(1) Overview:

- Overview
  - array doubling
    - In main()
    - In a function

(2) Array Doubling - in main()

The following shows how to perform a single instance of array doubling.

```
int main(){
    // say, initial array a has a length of 3
    int n = 3;

    int *a; = new int[n];

   // 1) allocate memory for a dynamic array that is 2n long
    int *temp = new int[2*n];

   // 2) copy the contents of array pointed to by a into first
   // n elements of array pointed to by temp
   for( int i = 0; i < n ; i++ )
      temp[i] = a[i];

   // return memory to heap

   delete [] a;
    a = temp;

    // assign a to point to temp
   n = 2*n;  // update the size


    ...
   return 0;

}
```

Say, we need to write a function to perform the array doubling. How do we pass the array?

void arrDouble(<array>)

(3) Array Doubling: function

Recall from last time: array doubling.

Now, what if we wanted to do the same thing but in a separate function?

```
void arrDouble(int a[], int n){ // is this right?
       //
    1) allocate memory for a dynamic array that is 2n long
       int *temp = new int[2*n];

       //
    2) copy the contents of array pointed to by a into first
       // n elements of array pointed to by temp
       for( int i = 0; i < n ; i++ )
          temp[i] = a[i];

       // return memory to heap
       delete [] a;

       a = temp; // assign a to point to temp
       n = 2*n;  // update the size
}

int main(){
    int n = 3;
    int *main_array;
    main_array = new int[n];
    arrDouble(main_array, n);
    /* more code here */

    delete [] main_array;
    return 0;

}
```

(4) Creating a dynamic array in a function

Sometimes you will need to generate a new array in a function and return the array. Here are two correct ways to do this with dynamically allocated arrays:

1. Return a pointer (covered in recitation)
2. Update a pointer (you'll do this on Assignment-3)

```
#include <iostream>
using namespace std;
int updateArray(int *& newArray){
    // example return array of random length
    // between 0 and 9
    int length = rand()%10;
    int *temp = new int[length];
    newArray = temp;
    return length;
    //note: could combine the lines:
    // newArray = new int[length];
    // showing two lines to stress what is happening
}

int main(){
    int * arrayPtr, arrayLength;
    arrayLength = updateArray(arrayPtr);


    ...

    delete [] arrayPtr;
    return 0;
}
```

(5) Check your understanding

```
void foo(int a_foo[]){
    cout << "C > " << a_foo << endl;
    cout << "D > " << &a_foo << endl;
}
int main(){
    int n = 3;
    int *a_main = new int[n];

    cout << "A > " << a_main << endl;
    cout << "B > " << &a_main << endl;
    foo(a_main);
    return 0;
}
```

If we run the following code, and the first print statement prints the following:

A > 0x7fffdbff6eb0

What can we expect to see from the second print statement (B >)?

a) Same address
b) Different address

(6) Check your understanding

```
void foo(int a_foo[]){
    cout << "C > " << a_foo << endl;
    cout << "D > " << &a_foo << endl;
}
int main(){
    int n = 3;
    int *a_main = new int[n];

    cout << "A > " << a_main << endl;
    cout << "B > " << &a_main << endl;
    foo(a_main);
    return 0;
}
```

If we run the following code, and the initial statments print the following:

```
A > 0x7fffdbff6eb0
B > 0x7fffe3bb9890
```

What can we expect to see from the third print statement (C >)?

a) Same address as A
b) Same address as B
c) Address different than A and B

(7) Check your understanding

```
void foo(int a_foo[]){
    cout << "C > " << a_foo << endl;
    cout << "D > " << &a_foo << endl;
}
int main(){
    int n = 3;
    int *a_main = new int[n];

    cout << "A > " << a_main << endl;
    cout << "B > " << &a_main << endl;
    foo(a_main);
    return 0;
}
```

If we run the following code, and the initial statments print the following:

```
A > 0x7fffdbff6eb0
B > 0x7fffe3bb9890
C > 0x7fffdbff6eb0
```

What can we expect to see from the fourth print statement (D >)?

a) Same address as A and C
b) Same address as B
c) Unique address

(8) Check your understanding

```
void foo(int *& a_foo){
    cout << "C > " << a_foo << endl;
    cout << "D > " << &a_foo << endl;
}
int main(){
    int n = 3;
    int *a_main = new int[n];

    cout << "A > " << a_main << endl;
    cout << "B > " << &a_main << endl;
    foo(a_main);
    return 0;
}
```

If we run the following code, and the initial statments print the following:

```
A > 0x7fffc601deb0
B > 0x7fffce8aa050
C > 0x7fffc601deb0
```

What can we expect to see from the fourth print statement (D >)?

a) Same address as A and C
b) Same address as B
c) Unique address

jump back to [array doubling example]