Module 20: void functions

Intro to Computer Science 1 - C++

Professor Scott Frees

Textbook

The following topic is covered in section 6.4 in the textbook

Always return data?

Our initial function *returned* the answer computed.

```
int sum(int x, int y) {
    int s = 0
    for ( int i = x; i <= y; i++ )
        s+= i
    return s;
}
```

However, if we **know** we'll always just print the answer, we have other options...

void functions

- Functions should only return data if it makes sense to return data!
- For example, if your function's job is to simply print an answer, you might not need to return anything

```
void print_sum(int x, int y) {
    int s = 0
    for ( int i = x; i <= y; i++ )
        s+= i
    cout << s << endl;
}
```

void functions

- Functions should only return data if it makes sense to return data!
- For example, if your function's job is to simply print an answer, you might not need to return anything

```
void print_sum(int x, int y) {
    int s = 0
    for ( int i = x; i <= y; i++ )
        s+= i
    cout << s << endl;
}
```

void?

`void` is *not* a data type - its the *absence of data.*

When declaring a function, you must decide on what kind of data you'll return.

You cannot choose to "sometimes" return integers, and other times not return anything!

Must you have parameters?

We've already seen a function that doesn't accept any parameters...

```
rand();  // returns a random number
```

Note, `rand()` doesn't need any information in order to do its job - so there are no parameters!

Making up your mind

It's critical that you can defend your decision to write every line and character of your program.

In particular, when writing functions - you should be able to clearly explain

1. why it declares the data type it does
2. why it accepts the parameters it does
3. why the parameters are the types they are

Simple tip: Never guess. Each line of code that you write must have a clear purpose behind it. If you don't know why you are writing something, **stop**. Think! Never write code "because that's what I saw in the book". *This tip is surprisingly difficult to follow sometimes - but it will pay huge dividends!*

Programming Example 24

Write a program that asks the user for a numeric (double) value between 0 and 100.

Write a function that accepts the numeric grade and prints out the letter grade (A, B, C, D, or F).

Programming Example 25

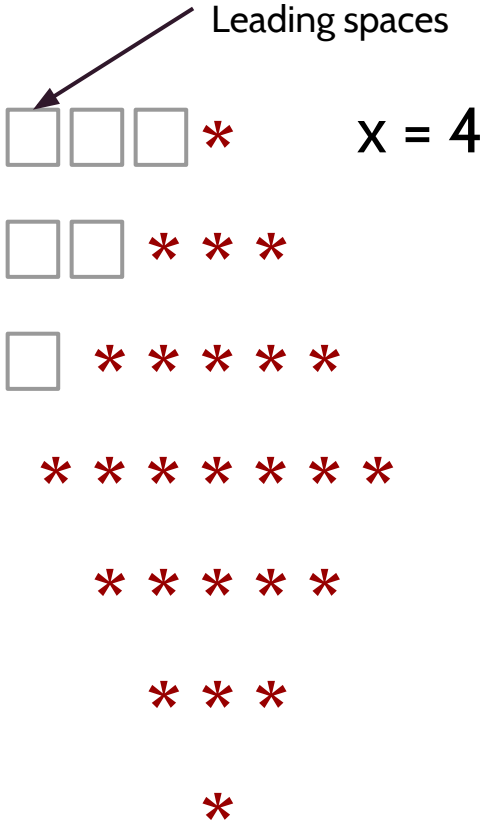
Modify the previous example:

- Using accumulation, calculate the numeric average of all grades.
- Re-use the same function to print out the grade.
- However - the program should clearly differentiate between a single grade and the average...

Lesson-learned

- In the original "print grade" example, the function did not return the letter grade, **it just printed it**.
- In the second example, we see that we actually have several reasons why we might call "print grade" (a single grade, or an average)
- Depending on *context*, the print statement might be different.
- In the second example, it makes a lot of sense to *return* the letter grade, and do the printing in main.
- **Reason**: The code in main knows ***why*** it called the letter grade function!

Lab 8



This lab builds on Lab 7, but includes the printing of both "spaces" and "stars"

You must create a function called **`printChars`** which accepts two parameters - the character to print, and the number of characters to print

You should call this function on each line, twice - once when printing the leading spaces, once when printing stars.