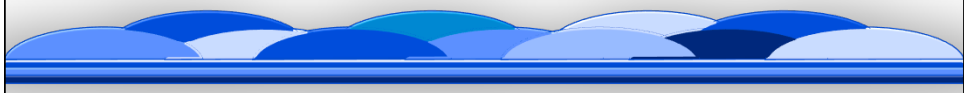
Topic 1- 1A Review – P4 - Functions



Announcements

- Assignment #1 – Intro to Functions

Why do we use functions?

- Divide and Conquer
  - break big complex problems into smaller comprehensible problems
  - This is how we build big programs
- Smaller pieces of code are easier to manage
  - Easier to code
  - Easier to test
  - Easier to collaborate with others
  - Supports reusability / modifiability
- Reduces the need for replicating code
  - Write it once – test it thoroughly – reuse over and over again
- Each function should have one task
  - Make it general when you can
    - Think about whether or not this might be useful in another program
    - Isolate differences by making them parameters
  - No Jack-of-all-trades functions!

Example

Write a program that will read in letter grades and output a GPA

- Break code into logical tasks (start with Main)
  - Think about what we need to accomplish
    - Input - validated
    - Get the grade pt value of a letter grade
    - Calculate Gpa
    - Output GPA
- Split these up into different functions
  - Each function processes one task
  - Think about code that needs to be replicated
  - Write them so they can work as independently as possible
  - Main should still indicate what the program does from a high level

Function basics

- All Functions must have a return datatype
  - This represents the datatype of the value it is returning
  - For main we use int
  - Some functions return some value
    - Value-returning functions have a return statement
    - The value is returned through the function call
  - Others do not
    - Void is the datatype for functions that don't return a value through the function call

Example

- A function that converts a letter grade into a grade point value
  - What values does it need to perform the task?
    - Those are our parameters
  - What kind of function is it? – does it return anything?
- A function that outputs a GPA
  - What values does it need ?
  - What does it return?

4

Function Process

For every function we need to..

1 - Declare the function

2 - Define the function

3 - Use the function

Using a Function

```
theSquareRoot = sqrt(num1);
```

The parens are required

Variable

Function Name

Argument

Argument → sometimes we need to send data to the function, we use arguments to do this.

Argument_List → a list of arguments that follows the function call

Variable → if our function returns a value we need a place to store it.

NOTE: This function returns a value so we need to either assign that value to a variable or cout it.

e.g. cout << sqrt(num1);

How do we define a function?

Syntax

```
returnType functionName ( type parameterName...)
{
    statements;
    return value;
}
```

- Somewhere in statements you need to have a return statement.
- The function terminates when the return statement is executed
- The return statement returns the value of the code.

Example

```
int AddTwoInts(int num1, int num2)
{
    return num1 + num2;
}
```

NOTE:
We have to let the compiler know what the types of our parameters (incoming data)

Declaring a function

We can declare them

Somewhere in our main source file

(main.cpp for example) → before int main () We have to tell the compiler about the function

We do this using a function prototype

Syntax
returnType functionName (type parameterName...);

Example

```
int AddTwoInts(int num1, int num2);
```

Note: this is exactly like the first line of our function definition except the ;

Why use a Prototype?

We need the prototype to be first because the compiler reads from top to bottom

1. We can put it in the same file as our int main() → Before the int main()
2. We can put it in a separate header file
   - We will use header files in this class
3. We can define the function before the int main() { } (rather then declaring the prototype) and then we won't need the prototype
   - This will require that our functions appear in a particular order making our code hard to maintain

NEVER use this option!
This will make code confusing
To follow!

Putting it all together

```
#include <iostream>
using namespace std;

int AddTwoInts(int num1, int num2);

int main ()
{
      int n1, n2, sum;
      cout << "Enter the first value: ";
      cin   >> n1;
      cout << "Enter the second: ";
      cin   >> n2;
      sum = AddTwoInts(n1, n2);
      cout << "\nThe sum is: " << sum << endl;
      return 0;
}

int AddTwoInts(int num1, int num2)
{
    return  num1 + num2;
}
```

Prototype

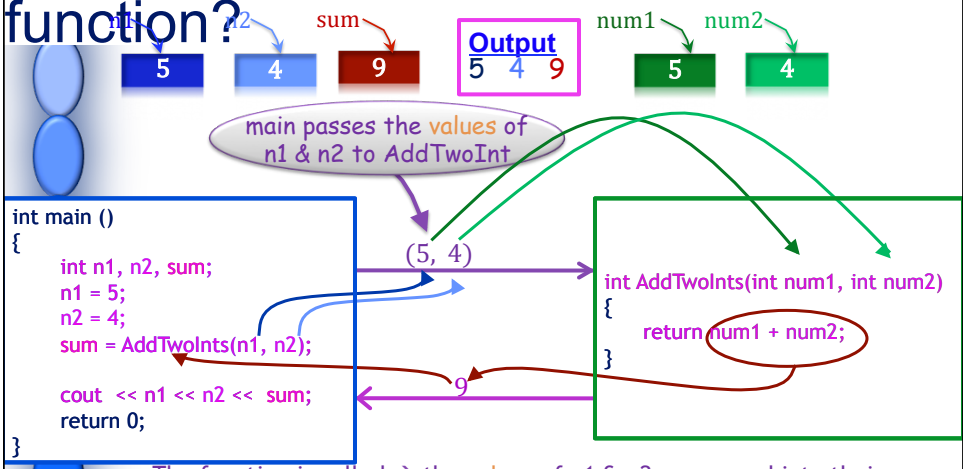
Calling Function

function call

Function Definition

10

How do the values get into the function?



- The function is called → the values of n1 & n2 are passed into their respective positions (num1 & num2)
- In this case the n1 is the first argument will be sent to the first parameter in the function → num1
- If we had put n2 first then the value in n2 would be sent to num1
- The value of num1 + num2 is returned → the function call gets the return value

11

Functions – Quick Review

To use a function you must have:

How do you declare a function (i.e. how do you write a prototype)

Where do you declare a prototype?

Where do you define a function?

Example Prototype and Function

```
int ValidateInput(int lowerBound, int upperBound);

int ValidateInput(int lowerBound, int upperBound)
{
   int    inputValue;
   bool  invalidInput;

   invalidInput = false;

   do
   {
        cout  << "Enter Integer Input: ";
        cin    >> inputValue;

        if (inputValue < lowerBound || inputValue > upperBound
             cout << "ERROR: Value is out of range - please try again
        else
             invalidInput = true;
    } while(invalidInput);

    return inputValue;
}
```

Prototype

Function Definition

What is wrong with this?

Function Calls

The function call goes in the body of a function

- Can be called in the main function (between the {})
- Can be called by another function
- Can call itself (this is called recursion and will be covered in 1C)

When a function is called

- The code in the function definition is executed
- Then function ends when the return statement is executed
- Execution of the calling function is resumed

Example Function

Prototype

```
...
int ValidateInput(int lowerBound, int upperBound);

int main()
{
...
    // get a value between 1 & 10
    firstInput     = ValidateInput(1,10);
    // get a value between 5 & 50
    secondInput = ValidateInput(5,50);
    // get a value between 2 & 100
    thirdInput     = ValidateInput(2,100);

}
```

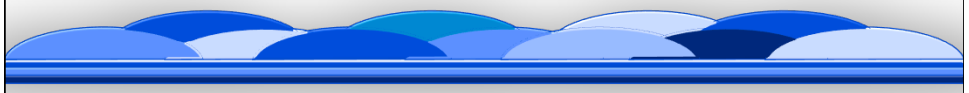
Function Calls

Function Definition

```
int ValidateInput(int lowerBound, int upperBound
{
    int    inputValue;
    bool  invalidInput;
    invalidInput = true;
    do
    {
        cout  << "Enter Integer Input: ";
        cin    >> inputValue;
        if (inputValue < lowerBound
                ||   inputValue > upperBound)
            cout << "ERROR - try again";
        else
            invalidInput = false;
    } while(invalidInput);
    return inputValue;
}
```

For now we will put it under the block of code for the main function after the last }

Variable Scope & Lifetime



Variable Scope & Lifetime

Variable Scope

- Where a variable can be accessed

Variable Lifetime

- How long it lasts

Scope & Lifetime are defined based on whether the variable is locally defined or globally defined

Where a variable is declared determines its scope and lifetime

Local Variables

- Declared within a block or function
  - A Block is the curly brackets
- The scope and lifetime a within those brackets
  - not accessible outside of that block or function ← Visible only to that function
  - ie the variable exists in memory only as long as that function is executing
  - Memory space is allocated when the function is called
  - Memory space is deallocated when the function ends (returns)
  - ie all local variables are destroyed when you exit a function
- Parameters are treated as local variables
- Variables declared within a function are declared within the {}
  - Just like we have been doing in main

Functions can't see variables declared in other functions including the main function
➔ This is why we have to pass in values using parameters

Local Variables Example 2

```
for (int count = 1; count <= 10; count = count + 1)
{
...
}
```

What is the life and scope of this variable?

It depends on the compiler

- Some consider it local to the for loop
- Others consider it local to the function

Make sure your variable names are unique within your functions and blocks of code to avoid problems

Global Variables

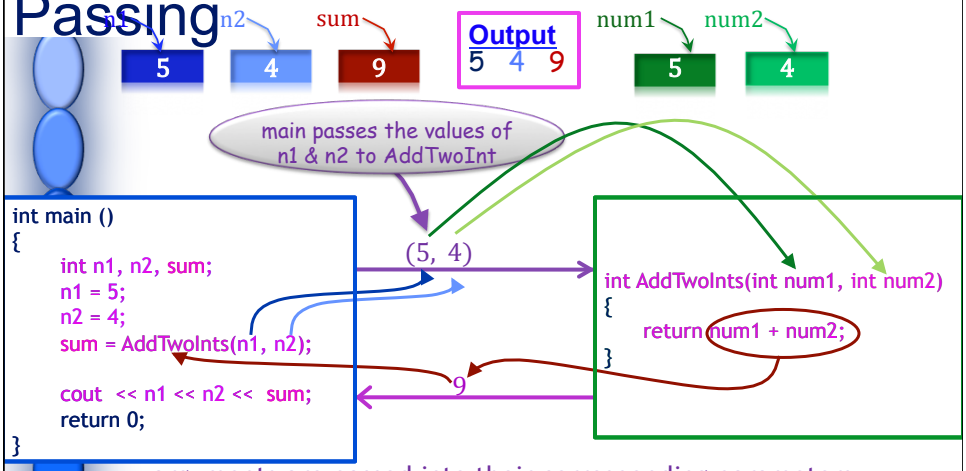
- Declared outside the block
- scope and lifetime are from the point of declaration until the end of the source file
- These are available to any function in the program including main()

- If a local variable has the same name as a global variable the global variable is ignored
- Global variables grew out of C and are rarely used in C++.
- They are considered bad practice
- They are dangerous because they share data
- Changes can occur in one function that are invisible to another function making bugs difficult to detect

Global variables are bad practice and problematic.
Global constants are fine!

Passing Parameters

Calling Functions – Parameter Passing



- arguments are passed into their corresponding parameters
- In this case the n1 is first on the calling list so it's value will be sent to the first parameter in the function → num1
- If we had put n2 first then the value in n2 would be sent to num1
- The value of num1+ num2 is returned → the function call gets the return value

22

Parameters

There are two types of parameters

- Value Parameters
  - A formal parameter that receives a copy of the contents of the corresponding argument (actual parameter).
- Reference Parameter
  - A formal parameter that receives the address (location in memory) of the corresponding argument (actual parameter).

Passing by Value

What we have been demonstrating so far is passing by value (i.e. using value parameters)

- A duplicate copy of each variable is created when the function is called
- The values of the parameters being passed from the calling function are copied into the parameters of the function
- If the called function changes these parameters it does not effect the calling functions values

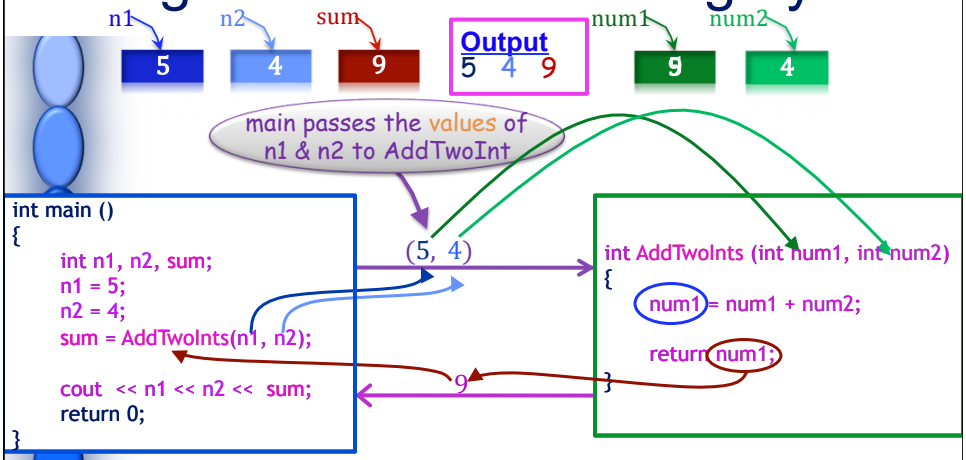
Advantage

- No accidental modifications of the arguments in the calling function

Disadvantage

- Passing large variables takes a lot of overhead
- The value of the passed variable has to be copied & initialized
- For small variables this is good
- Large variables time & space penalties become a problem

Calling Functions – Passing by Value



- The function is called → the values of n1 & n2 are passed into their respective positions (num1 & num2)
- The value of num1 is changed → it does not effect n1
- The value of num1 is returned → the function call gets the return value

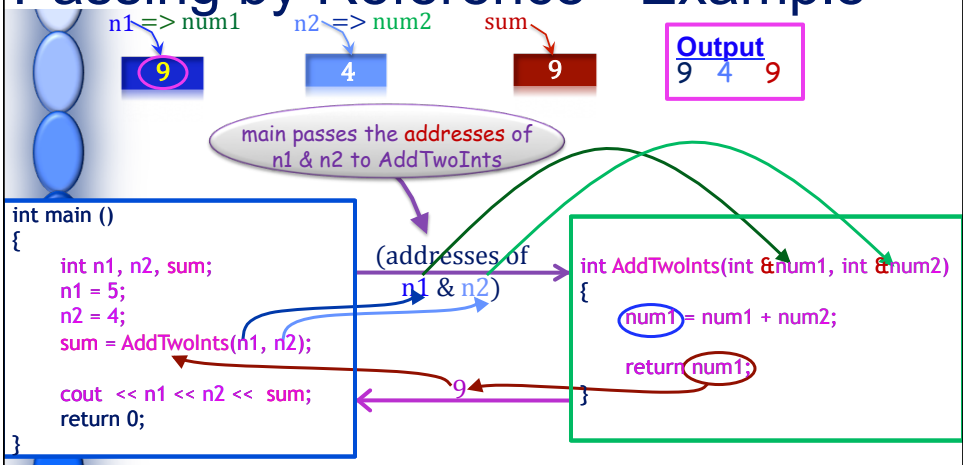
Passing by Reference

- A reference is an alias
  - Basically a different name is used for the same variable
- When we use Reference parameters the addresses of the variables passed from the calling function to are called function
  - The actual memory location is being passed
  - This means that the value in these locations can be changed
- Because you are passing a reference you must pass a variable
  - You can't pass a literal or an expression by reference

Syntax
returnType functionName(*parameterType* &*parameterName*)

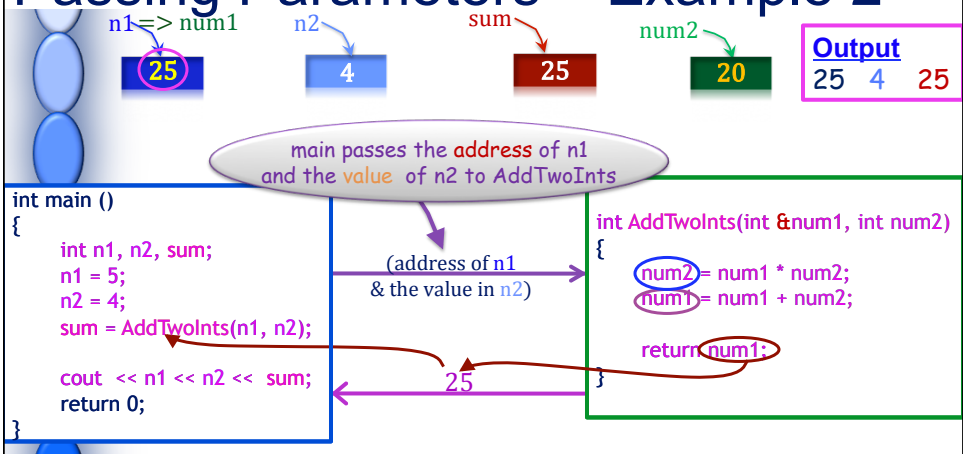
Example: int AddTwoInts(int &num1, int &num2)

Passing by Reference - Example



- The function is called →The addresses of the parameters are passed into their corresponding positions
- The value of num1 gets changed → so does the value of n1
  - This is called a side-effect
- The value of num1 is returned → the function call gets the return value

Passing Parameters – Example 2



- The function is called→The address of n1 is sent to num1 and the value of n2 is put into num2
- Changes num2, but not n2
- The value of num1 gets changed → so does the value of n1
- The value num1 is returned to main→ the value of the function is 25

28

Side-Effects

- when a parameter passed by reference is changed in the function that is called
- This can lead to trouble
  - It becomes to hard to determine how values are changed
- Can't happen with pass by value
  - All variables passed by value are treated like constants by the called function

Solution ➔ Pass by constant reference

Constant Reference

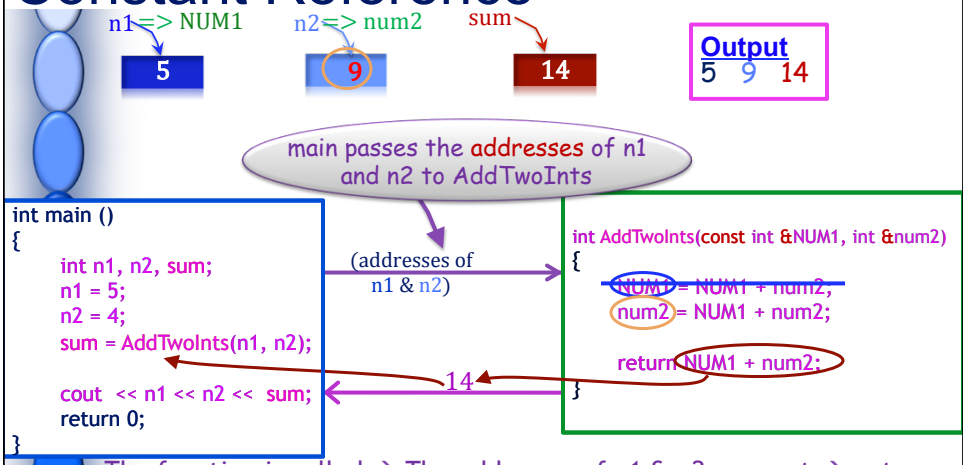
- A reference that does not allow the variable being referenced to be changed
- The called function can use the value but can't change it

Syntax
returnType functionName(const *parameterType* &*PARAM_NAME*)

Example:

int AddTwoInts(const int &NUM1, const int &NUM2)

Constant Reference



- The function is called → The addresses of n1 & n2 are sent → note that num1 is a constant now
- Can't do this! → num1 is a constant now
- The value of num2 gets changed → so does the value of n2
- The value of the expression num1 + num2 is returned to main→ the value of the function is 14

Passing by Reference Advantages

- A function can change the value of the argument, which is sometimes useful
- Because a copy of the argument is not made, it is fast, even when used with large arguments
- We can pass by const reference to avoid unintentional changes.
- We can return multiple values from a function.

.

Passing by Reference Disadvantages

- Because a reference can not be made to a literal or an expression, reference arguments must be normal variables.
- It can be hard to tell whether a parameter passed by reference is meant to be input, output, or both.
- It's impossible to tell from the function call that the argument may change.
- arguments passed by value and passed by reference look the same (from the calling functions perspective)
- We can only tell whether an argument is passed by value or reference by looking at the function declaration.

… This can lead to situations where the programmer does not realize a function will change the value of the argument.

Which should you use?

- PASS BY REFERENCE WHEN
  - You need to pass large variables
    - No overhead
    - If you absolutely have to change more than 1 value in a function
  - When you need to Return Multiple Values
    - All values passed by reference can be returned
    - If you pass by reference and are not planning on returning a value then pass by constant reference
- PASS BY VALUE WHEN
  - You need to pass simple variables
  - When you don't want the value in the calling function to be changed
    - No side-effects (accidental modification of the variables)
  - When you need to pass a literal, constant, or expression
  - Anytime except when you have large values or need to return multiple values

Some things to Remember

- Value parameters can be used in the called function as with any declared variable
  - Changes to it will not effect the value of the variable used in the parameter from of calling function.
- Reference parameters are modified by the function
  - can appear either on the left side of an assignment statement or in a *cin statement*.
  - *Unless they are constant reference parameters*

Parameters & Arguments

Parameters => formal parameter => formal argument

the identifier used to represent the value that is passed by the calling function

Arguments => actual parameter => actual argument

the actual value that is passed into the function by the calling

- Parameters & arguments
  - matched according to their relative positions.
- Arguments
  - appear in the function call and do not include their data type.
- Parameters
  - appear in the function heading and include their datatype.
- When the parameter is a value parameter
  - the argument may be a variable, named or literal constant, or expression

Some notes on Functions

- You can't define a function within a function → no nesting functions
- There is no limit to the number or types of statements that can be used in a function

HOWEVER → Keep them small

- REMEMBER: Each function should carry out a single easily understood task
- Should be small enough to fit on a screen
- Smaller functions are easier to understand, code, and debug
- If your function is large → look for places you can divide it into smaller functions (divide and conquer)

- Function Arguments don't all have to be the same type
  - Example:

    int ConvertTemp(char fromTemp, float temp)

Defaulting Arguments

- You can specify a default for parameter
- For example if you want ConvertTemp to default to converting F to C.

```
int ConvertTemp(float temp, char fromTemp = 'F');
```

Syntax
return_type functionName(*type parameter = default_value*);

NOTE: you **MUST** put the parameters with default values **following** all the parameters that don't have default values

➔ You can only put the default in the prototype or the definition – **not both**

→ it is best practice to put it with the prototype

Example

```
int ConvertTemp(float temp, char fromTemp = 'F');
```
Prototype

```
int ConvertTemp(float temp, char fromTemp)
{
    if (toupper(fromTemp) == 'F')
    {
        return ((temp-32)*(5/9));
    }
    elseif (toupper(fromTemp) == 'C')
    {
        return (temp * (9/5) + 32);
    }
    else
    {
        cout<< "some error message";
        return 0;
    }
```
Function Definition

Default!

We can call this function from another function like this:

```
newTemp=ConvertTemp(55,'F');
```
temp will get the value 55
fromTemp will get the value 'F'
newTemp = 12.8

```
newTemp=ConvertTemp(60,'C');
```
temp will get the value 60
fromTemp will get the value 'C'
newTemp = 140.0

```
newTemp=ConvertTemp(100);
```
temp will get the value 100
fromTemp will get the value 'F'
newTemp = 37.8

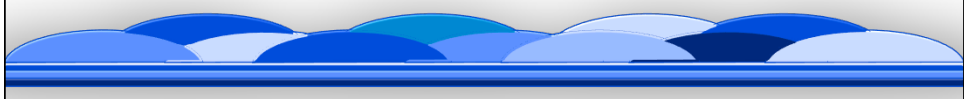
Multiple Return Statements

- You can have more than 1 return statement i a function
- For example if you want to return a different value based on some condition
  - You can use an if statement

See previous Example

HOWEVER, it is not a best practice avoid them

Void Functions



Chapter 7 in the shrinkwrap

Void Functions

What are they?

- Functions that don't have an explicitly stated return value
  - or a return statement

They are good for functions that…

- Don't return anything → such as a series of input/output statements
- have more than 1 return value ← make sure you don't make your functions too complicated!
- OTHERWISE USE A VALUE RETURNING FUNCTION

Naming Void functions

- Choose a name that will sound like a command or an instruction
  - They will be called by themselves → not as an assignment statement or a cout(they don't return anything)
- Example void function calls

```
PrintHeader();
FindAndPrintSmallest();
```


Declaring Void functions

- Just like with regular functions you need to
  - Have a prototype
  - Define function below int main()
  - Then you can call the function

Example Definition

Use "void" for the datatype

Void function with no parameters
This is optional → could just have
void PrintHeader()

```
void PrintHeader(void)
{
    cout << "*************************************************************";
    cout << "*  PROGRAMMED BY :  Michele  Rousseau          *";
    cout << "*  STUDENT ID    :  7502312                    *";
    cout << "*  CLASS         :  CS1B - MW 6p-7:30p         *";
    cout << "*  LAB #3        :  Intro to Functions         *";
    cout << "*************************************************************";
}
```

Void functions with Parameters

```
void PrintHeader(string asName, char asType, int asNum)
{
    cout << left;
    cout << "*************************************************\n";
    cout << "*  PROGRAMMED BY : Michele Rousseau";
    cout << "\n*  " << setw(14) << "STUDENT ID"  << ": 7502312";
    cout << "\n*  " << setw(14) << "CLASS" << ": CS1B --> MW - 6p-7:30p";
    cout << "\n*  " ;

    // PROC - This will output "LAB #" or "ASSIGNMENT #"  based on the
    //           asType and adjust the setw accordingly
    if (toupper(asType) == 'L')
    {
       cout << "LAB #" << setw(9);
    }
    else
    {
       cout << "ASSIGNMENT #" << setw(2);
    }
    cout << asNum << ": " << asName;
    cout << "\n*************************************************\n\n";
    cout << right;
```

44

Documenting Functions

Read through this on your own!

Some things to remember about Comments

How to add comments

- // ← for a few lines or after a line of code
  - You can select a group of code and ctrl - // to comment out several lines at a time
  - If you ctrl- // on a comment it will uncomment the line
  - This can be useful in debugging – by isolating parts of your code
- Block comments

```
/*
    <anything between these will be commented>
*/
```

46

Commenting your code

For all programs in this class

- Before EVERY FUNCTION
  - Use comments to describe your program
- Data Table
  - The declaration section must contain a data table
  - The data table
    - states the use of the variable or named constant and
    - how its value is obtained/used.
- Other comments should be used throughout your code to
  - Describe what each section is doing
    - (think in terms of input, processing, & output)
  - Complicated parts of the code → be descriptive!
- Try to line to comments up as best as you can!

47

How to doc your code

First thing in your code should be your name and assignment info

```
/*************************************************************
* AUTHOR   :
* LAB #0   : Template
* CLASS    :
* SECTION  :
* DUE DATE :
*************************************************************/
```


Next…

- Preprocessor Directives then doc for the main program

```
#include <iostream>
#include<iomanip>
#include <string>
using namespace std;
/*************************************************************
*
* ADD & MULTIPLY TWO INTS
*-------------------------------------------------------------
* This program does whatever this program does
*  save this template and fill in the appropriate info for
*  your program
*-------------------------------------------------------------
* INPUTS:
*   int1: First integer to be summed received as input
*   int2: Second integer to be summed received as input
*
* OUTPUTS:
*   sum    : the sum of the two ages
*   product: The product of the two integers
*************************************************************/
```

Next

- Prototypes

```
/*************************************************************
 * PrintHeader
 *   This function receives receives an assignment name, type
 *   and number then outputs the appropriate header
 *   - returns  nothing → This will output the class heading.
 *************************************************************/
void PrintHeader(string asName, // IN - assignment Name
                 char   asType, // IN - assignment type
                                //      (LAB or ASSIGNMENT)
                 int    asNum); // IN - assignment number
```


Next → int main

```
int main ()
{
       // declare your variables here - include your data table

       // PrintHeader - Will output a header for this assignment
       PrintHeader("Functions", 'A', 14);

       // INPUT:  A description of what is being input.

       // PROCESSING:  Detail what is being processed.

       // OUTPUT:  Details of what is being output.
}
```

FUNCTIONS should go in another file and should be documented

```
/*********************************************************
 *
 * FUNCTION PrintHeader
 *_______________________________________________________
 * This function receives an assignment name, type
 *   and number then outputs the appropriate header -
 *   returns nothing.
 *_______________________________________________________
 * PRE-CONDITIONS
 *      asName: Assignment Name has to be previously defined
 *      asType: Assignment Type has to be previously defined
 *      asNum : Assignment Number has to be previously defined
 *
 * POST-CONDITIONS
 *     This function will output the class heading.
 *     <Post-conditions are the changed outputs either
 *      passed by value or by reference OR anything affected
 *      by the function>
 *********************************************************/
 void PrintHeader(string asName, // IN - Assignment Name
                  char   asType, // IN - assignment type
                                 //    - (LAB or ASSIGNMENT)
                  int    asNum)  // IN - assignment number
{
```


Function Definition

```
void PrintHeader(string asName, // IN - assignment Name
                 char   asType, // IN - assignment type
                                //    - (LAB or ASSIGNMENT)
                 int    asNum   // IN - assignment number
{
  cout << left;
  cout << "*************************************************\n";
  cout << "*  PROGRAMMED BY : Michele Rousseau\n";
  cout << "*  " << setw(14) << "STUDENT ID" << ": 7502312\n";
  cout << "*  " << setw(14) << "CLASS"      << ": CS1B --> MW - 6p-7:30p\n";
  cout << "*  " ;

  // PROC - This will output "LAB #" or "ASSIGNMENT #"  based on the
  //        asType and adjust the setw accordingly
  if (toupper(asType) == 'L')
  {
      cout << "LAB #"        << setw(9);
  }
  else
  {
      cout << "ASSIGNMENT #" << setw(2);
  }
  cout << asNum << ": " << asName << endl;
  cout << "*************************************************\n\n";
  cout << right;
}
```

Some notes on Functions

- Keep them simple and try to make them generic
  → that way you can reuse them

Example:

```
// this function searches a string array for one string
// returns the appropriate index #
 int SearchStringArray(const string STR_AR[] ,
                              const int AR_SIZE, string searchStr)
```

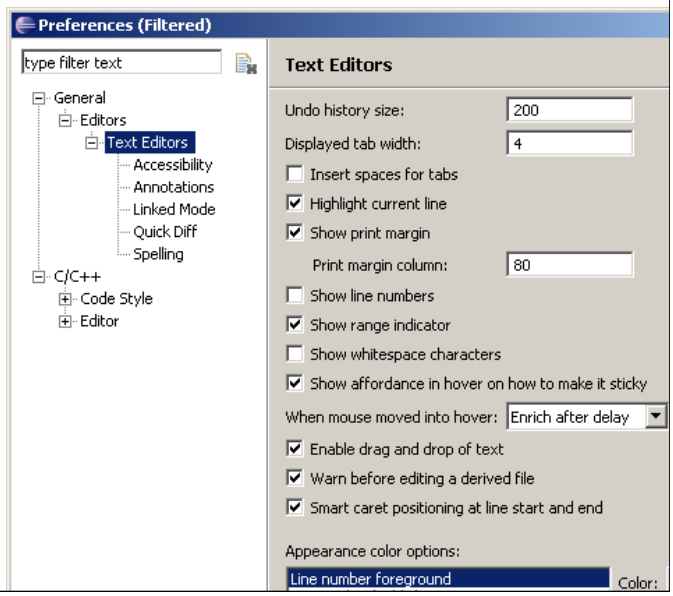
Instead of

```
 int SearchName(const string NAME_AR[],
                          const int AR_SIZE, string searchName)
```

- Keep them Simple!
  - each function should do 1 thing
  - In otherwords → if you need to search for something your function should just search for that something not deal with I/O specific to your project

Right click
on side
To get
this menu

Show Line #s
Change print
column to 75



Good Practices

- Keep related functions in the same files
  - e.g. I/O
- Separating your files makes them easier to manage
  - your main.cpp can get long and difficult to find things

Some notes on Functions

- Keep them simple and try to make them generic
  → that way you can reuse them

Example:

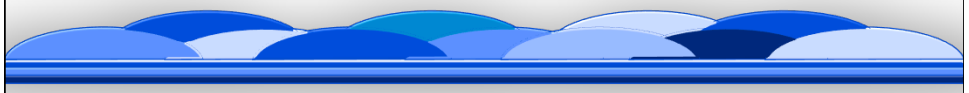
```
// this function searches a string array for one string
// returns the appropriate index #
 int SearchStringArray(const string STR_AR[] ,
                                const int AR_SIZE, string searchStr)
```

Instead of

```
 int SearchName(const string NAME_AR[],
                        const int AR_SIZE, string searchName)
```

- Keep them Simple!
  - each function should do 1 thing
  - In otherwords → if you need to search for something your function should just search for that something not deal with I/O specific to your project

User Defined Header Files



Header files

- So far we've worked with several header files
  - files that follow #include
  - <iostream>
  - <iomanip>
  - <fstream>
  - <string>
- We include these to be able to access certain predefined functions, classes, or variables in C++

Creating our own

- It is often convenient to create your own header files
- To do this we need to
  - create the file
  - Include it in our source code
- Creating the file
  - create a new file *filename.h*
    - end it with .h
- Including the file
  - #include "*filename*.h"

Header File

```
// these two lines and the last one ensure that you
//   don't accidentally make the same definitions twice - it is a good
//   practice to include them
// this example assumes your header file name is MyHeader.h

#ifndef MYHEADER_H_
#define MYHEADER_H_

<your preprocessor directives>
<global constants>
<your typedefs and enumerated types>
<your function prototypes>
#endif
```

NOTE:
eclipse will automatically include the lines of code that are in black
→ you **MUST** insert your preprocessor directives, tyedefs,
and enumerated types as specified

Example: Creating a header file

```
// this file is called myheader.h
#ifndef MYHEADER_H_
#define MYHEADER_H_

// preprocessor directives go here
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

// Global Constants
// User Defined Types go here (more on this later)

// Prototypes go here
int SearchStArray(string stAr[], string searchStr);

#endif     /* MYHEADER_H_ */
```

- To include this file

```
#include "MyHeader.h"
```


Some points to mention

- you must use quotes in your header file
  - "MyHeader.h" → NOT <MyHeader.h>
- the file must be located in your project folder
  - otherwise C++ can't find it

Common Errors

- Make sure your files are all in the same folder
- Make sure that you have your preprocessor directives BEFORE your prototypes
  - ORDER MATTERS
    - 1 – preprocessor directives
      - **# includes & namespace**
    - 2 – global constants
    - 3 – typedefs and enumerated types
    - 4 – prototypes
- You can't have code in the header file
- You can have code in a separate file
- You can only have 1 int main()

Setting the seed for a random value

- To get a random value we need a seed
  - The seed value can be sets the starting value for the random values

Syntax
srand(seed);

  - We will use time as a seed since the time will provide a unique runtime value

Syntax
time(NULL)

This goes in main()

  - So to set the seed based off of the time we write

```
srand(time(NULL));
```

  - The seed should only be set 1X
    - otherwise it will start the set of random values over again
    - meaning it will produce the same value every time

Getting a Random Value

- Finally – when you want a random value

Syntax
rand()

- This will return a random integer from 0 to RAND_MAX

```
myRandomValue = rand();
```

- Use the mod function to get values within a specific range
  rand() % 25 – will give you values from 0 - 24
- For example if I want a random number from 1 to 25

```
myRandomValue = rand() % 25 + 1;
```

You will need to include the following two header files

```
#include <stdlib.h>        /* for srand, rand */
#include <time.h>          /* for time */
```


Pair Programming

- One component of the XP (eXtreme Programming) software process model
- TWO programmers - ONE computer
  - Driver – types the code
    - Comes up with the algorithms and etc…
  - Navigator (or Observer)
    - Looks for ways to improve the code
  - Roles are switched frequently
- In this class…
  - Switch roles every 20 minutes – or after each function
  - Must be co-located – can't be done remotely
  - 3 senarios
    →BOTH students are responsible for understanding the code
  - Must pick a different partner for each lab for credit