PIC 10A 2B

TA: Bumsu Kim

UCLA

Today…

- Functions
  - Review
  - Passing by Reference vs. Passing by Value
- Exercises Problems
  - Swap
  - Coding Concepts and Practices

UCLA

Functions

Basic Syntax and Terminology

(R) Functions

- Functions can be
  - Declared and then defined later
    - If you declare the function first (not define it right now) then need ;
  - Defined when it is declared (so both are done at the same time)

- The declaration determines a “signature” of the function
  - [Return Type] [Function Name] [Input Parameters] (and other options later)

```
double find_max(double a, double b);
```

UCLA

(R) Functions

- The declaration determines a "signature" of the function
  - [Return Type] [Function Name] [Input Parameters] (and other options later)

```
double find_max(double a, double b);
```

- Return Type
  - The type of the expression returned by the function
  - If the function returns nothing, can be `void`
- Function Name
  - Name of the function; cannot be a reserved word, the same naming rule applies with variables
- Input Parameters
  - The objects passed to the function
  - Number of arguments can be 0, 1, or more
  - Still need parentheses () even when the function gets zero parameters

UCLA

(R) Functions – Examples

- A function returning the maximum of two numbers

```
double find_max(double a, double b) {
        return (a > b) ? a : b;
}
```

Recall: The ternary operator

```
A ? B : C
```

is (almost) equivalent to

```
if (A) { B; }
else {C;}
```

- A ***procedure*** is a function returning `void`
  - Which means, it doesn't return anything
  - Example: a function printing the max to the console

```
void print_max(double a, double b) {
    cout << "The max of " << a << " and " << b
         << " is " << find_max(a, b);
}
```

Possible if `find_max` is ***declared*** before this expression

Q: [`true` or `false`?]
Using "`return`" keyword in a ***procedure*** results in a syntax error.

UCLA

(R) Functions – Examples

- A function that inputs several different types of arguments

```
void func(unsigned int i, string str) {
        cout << str[i];
}
```

Here `i` cannot be negative

- A ***predicate*** function is a function returning `bool`
  - The following function checks whether the first letter of the string is capitalized or not

```
bool isCapitalized(string str) {
    if ('A' <= str[0] && str[0] <= 'Z') {
        return true;
    } else {
        return false;
    }
}
```

True if `str[0]` is between ‘A’ and ‘Z’ (otherwise it is not a capital letter)

UCLA

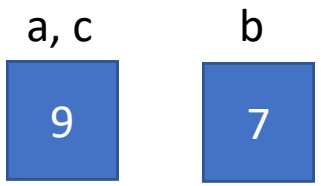
References

- A **reference** to a variable is just "another name" of the other

```
T tvar;                    // variable of type T (can be int, double, string, etc)
T& ref_tvar = tvar;        // a reference to a type T variable
```

- They refer to the same "address" in the memory

```
int a = 5;
int b = a;
int& c = a;
cout << a << ' ' << b << ' ' << c << endl;
b = 7;
cout << a << ' ' << b << ' ' << c << endl;
c = 9;
cout << a << ' ' << b << ' ' << c << endl;
```




```
5 5 5
5 7 5
9 7 9
```

Passing Parameters by Reference

- The default behavior of passing parameters to a function is to "pass by copy"
  - Also called "pass by value"
- Parameters are copied to a local variable

```
bool isPalindrome(string str) {
        // ftn body ...
}

int main() {
    string huge_str = "ABC..xyz..123...\n\n...";
    if (isPalindrome(huge_str)) {
        // ...
    }
}
```

The string variable `huge_str` is copied to a local string variable `str`

- If the parameter has huge size (in terms of memory), then copying it is inefficient

UCLA

Passing Parameters by Reference

- Solution to this inefficiency ⍰ Use a reference!
- Recall: a reference is just another name of the variable, so unnecessary copies don't occur

```
bool isPalindrome(string& str) {
        // ftn body ...
}

int main() {
    string huge_str = "ABC..xyz..123...\n\n...";
    if (isPalindrome(huge_str)) {
        // ...
    }
}
```

str is now "*a reference to a string*" (type name) and it refers to huge_str

- However, as we've seen before, this is dangerous since str can be modified in the function, and we don't want this change to affect the original huge_str

UCLA

Passing Parameters by **Const** Reference

- Solution to this danger ⍰ Use a **const** reference!
- A const reference is a reference whose value cannot be modified (so it's safe)

```
bool isPalindrome(const string& str) {
        // ftn body ...
}

int main() {
    string huge_str = "ABC..xyz..123...\n\n...";
    if (isPalindrome(huge_str)) {
        // ...
    }
}
```

str is now "*a reference to a **const** string*" (type name) and it refers to huge_str

- Here, we pass a huge string object in a **safe** and **efficient** way
  - No unnecessary copies occurred
  - No changes in the original object

UCLA

Passing Parameters by Reference

- Passing by non-const reference can be used to change multiple parameters in a single function evaluation
- Say we have three variables a, b, c and want to increment them by two

```
int a = 2, b = 3, c = 4;
cout << a << ' ' << b << ' ' << c << endl;
add2(a, b, c);
cout << a << ' ' << b << ' ' << c << endl;
```

Desired output:

```
2 3 4
4 5 6
```

- What should be ***the signature of add2***?

```
void add2(int& x, int& y, int& z)
```

- Implementation?

```
{ x+=2; y+=2; z+=2; }
```

UCLA

Exercise (Swap)

- Q) Suppose a,b,c are int type variables that have been initialized appropriately. Reorder the following lines to swap the values of a and b. The value that c stores doesn't matter.

```
(1) a = b;
(2) b = c;
(3) c = a;
```

Exercise (Swap)

- Q) Suppose a,b,c are int type variables that have been initialized appropriately. Reorder the following lines to swap the values of a and b. The value that c stores doesn't matter.

```
(1) a = b;
(2) b = c;
(3) c = a;
```

- A) (3)-(1)-(2)

UCLA

Exercise (Swap function)

- Write a function that swaps two variables (that are passed by reference)

UCLA

Exercise (Variables)

- Q) Which of the following are literals of type `float`? (Choose all correct choices.)

  - ◯ `23.45`
  - ◯ `23.45f`
  - ◯ `23.45F`
  - ◯ `23E0`
  - ◯ `23E0f`
  - ◯ `23e0F`
  - ◯ None of the above.

UCLA

Exercise (Variables)

- Q) Which of the following are literals of type `float`? (Choose all correct choices.)

  - ◯ `23.45`
  - ◯ `23.45f`
  - ◯ `23.45F`
  - ◯ `23E0`
  - ◯ `23E0f`
  - ◯ `23e0F`
  - ◯ None of the above.

A) B, C, E, F

UCLA

Exercise (Roman Digits)

- Write the following functions:
  - 1. “roman_digit” which gets a number between 0 and 9, and roman characters for one/five/ten. This function returns a string that corresponds to a roman numeral

```
e.g.   roman_digit(7, 'I', 'V', 'X') == "VII" // 7
       roman_digit(9, 'I', 'V', 'X') == "IX"  // 9
       roman_digit(7, 'X', 'L', 'C') == "LXX" // 70
       roman_digit(9, 'C', 'D', 'M') == "MC"  // 900
```

  - 2. “num2roman” which gets a number between 0 and 1000, and returns a roman numeral

```
e.g.   num2roman(999) == "CMXCIX"
       num2roman(765) == "DCCLXV"
```

- Functions are useful for **breaking a larger problem down to smaller pieces**

UCLA

Your Feedback is welcome

- Don’t hesitate to give a feedback on the discussion
- Use the link on my Github repo, or the link below:
  - https://forms.gle/erZj1iSgHNrHQuXk6

My Github repo on the web looks like:

code	Week2 Tu
LICENSE	Initial commit
README.md	Update README.md

README.md

PIC10A

PIC10A discussion 2B, UCLA for Fall 2022

Google form link for feedbacks: https://forms.gle/erZj1iSgHNrHQuXk6

Click this link

UCLA