Review Sheet 2a

CS 70: Data Structures and Program Development

Tuesday, January 28, 2020

1. Declaring variables

```
int x = 3;
// Read: "x is an int whose value is 3."
```

C++ variables have: a name, a type, a value, and a location in memory.

   1. Who chooses these four?

   2. Which of these four can change while the program runs?

   3. Suppose we pause the running program and read the bits in memory. Which can we see?

2. The Life-Cycle of C++ Data

Every *individual* piece of data, over the course of its life:

1. **Allocation**: acquire memory for the data
2. **Initialization**: create the data
3. **Use**: read and/or modify the data
4. **Destruction**: clean up the data
5. **Deallocation**: relinquish the data's memory

3. For local variables

1. **Allocation**: at the opening { of the function
2. **Initialization**: Line of declaration (for parameters, the opening '{')
   - If you don't specify, default initialization
   - For primitives, default initialization does nothing! (So initial value is undefined).
3. **Use**: from initialization to destruction
4. **Destruction**: ending '}' of the declaring block
   - For primitive types, destruction doesn't do anything
   - But after destruction you can't use the variable
5. **Deallocation**: ending '}' of the function

4. Stack? Life Cycles?

```
int triple(int multiplier)                          // 1
{                                                   // 2
   int product = 3 * multiplier;                    // 3
   return product;                                  // 4
}                                                   // 5

int main()                                          // 6
{                                                   // 7
    int myInt;                                      // 8
    cout << "Enter an even number: " << endl;       // 9
    cin >> myInt;                                   // 10
    if (myInt % 2 == 0) {                           // 11
        int result = triple(myInt);                 // 12
        cout << result << endl;                     // 13
    }                                               // 14
    else {                                          // 15
        cout << "Not even!" << endl;                // 16
    }                                               // 17
    return 0;                                       // 18
}                                                   // 19
```

5. Stack? Life Cycles?

```
int absCube(int base)              // 1
{                                  // 2
    int outcome = base * base;     // 3
    outcome = outcome * base;      // 4
    if (outcome < 0) {             // 5
        outcome = -outcome;        // 6
    }                              // 7
    return outcome;                // 8
}                                  // 9

int main()                         // 10
{                                  // 11
    int myInt = 0;                 // 12
    int myConstant = -3;           // 13
    myInt = absCube(myConstant);   // 14

    cout << myInt << endl;         // 15

    return 0;                      // 16
}                                  // 17
```

6. What are Arrays?

-
-
-
-
-

7. Declaring an Array

```
int values[42];
```

(What is `values[5]`?)

~~`int values[]`~~

8. Declaring an Array: Variable Size

```
const int DAYS_IN_WEEK = 7;
int payments[DAYS_IN_WEEK];
```

~~`int x = 42;`~~

~~`int values[x];`~~

9. Declaring an Array: List Initialization

```
int payments[DAYS_IN_WEEK] = {10, 5, 5, 5, 5, 5, 10};
```

```
int values[42] = {1, 2, 3};
```

(What is `values[5]`?)

10. Array Idiom

```
int payments[DAYS_IN_WEEK];

for (size_t day = 0; day < DAYS_IN_WEEK; ++day) {
   cin >> payments[day];
}
```

11. What happens if we write:

```
int values[3] = {1, 2, 3};
cout << values[10000] << endl;
```