CS161: Introduction to Computer Science I

Week 7 – Functions

11/2020

What is in CS161 today?

fit@hcmus

- Introduction to **functions**
  - What is a function?
  - Why would you want to use a function?
  - How do you define functions?
  - How do you call functions?
- Functions with **Arguments**
  - What are arguments?
  - How do we define a function with args?
  - Actual arguments versus Formal arguments
- Functions with Arguments
  - **Call by value** vs. **Call by reference**

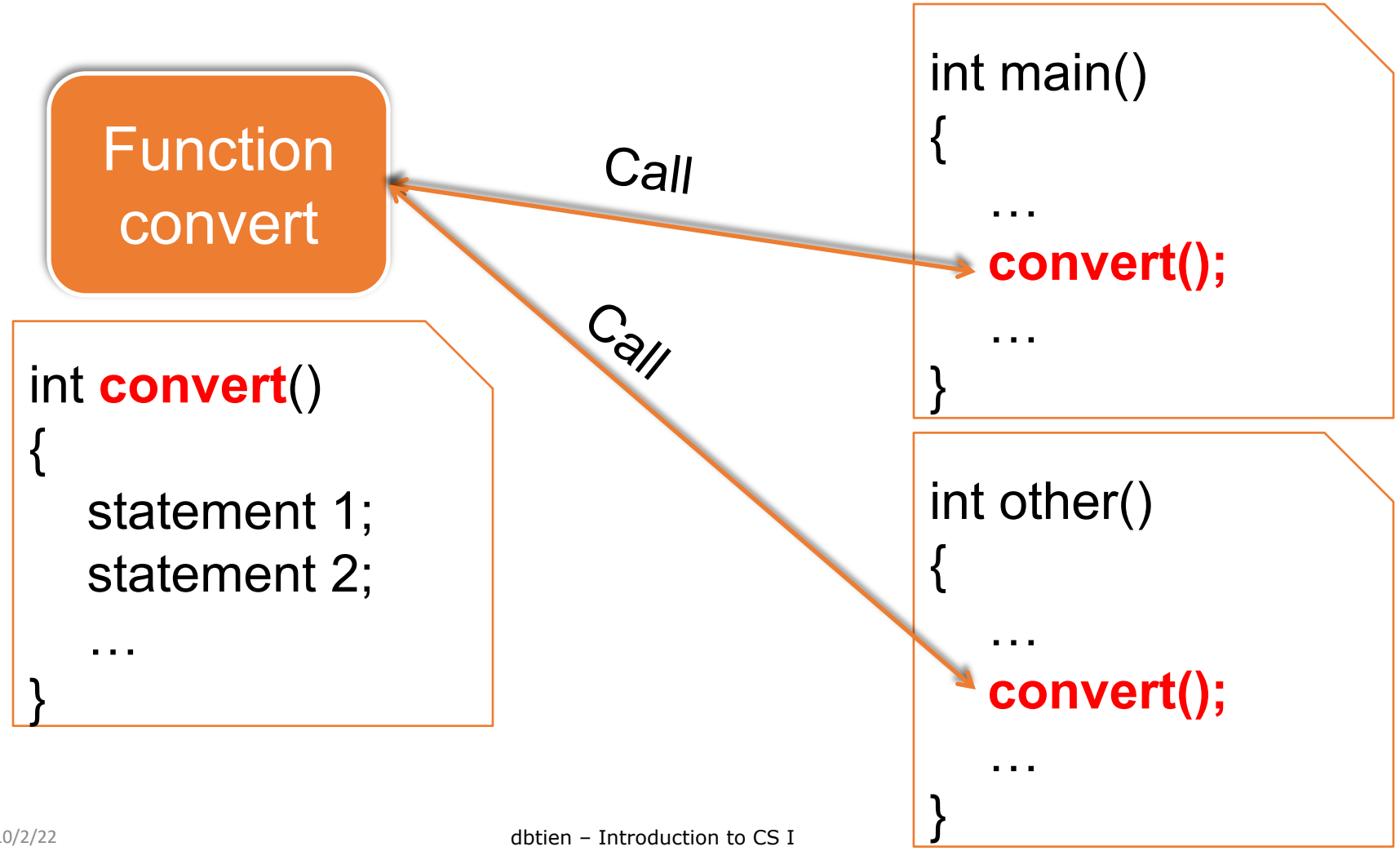
Functions: What are they?

- We can write our own **functions** in C++
- These functions can be called from your **main** program or from **other functions**
- A C++ function consists of a **grouping of statements** to perform a certain task
- This means that all of the code necessary to get a task done **doesn't** have to be in your main program
- You can begin execution of a function by **calling** the function

Functions: What are they?

fit@hcmus

- We can write our own **functions** in C++



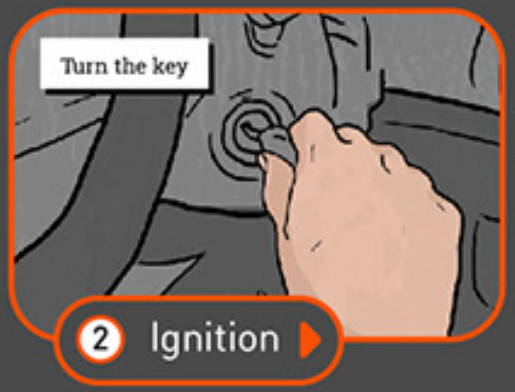
Functions: What are they?

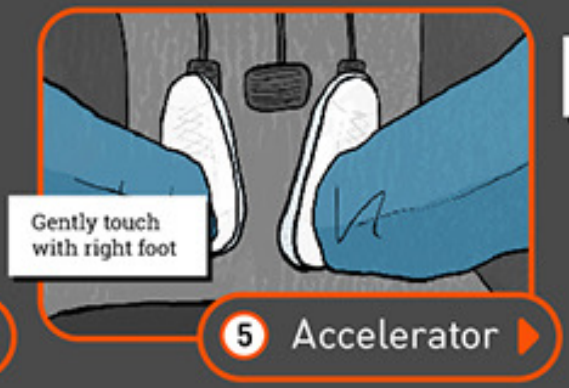
- When we write algorithms, we should divide our programs into a series of major tasks...
  - where each major task is a **function**, called by the **main** program
- We can group together statements that perform a distinct task and give the overall action a name.
- This is accomplished by writing a C++ function.

Functions: What are they?

- For example, tasks such as driving a car, or cooking breakfast are every day functions that we use.
- The exact details of driving a car or cooking are hidden in the actual process, but even though you don't know the details -- just from the statement "driving a car" you know what that involves and what I am talking about. I don't need to have to tell you that first I get out my keys, then unlock the car door, then get inside, then.....

How to drive a manual car





To avoid stalling

- DON'T take your foot off the clutch too quickly
- DON'T be too light on the revs

If you do stall

- Turn off engine, return gear stick to neutral and try again!

How to take a bath

Functions: What are they?

- The same thing applies to functions in C++.
- A function has a **name** assigned to it and contains a sequence of statements that you want executed every time you invoke the function from your main program!

Functions: What are they?

- Data is passed from one function to another by using **arguments** (in parens after the function name).
- When no arguments are used, the function names are followed by: " **( )** ".

```
int CONV(int arg1, int arg2)
{
   statement 1;
   statement 2;
   ...
}
```

```
int main()
{
   ...
   CONV(a, b);
   ...
}
```

Functions: Defining Them...

fit@hcmus

- The syntax of a function is very much like that of a main program.
- We start with a function header, followed by variable definitions and executable statements:

```
data_type function_name()
{
    <variable definitions>
    <executable statements>
}
```

Functions: Defining Them...

- A function must always be declared before it can be used
  - This means that we must put a one-line function declaration at the beginning of our programs which allow all other functions and the main program to access it.
  - This is called a **function prototype** (or **function declaration**)

```
data_type function_name();
```

  - The function itself can be defined anywhere within the program.

Functions: Using Them...

- When you want to use a function, it needs to be CALLED or INVOKED from your main program or from another function.
- If you never call a function, it will never be used.
- We have all had experience calling functions from the math library (e.g., the pow function)
- To call a function we must use the function call operator ()

```
some_variable = pow(x,3);
```

Functions: Calling pow...

- When we called the power function, we are temporarily suspending execution of our main program (or calling routine) and executing a function called **`pow`** that someone else has written.
- It takes two values as arguments (**`x`** and **`3`**), called **actual arguments** and returns to the calling routine the result (a floating point value)

Let's try writing a function!

fit@hcmus

- What might the major steps be for a program that calculates the day of the year based on a given date:
  - Welcome the user
  - Get the date from the user
  - Calculate the day of the year
  - Display the result
  - Signoff message
- Each one of these steps could be written as a function...

Let's try writing a function!

fit@hcmus

```
void welcome();   //function prototype

int main()  {
   welcome();      //function call
   ...
   return 0;
}

void welcome()  {  //function definition
   cout << "Welcome to .... ";
}
```

Let's try writing a function!

- Notice that in this example we use a **function prototype** for our function declarations.
- They are very similar to the function header except that they must be terminated by a semicolon... just like any other declaration in C++.

Why write functions?

- You might ask, why go through the trouble to write a program that does no more than the original, shorter version?
- One reason is that functions can be used as prefabricated parts for the easy construction of more complicated programs.
- Another reason is that a function - once created - can be called any number of times without writing its code again.

Why write functions?

- As our programs get more complicated, it is really important that you clearly understand the order in which statements are executed.
- The main program runs first, executing its statements, one after another.
- Even though the functions are declared before the main program (and may also be defined before the main program), they are not executed until they are called.
- They can be called as many times as you wish

Why write functions?

- By giving the task a name, we make it easier to refer to.
- Code that calls clearly named functions is easier to understand than code in which all tasks are described in the main program.
- Programs that use functions are easier to design because of the way they "divide and conquer" the whole problem.

Why write functions?

- By having a function perform the task, we can perform the task many times in the same program by simply invoking the function repeatedly.
- The code for the task need not be reproduced every time we need it.
- A function can be saved in a library of useful routines and plugged into any program that needs it. (like we have seen with the pow function)

Why write functions?

- Once a function is written and properly tested, we can use the function without any further concern for its validity.
- We can therefore stop thinking about how the function does something and start thinking of what it does.
- It becomes an abstract object in itself - to be used and referred to.

Why write functions?

- Functions enable us to implement our program in logically independent sections as the same way that we develop the solution algorithm.
- Our main for could be:

```
welcome();
get_date(month, day, year);
day = calculate (month, day, year);
display(day);
signoff();
```

Some details about functions:

- Each function declaration can contain declarations for its own... this includes constants, variables.
- These are considered to be LOCAL to the function and can be referenced only within the function in which they are defined

```
data_type function_name()
{
   data_type variable; //local variable
}
```

Some details about functions:

fit@hcmus

```
#include <iostream>
using namespace std;
int get_asterisk(void);
int main(){
    int number;                 //local variable
    number = get_asterisk();
    ...
    return 0;
}

// put comments here describing the purpose of this function!
int get_asterisk () {
      int num_asterisk;     //local variable

      cout << "How many asterisks would you like?" << endl;
      cin >> num_asterisk;
      return(num_asterisk);
}
```

Some details about functions:

- To have a function return a value - you simply say "`return expression`".
- The expression may or may not be in parens.
- Or, if you just want to return without actually returning a value, just say `return;` (note: return(); is illegal).
- If you normally reach the end of a function (the function's closing "}"), its just like saying `return;` and no value is returned.

Some details about functions:

- For functions that don't return anything, you should preface the declaration with the word "`void`".
- When using void, it is illegal to have your return statement(s) try to return a value
- Also notice, that the type of a function must be specified in both the function declaration and in the function definition.

Functions: What are arguments?

fit@hcmus

- If we want to send information to a function when we call it, we can use arguments
- For example, when we supplied two items within the parentheses for the **pow** function -- these were arguments that were being passed to the function **pow**!
- We can define functions with no arguments, or with many arguments

Functions: What are arguments?

- **If we go back to our example of converting inches to millimeters...**
  - if we write a function to perform the calculations, we would need to somehow send to the function the number of inches to convert
  - this can be done by passing in the number of inches as an argument
  - and receiving the number of millimeters back as the returned value

Functions: What are arguments?

- For example, from our main program we could say:

```
float convert (float inches);    //prototype
int main() {
      float in;  //local variable to hold # inches
      float mm;  //local variable for the result
      cout << "Enter the number of inches: ";
      cin >> in;
      mm = convert (in);    //function call
      cout << in << " inches converts to "
            << mm << "mm";
      return 0;
}
```

Functions: What are arguments?

- Then, to implement the function we might say:

```
float convert (float inches) {
    float mils;       //local variable
    mils = 25.4 * inches;
    return mils;      //return (mils);
}
```

Functions: What are arguments?

- Notice that we can have arguments to functions!
- These must be in the function header for both the function declaration (prototype) and function definition.
- In this example, `inches` is a variable...which is a argument because it is defined in the function header.

Functions: What are arguments?

fit@hcmus

- **When you call convert,**
  - you are establishing an association between the main program's `in` variable
  - and the function's `inches` variable;
  - this function does some calculations,
  - and returns a real number which is stored in the calling routines `mm` variable.

Functions: What are arguments?

fit@hcmus

- **Notice that variables are declared in a function heading;**
  - these are FORMAL ARGUMENTS
  - they look very much like regular variable declarations, except that they receive an initial value from the function call
- The arguments in the function call (invocation) are called ACTUAL ARGUMENTS.

Functions: What are arguments?

fit@hcmus

```
float convert (float inches);
int main() {
      float in;      //local variable to hold # inches
      float mm;      //local variable for the result
      cout << "Enter the number of inches: ";
      cin >> in;
      mm = convert (in);      //function call
      cout <<in << "inches converts to" << mm << "mm";
      return 0;
}
float convert (float inches) {
      return 25.4 * inches;
}
```

Formal Argument

Actual Argument

Functions: What are arguments?

fit@hcmus

- When the function call is executed,
  - the actual arguments are conceptually copied into a storage area local to the called function.
  - If you then alter the value of a formal argument, only the local copy of the argument is altered.
  - The actual argument never gets changed in the calling routine.

Functions: What are arguments?

- C++ checks to make sure that the number and type of actual arguments sent into a function when it is invoked match the number and type of the formal arguments defined for the function.
- The return type for the function is checked to ensure that the value returned by the function is correctly used in an expression or assignment to a variable.

Functions: What are arguments?

fit@hcmus

- When we deal with FORMAL VALUE ARGUMENTS...
  - the calling actual argument values cannot be modified by the function.
  - This allows us to use these functions, giving literals and constants as arguments without having conflicts.
  - This is the default way of doing things in C++.

Let's write a function to sum two numbers:

fit@hcmus

```
int sumup(int first, int second);     //function prototype
int main() {
    int total, number, count;
    total = 0;
    for (count = 1; count <= 5; count++) {
      cout << " Enter a number to add: ";
      cin >> number;
      total = sumup(total, number);     //function call
    }
    cout << " The result is: " << total << endl;
    return 0;
}
int sumup(int first, int second) {    //function definition
    return  first + second;
}
```

Functions: Value vs. Reference

- **Call by value brings values into a function (as the initial value of formal arguments)**
  - that the function can access but not permanently change the original actual args
- **Call by reference can bring information into the function or pass information to the rest of the program;**
  - the function can access the values and can permanently change the actual arguments!

Functions: Value vs. Reference

- Call by value is useful for:
  - passing information to a function
  - allows us to use expressions instead of variables in a function call
  - value arguments are restrained to be modified only within the called function; they do not affect the calling function.
  - can't be used to pass information back, except through a returned value

Functions: Value vs. Reference

- Call by reference is useful for:
  - allowing functions to modify the value of an argument, permanently
  - requires that you use variables as your actual arguments since their value may be altered by the called function;
  - you can't use constants or literals in the function call!

Example of call by value & reference:


```
void convert (float inches, float & mils);
int main() {
     float in;     //local variable to hold # inches
     float mm;     //local variable for the result
     cout << "Enter the number of inches: ";
     cin >> in;
     convert (in, mm); //function call
     cout <<in << "inches converts to" << mm << "mm";
     return 0;
}
void convert (float inches, float & mils) {
     mils = 25.4 * inches;
}
```

Call by value

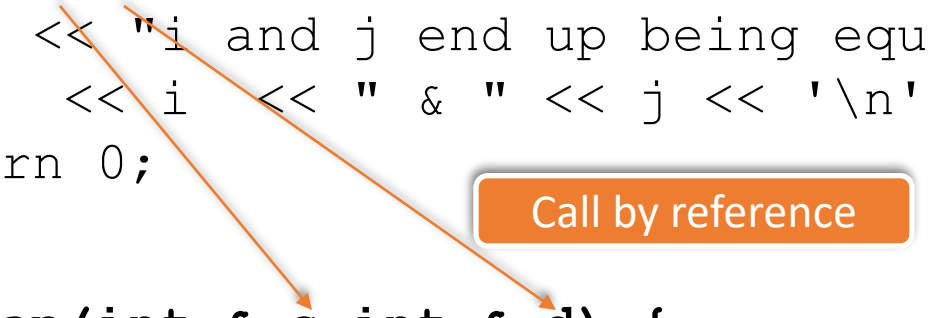
Call by reference

Example of call by reference:


```
void swap (int & a, int & b);

int main() {
    int i = 7, j = -3;
    cout << "i and j start off being equal to: "
         << i << " & " << j << '\n';
    swap(i,j);
    cout << "i and j end up being equal to: "
         << i  << " & " << j << '\n';
    return 0;
}

void swap(int & c,int & d) {
    int temp = d;
    d = c;
    c = temp;
}
```



Call by reference vs by value

fit@hcmus

pass by reference

cup =

fillCup( )

pass by value

cup =

fillCup( )

www.mathwarehouse.com

What kind of args to use?

- Use a **call by value** if:

  1) The argument is only to give information to the function - not get it back

  2) They are considered to only be IN parameters. And can't get information back OUT!

  3) You want to use an expression or a constant in function call.

  4) In reality, use them only if you need a complete and duplicate copy of the data

What kind of args to use?

fit@hcmus

- Use a **call by reference** if:

  1) The function is supposed to provide information to some other part of the program. Like returning a result and returning it to the main.

  2) They are OUT or both IN and OUT arguments.

  3) In reality, use them WHENEVER you don't want a duplicate copy of the arg...

Exercises

1. Write a function declaration (function prototype) and a function definition for a function that takes three arguments, all of type int, and that returns the sum of its three arguments.
2. Write a function declaration and a function definition for a function that takes one argument of type double. The function returns the character value 'P' if its argument is positive and returns 'N' if its argument is zero or negative.
3. Can a function definition appear inside the body of another function definition?
4. List the similarities and differences between how you invoke (call) a predefined (that is, library) function and a user-defined function

5. Write a function definition for a function called inOrder that takes three arguments of type int. The function returns true if the three arguments are in ascending order; otherwise, it returns false. For example, inOrder(1, 2, 3) and inOrder(1, 2, 2) both return true, whereas inOrder(1, 3, 2) returns false.
6. Write a function definition for a function called even that takes one argument of type int and returns a bool value. The function returns true if its one argument is an even number; otherwise, it returns false.
7. Write a function definition for a function isDigit that takes one argument of type char and returns a bool value. The function returns true if the argument is a decimal digit; otherwise, it returns false.

- Write a **function prototype** and **function definition** for a function **isDigit** that takes one argument of type char and returns a bool value. The function returns true if the argument is a decimal digit; otherwise, it returns false.