CS 102
Introduction to Programming Using C++

Chapter 7

Pointers

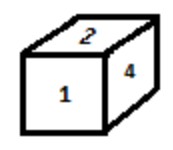
Homework

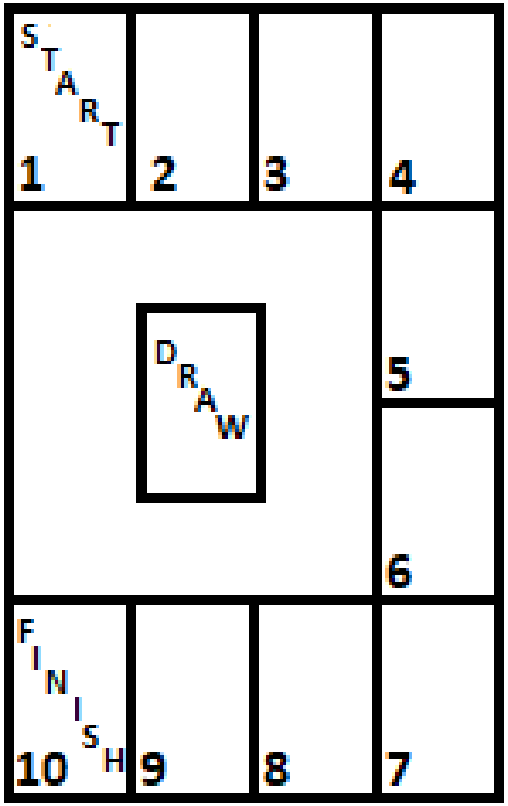
- Written homework
- R7.1, 2, 4, 7, 8, 15, 16, 19, 21 to 27

- Programs
- p. 344, choose one of P7.1, 2, or 4.
- Also, implement a linked list.

A Game

- A picture of the game





Playing the Game

- The players start at START and take turns
- On your turn
  - You roll the die
    - The number on the die tells which card you take from the draw pile
  - You take the correct card from the draw pile
  - You move the number of spaces shown on the card
- The first person to reach "FINISH" wins

Analyzing the Game

- The number on the die did not tell you how many squares to move
- Instead, it told you which card to take
- The card told you how many squares to move

Pointers

- A pointer is like the die
- A pointer doesn't contain the data itself
  - The die didn't tell you how many squares to move
- A pointer just tells you where to go to find the data
  - You used the die to find a card in the pile
  - Then, the card told you how to move

Pointer = Address

- From now on, we will think of a pointer as containing a number
- This number can be 0, 1, 2, 3, …
- It is not the data we need
- It contains the address *in memory* of the data
- Example: A pointer contains 6
  - This means the data is in location 6

Is This Like an Index into an Array?

- Yes, it's exactly like that
  - The only difference is that a pointer points to actual memory, not just the cells in an array
- If a pointer contains 0, that means *actual* byte 0
  - This is the very first byte in memory
- All of Chapter 7 is about pointers

Symbols to Know
Symbol 1: *

- The symbol * is used to declare a pointer variable
- The declaration

  double* data_ptr;

- means that data_ptr is a pointer to a double precision value
- data_ptr does not actually contain a double precision value

Symbols to Know
Symbol 2: &

- An & in front of a variable gets the address of the data contained in the variable
- The declaration double data_val;
- creates an actual variable data_val that can contain a double precision value
- Then, in a program, you can code

  data_ptr = &data_val;
- This means to get the address of data_val and store it in data_ptr

The Word Dereference

- The word dereference means getting or using the value a pointer points to

Practicing with * and &

- Background:

  int* id_pointer;

  int sams_id = 2515;

- Now assume that sams_id is stored at location 32.
- What is the effect of this statement?

  id_pointer = &sams_id;

- What is the effect of this statement?

  cout << *id_pointer;

Uninitialized Pointers

- There is a danger here
- When you create a pointer, you have to make sure it points to something before using it
- What is wrong with this code?

```
char* user_code;

cout << *user_code;
```

The Value NULL

- Initializing a pointer variable to NULL is a safety feature
  - NULL is a value that points to nowhere
- This guarantees that any references to it will not cause weird behavior
  - Instead, references to it will cause the program to exit with an error message

Trying to Dereference a NULL Pointer

- I tried to run a program containing this code

  ```
  char* user_code = NULL;
  *user_code = 'A';
  ```

- I got a pop-up box with the error message

  a.exe stopped working

Actually Using NULL

- You can use the value NULL in a program
- You can check if a pointer is NULL before using it

```
if (my_pointer == NULL)
    //  Assign it to point to something before using it
else
    // Use the pointer
```

NULL Is Only for Pointers

- You can use NULL like any other value
  - Except that NULL is only for pointer variables
- Typically, NULL represents some kind of error or an uninitialized pointer

Tricky Coding—Avoid This

- It's common to declare multiple variables on one line

```
int client_count = 0, file_count = 0, user_count = 0;
```

- Be careful if you do that with pointer variables

Tricky Coding—Avoid This

- This code is deceptive

  int *pointer1, pointer2, pointer3;

- Only pointer1 is a pointer to an int
  - The rest are actual integers
- Each pointer needs its own *

accounts.cpp

- This is a program from Chapter 7 that uses pointers
- Let's examine it

A Note about the Program

- Notice that by creating account_pointer as a pointer to an account (really a double) we can reuse it
- The first part of the program is for Harry's account
- The second part of the program is for the joint account

A Bit of Confusion

- What is the syntax error in this code?

```
double joint_account = 0;
double* account_pointer = &joint_account;
account_pointer = 1000;
```

- How can we fix the code?

Call By Value

- Recall that, by default, C++ is call-by-value
- Suppose we have the function incr (x):

```
void incr (int x)
{
    x++;
}
```

- This function does not actually increment x
- It didn't actually get x

Reference Parameters

- If you code

```
void incr (int &x)
{
    x++;
}
```

- then x can be changed
- x is called a reference parameter

A Reference Parameter is a Pointer

- Here is a function

```
void withdraw (double& balance, double amount)
{
    if (balance >= amount)
        balance = balance – amount;
}
```

A Reference Parameter is a Pointer

- Here is the same code using a pointer

```
void withdraw (double* balance, double amount)
{
    if (*balance >= amount)
        *balance = *balance – amount;
}
```

Is There Really a Difference?

- Yes!
- You call the first function with

  withdraw (harrys_account, 100);
- and the second with

  withdraw (&harrys_account, 100);

What Is the Difference?

- It's a question of who has to do the work—the compiler or you
  - The compiler translates reference parameters into pointers
- There may be other considerations

Questions?

- Are there any questions?