

183 Discussion

Week 3 – Diana Gage

www-personal.umich.edu/~drgage

Agenda

- Announcements
- More on cin
- Introduction to functions, RME's
- Explanation of scope
- Practice problems
- Discussion 2 Challenge!
- Looking ahead: conditionals

Announcements

- Project 1 due TONIGHT by 11:59:59pm!
- Last minute questions?
- ** Submit projects by Wednesday night for 5% extra credit, by Thursday night for 2.5%
- Incentive for starting early 😊
- Assignment 2 due a week from today!

More on cin

Using the **extraction operator** <<

- Ignores **leading** whitespace – whitespace is read in but not stored anywhere
- Ex. user types “ 2” instead of “2” → this is okay! “2” still stored in variable
- Reads in char by char
- Converts char(s) to specified data type (type variable expects)
- Stops reading **when it hits** whitespace, or a character of an unacceptable data type
- Will only **successfully** read in if the data type read in can fit into desired data type

cin Fail State

- When the data type of the value read in doesn't match the expected data type, or cannot be easily converted, cin will enter a **fail state**
- **NOTHING** else will happen (no more values read in, no more progress) until the fail state is cleared

What is read in? – 2 Examples

```
int main() {  
    int my_number = 0;           // initialize what we'll read into  
    cin >> my_number;           // user types "2"  
    return 0;  
}
```

```
int main() {  
    int my_number = 0;           // initialize what we'll read into  
    cin >> my_number;           // user types "2222hello!"  
    return 0;  
}
```

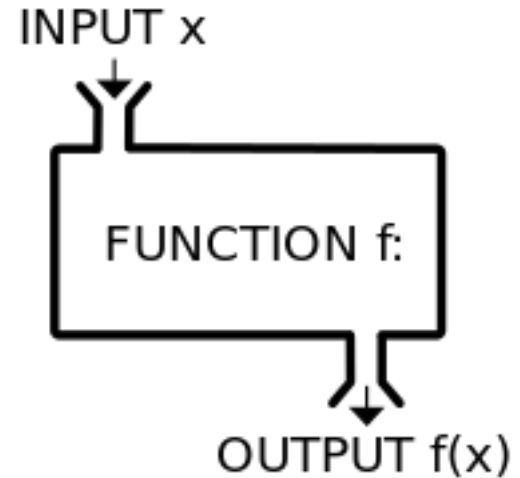
What is read in? – 2 Examples

```
int main() {  
    int my_number = 0;  
    cin >> my_number;           // my_number = 2  
    return 0;  
}
```

```
int main() {  
    int my_number = 0;           // my_number = 2222  
    cin >> my_number;           // cin enters fail state when hits "h"  
    return 0;  
}
```

Moving onto functions...

- **Definition:** list of statements that can be executed by calling its name → `int main()` is a function!
- A **function** is a block of code with a *specific task* within a program → essentially a **mini program**
- Functions often take in input values, do some work, and return a **single value** that will be used by whoever called the current function (i.e. `main` or another function)



Output (return value) is sent back to where function was called from

Moving onto functions...

- There are **library functions** (functions that already exist in different libraries available to you in C++)
 - Sqrt(), abs(), ceil(), floor() → include <cmath> library
- There are also **user-defined functions** (functions you create to develop your program)
- Functions help reduce duplication of code
- You can reuse them with **different parameters (inputs)** to complete a specific task → you don't have to type the same calculations over and over again! Just use a function

Thinking about functions...

Separate print statements for each ingredient vs.
One function that takes in ingredient and needed amount, and prints these out

```
How many people do you need to serve? 3
```

```
You need to make: 1 batch of cupcakes
```

```
Shopping List for "Best Ever" Vanilla Cupcakes
```

```
-----  
1 bag of flour  
1 bag of granulated sugar  
1 pound of butter  
1 container of sour cream  
1 dozen eggs  
1 bag of powdered sugar  
1 bottle of vanilla
```

```
Total expected cost of ingredients: $17.84
```

```
Have a great party!
```

Functions

Most important elements of a function:
**name, parameters/inputs, output (what's returned),
return type, task/body**

Setting up a function:

- What should the function do?
- What descriptive name should it have?
- Will it take inputs? Why? What kind?
- Should it return a value? Why? What kind?
- How will it do the necessary work?

Add Function

```
int main() {  
    cout << add(2, 3);  
    return 0;  
}
```

main() calls add
function and passes it
two ints, 2 and 3

```
int add(int x, int y) {  
    int sum = x + y;  
    return sum;  
}
```

This works, because function
expects two ints!

parameters

add() uses **parameters** to calculate
sum (local int variable to add()) by
adding x and y

add() **returns** sum to where add()
was called from and exits

output

***What's add()'s
return type?***

Add Function

```
int main() {  
    cout << add(2, 3);  
    return 0;  
}
```

main() calls add
function and passes it
two ints, 2 and 3

```
int add(int x, int y) {  
    int sum = x + y;  
    return sum;  
}
```

This works, because function
expects two ints!

parameters

add() uses **parameters** to calculate
sum (local int variable to add()) by
adding x and y

add() **returns** sum to where add()
was called from and exits

output

***What's add()'s
return type?***

int!

Function Practice

Finish the following user-defined function, which says hello to whoever (the name) we pass into it as input

```
_____ say_hello (string _____) {  
    cout << _____ << _____ << endl;  
}
```

Function Practice

Finish the following user-defined function, which says hello to whoever (the name) we pass into it as input

```
void say_hello (string _____) {  
    cout << _____ << _____ << endl;  
}
```

Function Practice

Finish the following user-defined function, which says hello to whoever (the name) we pass into it as input

```
void say_hello (string name_in) {  
    cout << _____ << _____ << endl;  
}
```


Function Practice

Finish the following user-defined function, which says hello to whoever (the name) we pass into it as input

```
void say_hello (string name_in) {  
    cout << "Hello " << _____ << endl;  
}
```

Function Practice

Finish the following user-defined function, which says hello to whoever (the name) we pass into it as input

```
void say_hello (string name_in) {  
    cout << "Hello " << name_in << endl;  
}
```

Function Practice

Finish the following user-defined function, which says hello to whoever (the name) we pass into it as input

```
void say_hello (string name_in) {  
    cout << "Hello " << name_in << endl;  
    return; //ends the function  
}
```

Function Practice

- Now let's call this function from main()
- We need to:
- Declare and initialize the variable we will pass in
- Call our function

```
void say_hello (string name_in) ; //function declaration
```

```
int main(){  
    _____ name = "Jimmy";  
    say_hello(_____);  
}
```

Function Practice

- Now let's call this function from main()
- We need to:
- Declare and initialize the variable we will pass in
- Call our function

```
void say_hello (string name_in) ; //function declaration
```

```
int main(){  
    string name = "Jimmy";  
    say_hello(name);  
}
```

Function Practice

- Now let's call this function from main()
- We need to:
- Declare and initialize the variable we will pass in
- Call our function

```
void say_hello (string name_in) ; //function declaration
```

```
int main(){  
          name = "Jimmy";  
    say_hello("Jimmy");  
}
```

← Could we call say_hello() this way too?


Function Practice

- Now let's call this function from main()
- We need to:
- Declare and initialize the variable we will pass in
- Call our function

```
void say_hello (string name_in) ; //function declaration
```

```
int main(){  
    _____name = "Jimmy";  
    say_hello("Jimmy");  
}
```

Could we call say_hello()
this way too?
Yes!



Full Program

```
void say_hello (string name_in); // function declaration
```

```
int main(){  
    string name = "Jimmy";  
    say_hello(name);  
}
```

```
void say_hello (string name_in) {  
    cout << "Hello " << name_in << endl; // implementation  
}
```


Scope

- A variable can either have **local scope** or **global scope**
- Local scope
 - Exists only within **current function**
- Global scope
 - Exists for **all functions** in the program

Full Program && Scope

```
void say_hello (string name_in); // function declaration
```

```
int main(){  
    string name = "Jimmy";  
    say_hello(name);  
}
```

What is the scope of the variable `name`?

```
void say_hello (string name_in) {  
    cout << "Hello " << name_in << endl; // implementation  
}
```

What is the scope of the variable `name_in`?

Full Program && Scope

```
void say_hello (string name_in); // function declaration
```

```
int main(){  
    string name = "Jimmy";  
    say_hello(name);  
}
```

What is the scope of the variable `name`?
LOCAL to main()

```
void say_hello (string name_in) {  
    cout << "Hello " << name_in << endl; // implementation  
}
```

What is the scope of the variable `name_in`?
LOCAL to say_hello()

Global Variables

- A global variable is declared outside of any and all functions → *above everything so any function (main included) can use it!*
- Must be named in ALL CAPS
- Must be declared **const**

** Remember: **const** means the variables cannot be changed anywhere in the program

RME's

// REQUIRES:

// MODIFIES:

// EFFECTS:

- Put an RME above any user-defined function you create
- These are special types of comments specific to functions
- Functions given to you will have RME's
- RME's are meant to aid the user

Fix this Function!

```
int code_master(string n){  
  
    cout >> "Hello" >> n >> endl  
    "You are the code master!";  
    n = Code Master;  
    return n;  
}
```

Fix this Function!

```
int code_master(string n){  
  
    cout >> "Hello_" >> n >> endl_  
    _____ "You are the code master!";  
    n = Code Master;  
    return n;  
}
```

Corrected Version:

```
string code_master(string name_in){  
  
    cout << "Hello_" << name_in << endl;  
    cout << "You are the code master!";  
    name_in = "Code Master";  
    return name_in;  
}
```


Challenge Problem!

- Let's say we want a program that will print the **exact integer square root** of a number, if there is one.
 - ie $\sqrt{64} = 8$ //good, this is an integer
 - But $\sqrt{12} = 3.4641$ // no good
- Otherwise, it will tell the user that there is no exact square root, and round to the nearest integer

Challenge: Notes for Implementation

Remember:

- `#include <_____>`
- We need `iostream`, `string`, and `cmath`
- `using namespace std;`
- Function declarations
- `int main()`
- `return 0;` at the end of `main`
- Function implementations
- **ceil and floor functions**
- We will need write two of our own functions
- One will call the other!

**Follow this order
for your program!**