print it backwards. Two approaches are proposed:

```
#define maxLen 10000
#include <stdio.h>
#include <string.h>
int main()
{
    char line[maxLen+1];
    fgets(line, maxLen, stdin);
    for(int i=strlen(line)-1;i >=0; i--)
        printf("%c", line[i]);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
    int curLen = 0;
    int maxLen = 10;
    char * line = malloc(maxLen*sizeof(char));
    char ch;
    int i;
    while ((ch = getchar()) != 'n')
    {
        if (curLen == maxLen-1)
        {
            char * temp = malloc((10+maxLen)*sizeof(char));
            for (i = 0; i < maxLen-1; i++)
                temp[i] = line[i];
            free(line);
            line = temp;
            maxLen += 10;
        }
        line[curLen] = ch;
        curLen++;
    }
    line[curLen] = 0;  // add null at end of string
    for(int i=strlen(line)-1;i >=0; i--)
        printf("%c", line[i]);
}
```

Which approach(es) work?
A. Both Left and Right
B. Left only
C. Right only
D. Neither