Agenda / Learning Objectives:

1. Map out a plan to study for mid-term 2.
2. Review the C++ operators up to logical operators.
3. Read about the tips and pitfalls on using arrays (see below.)
4. Understand 1D array and get familiar with different types of questions about array in exams.

Tips (from chapter 5 of Absolute C++ textbook)

Initializing Arrays. In initializing an array,

```
int children[3] = {1, 12, 1};
```

the number of initializers must be no greater than the declared size of the array. There can be fewer initializers. In such an event the first array entries will be initialized from the list, the C++ Standard says the arrays are to be initialized to a zero appropriate to the type. Note that this is true even for primitive types and for class types, where the default constructor is used for the "zero of appropriate type." However, if there are no initializers, and the base type is primitive, the array is not initialized at all. If the base array type is a class type, if there are no initializers, the default constructor is called for each array item.

Do Not Skimp on Formal Parameters. There is sometimes a tendency to use global variables instead of passing formal parameters to functions (e.g., the size of an array). However, the use of formal parameters makes the code more readable and modular which makes it easier to maintain and less likely to contain errors.

Memory Layout for Multidimensional Arrays. The layout in memory of a multidimensional array helps to explain why it is necessary to specify the all indices except the first. If you declare an array with more than one index, as you move through the memory that the array occupies, the last index changes most rapidly:

```
char x[3][4];
```

```
Mem Addr:   1000     1001     1002     1003     1004     1005
            x[0][0]  x[0][1]  x[0][2]  x[0][3]  x[1][0]  x[1][1]

            1006     1007     1008     1009     1010     1011
            x[1][2]  x[1][3]  x[2][0]  x[2][1]  x[2][2]  x[2][3]
```

If this were a three index array:

```
char x[3][2][4]
```

Then if the memory address of x[0][0][0] is 1000, the rest of the memory addresses are:

```
Mem Adr:     1000        1001        1002        1003
array elemt  x[0][0][0]  x[0][0][1]  x[0][0][2]  x[0][0][3]

             1004        1005        1006        1007
             x[0][1][0]  x[0][1][1]  x[0][1][2]  x[0][1][3]

             1008        1009        1010        1011
             x[1][0][0]  x[1][0][1]  x[1][][2]   x[1][0][3]

             1012        1013        1014        1015
             x[1][1][0]  x[1][1][1]  x[1][1][2]  x[1][1][3]

             1016        1017        1018        1019
             x[2][0][0]  x[2][0][1]  x[2][0][2]  x[2][0][3]

             1020        1021        1022        1023
             x[2][1][0]  x[2][1][1]  x[2][1][2]  x[2][1][3]
```

The requirement that function parameters specify all array sizes except the first is a direct consequence of the memory layout. The extent in memory of the array is determined by the maximum size of the first index. The organization in memory is determined by the sizes after the first. If this array had first index size 2 instead of 3, we would have occupied 16 bytes in memory instead of 24 as above:

```
Mem Adress: 1000        1001        1002        1003
array elemt x[0][0][0]  x[0][0][1]  x[0][0][2]  x[0][0][3]

            1004         1005        1006        1007
            x[0][1][0]  x[0][1][1]  x[0][1][2]  x[0][1][3]

            1008        1009        1010       1011
            x[1][0][0]  x[1][0][1]  x[1][][2]  x[1][0][3]

            1012        1013        1014        1015
            x[1][1][0]  x[1][1][1]  x[1][1][2]  x[1][1][3]
```

Pitfalls (from chapter 5 of Absolute C++ textbook)

Array Indices Always Start with Zero. This is a stumbling block for many beginning students that want to number the first spot in an array as 1. However, the first space in an array is actually at index zero.

Attempting to Declare an Array with a Non-Constant Size. This is a pitfall, albeit one many compilers will catch. If you wanted to have an array whose size you could set at runtime, you might write the following code. Unfortunately, this is illegal, though a few compilers (g++) allow it.

```
//To test behavior when arrays are declared, bounds are
// violated
#include <iostream>
using namespace std;
int main()
{
  int x[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  int n;
  cin >> n;
  int y[n];  // Here we have (illegally) declared
             // an array of non-constant size
  for (int i = 0; i < n; i++)
    cout << (y[i] = x[i]) << " " ;
  cout<<  endl;
  return 0;
}
```

The bottom line is that the array size must be known at compile time. The alternative is to dynamically allocate space on the heap. That is treated later.

Array Index Out of Range. The most common programming error made with arrays is an attempt to reference a nonexistent array index. Array indexes most commonly get out of range at the first or last iteration of a loop.

Using commas between array indices with C/C++ arrays. Unlike some other languages, C++ does not use a comma to separate array indices. If we have an array declared as

```
char page[30][100];
```

Then an incorrect attempt to access page[1][5] might be written as

```
page[1, 5]
```

However, a syntax error will not result. Instead, "1,5" is evaluated using the comma operator. In a comma expression, the expression to the left of the comma is evaluated. If it produces a value (e.g. is not void) that value is discarded. The expression to the right of the comma is evaluated. The type and value of the comma expression is the type and value of the right hand expression. Thus, the comma expression evaluates the index expression "1, 5" by "evaluating" the first integer and ignoring it, then using the value of the second integer for the value of the expression. The result is

```
page[5]
```

which is a one-dimensional array. This can result in grief for people with experience in languages that use a comma to separate indices in an array access, and great grief for are neophytes in computing.

Use of = and == with Arrays. Arrays are constant pointers and may not be assigned to different memory addresses. This is not the case with dynamic arrays, discussed in Chapter 10. Also, == does not test to see if two arrays have the same contents, but rather if the two arrays are stored at the same place in memory. If you need to test for equality of arrays, then you need to write code that explicitly looks at each element in both arrays to see that they are the same.

Array Questions:

1) Provide the output of the given line in the blanks below.

```
#include <iostream>
using namespace std;
int main() {
   int a[10] = {1, 6, 4, 2, 8, 4, 1, 3, 2, -2};
   cout << a[0] << endl; //(1)_____________________
   cout << a[3] << endl; //(2)_____________________
   cout << a[5] << endl; //(3)_____________________
   cout << a[9] << endl; //(4)_____________________

   for (int i = 0; i < 10; ++i)
     a[i] += i;
   cout << a[1] << endl; //(5)_____________________
   cout << a[2] << endl; //(6)_____________________
   cout << a[6] << endl; //(7)_____________________
   cout << a[8] << endl; //(8)_____________________
   return 0;
}
```

Writing a program:

2) Write a complete C++ program which carries out the following tasks:
   - Declares an integer array of size 10.
   - Accepts 10 inputs from a user (to be stored in the array).
   - Squares each element of the array.
   - Prints the array elements.

3) Write a complete C++ program which carries out the following tasks:
   a. Declares an integer array of size 10.
   b. Fills the array with random numbers between 1 and 100.
   c. Prints the array elements.
   d. Gets and prints the maximum number in the array.

Title Lines:

4) (prac2.pdf) Write **title lines** for the functions that are called by the following main program. **Do not supply the blocks for the functions.**

```
int main() {
  int a[5] = {3,1,4,1,5};

  printAverage(a, 9);             // prints average
  swap(a, 3, 2);                  // swap elements 3 and 2
  reverse(a[1]);                  // reverse the digits in a[1]
  if (isPositive(a[0])) cout << "Positive" << endl;
                                   // prints: Positive
  cout << midEntry(a, 5) << endl; //  prints: 4
  return 0;
}
```

a) Title line for **printAverage**

b) Title line for **swap**

c) Title line for **reverse**

d) Title line for **isPositive**

e) Title line for **midEntry**

5) (prac2.pdf) Write title lines (header lines or prototypes) for the following functions. Do not supply the blocks for the functions.

(a) A function called **add3** which returns the sum of three double parameters.

(b) A function called **reverseIt** that returns the number obtained by reversing the digits in an integer parameter.

(c) A function called **randomArray** that sets the values in an array of doubles to have random values.

(d) A function called **addTwo** that adds 2 to every entry in an array of integers.

(e) A function called **biggerAverage** which determines which of two arrays of integers has the bigger average. It should return the value of this bigger average.

Tracing through a program for output:

6) (prac2.pdf) Consider the following C++ program.

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

void mystery(int data[], int p, int q) {
     data[p] = data[q];
     data[q] = data[p];
}

void m2(int &p, int q) {
     int temp = p;
     p = q;
     q = temp;
}
void print(int data[], int p) {
     for (int i = 0; i < p; i++)
          cout << data[i] << " ";
     cout << endl;
}
int main() {
     int x[8] = {0, 1, 2, 3, 4, 5, 6, 7};
     int y[7] = {0, 1, 2, 3, 4, 5, 6};
     int a = 3, b = 4;

     print(x, 3);                              // line (a)
     mystery(x, 1, 2); print(x, 5);            // line (b)
     for (int i = 1; i <= 7; i++) mystery(x, 0 ,i);
     print(x, 8);                              // line (c)
     m2(a, b); cout << a << b << endl;   //  line (d)
     m2(y[3], 7); print(y, 6);                 // line (e)
     return 0;
}
```

(a) What is the output at line (a)?

(b) What is the output at line (b)?

(c) What is the output at line (c)?

(d) What is the output at line (d)?

(e) What is the output at line (e)?

7) (prac3.pdf) Consider the following C++ program.

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

int recursive (int x) {
      if (x < 5) return 3;
      return recursive (x / 3) + x % 6;
}
char swap (int x, int y) {
      x = y;
      y = x;
      cout << x << y;
      return 's';
}
void set (int arr []) {
      arr[0] = 1; arr[1] = 9; arr[2] = 6; arr[3] = 8; arr[4] = 3;
}

int main() {
      int x[5];
      set(x);
      swap(1, 2); cout << endl;                                    //line (a)
      set(x);
      cout << x[0 + 2] << x[0] + 2 << endl;                        //line (b)
      cout << swap(1, 2) << endl;                                  //line (c)
      for (int i = 1; i < 4; i++) cout << x[i]; cout << endl;      //line (d)
      int e = 21;
      cout << recursive(e) << endl;                                //line (e)
      return 0;
}
```

(a) What is the output at line (a)?

(b) What is the output at line (b)?

(c) What is the output at line (c)?

(d) What is the output at line (d)?

(e) What is the output at line (e)?

Short Blocks of code:

8) (prac3.pdf) Write blocks of code to perform the functions used in the following main program. Your blocks must match the given title lines. Each block should be a short function of only a few lines.

```
int main() {
      int i = 2;
      int x[5] = {3, 1, 4, 1, 5};
// (a) Return the sum. Here 4 is printed.
      cout << add(i, 2) << endl;
// (b) Return number of odd entries. Here 4 is printed.
      cout << numOdd(x, 5) << endl;
// (c) Multiply i by 2. Here 4 is printed.
      doubleIt(i); cout << i << endl;
// (d) Find the index of the largest entry. Here 4 is printed.
      cout << findIndexMax(x, 5) << endl;
// (e) Return the absoluteValue. Here 4 is printed.
      cout << absoluteValue(i) << endl;
return 0;
}
```

(a)

(b)

(c)

(d)

(e)

9) (prac3.pdf) Write blocks of code to perform the functions used in the following main program. Your blocks must match the given title lines. Each block should be a short function of only a few lines.

```
int main() {
      int x = 5;
      double e = 2.718;
      double a[4] = {1.0, 2.0, -3.0, -4.0};
      double b[2] = {5.5, 4.5};
// (a) Changes the sign. Here to -5
      changeSign(x);
// (b) Return first digit after decimal point. Here 7 is printed.
      cout << firstDecimal(e) << endl;
// (c) Return the number of negative entries. Here 2 is printed.
      cout << numberNeg(a, 4) << endl;
// (d) Test whether the first argument is a factor of the second. Here: Yes
      if (isFactor(7, 14)) cout << "Yes\n";
// (e) print average of all entries both arrays: Here 1.0 is printed.
      averageArrays(a, 4, b, 2);
      return 0;
}
```

(a)

(b)

(c)

(d)

(e)