Lecture 3

Introduction to Programming:

Cin, Functions, Typecast

Chapters 2, 3

Console Input Example

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

int main()
{
        int number;
        cout << “Enter a number:";
        cin >> number;
        cout << “Your number is “ << number;
        return 0;
}
```

IO:

```
Enter a number: 8
Your number is 8
```

Console Input cont.

- Cin allows for the user to input data directly into the program
  - The program's output is no longer predetermined, rather it is dependent on the data inputted by the use
- Cin is a class that resides in the *iostream* library
- Cin allows the user to enter data type:
  - Integers
  - Floats/Doubles
  - Chars
  - Strings
- Cin uses the extraction operator (instream) between each element

Console Input Statement

```
double number;
cin >> number;
```

- All operands of the extraction operator must be variables
  - This is intuitive since all data from the user must be stored
  - endl cannot be used with cin since it is not a variable
- The data inputted by the user shall be interpreted as the data type of the variable of the extraction operand
  - i.e. in this case if the user types "hello" as the input, the string "hello" will be parsed into a double resulting in a program error since number is a double
- When typing in the terminal, any whitespace dictates the end of a data input

Multiple Cins

```
double x, y;
cout << “Enter x and y:”
cin >> x >> y;
```

- The first number entered will be stored in x
- The second number entered will be stored in y
- When entering the numbers into terminal they must be separated by a whitespace character (space, tab, enter)
- DO NOT use a comma when enter the numbers into terminal, the data type will be a string that will cause an error when being parsed as a double

Calculating Area of a Circle

```
float radius, area;
cout << “Enter the radius of the circle: “ << endl;
cin >> radius;
area = radius * radius * 3.14;
cout << “The area is: “ << area << endl;
```

- The radius of the circle is fetched by the user making the output of the program different depending on the user input
- Notice radius * radius instead of radius^2
  - the operator ^ is not an exponent operator and therefore should not be used as one

Cin/Cout Pro Tip

```
cin >> number
cout << “Hello” << endl;
```

- Many students get the extraction/insertion operators switched
- The operators point to the direction to which the data is going
  - With cout, the arrows point towards the cout since the data is being sent to the console
  - With cin, the arrows point to the variables since the data is being sent from the console to the variable

Example to Consider

```
float amountPaid = 10, amountOwed = 2.34;
int change;
change = amountPaid – amountOwed;
cout << “Change is: “ << change << endl;
```

- In this case the variable change cannot accept fractions of a dollar making it in accurate
  - Thus there is a loss in precision
- amountPaid – amountOwed is evaluated to a float, but is changed to an integer
  - This is referred to as type casting

Explicit Typecasting

```
float number;
number = (int) (3.0 / 2.0);
```

- number will have 1.0 stored in it
- The right hand side can be typecast by placing the data type which you wish to convert to inside a pair of parentheses
- Therefore 3.0 / 2.0 -> 1.5 is first computed, then converted to 1 which is finally stored as 1.0

Safe Typecasting

- Any primitive data type can be typecast as another data type
  - A safe typecast is a typecast where no loss of data can take place
- Examples
  - float to double
    - safe, double is a float with more precision no data will be lost
  - double to float
    - unsafe, possibly could result in loss of precision
  - int to float
    - unsafe, floats precision is ~7 digits, therefore a loss of precision can occur
  - char to int
    - safe, char is a small int, therefore a larger int will not result in a loss of data

Final Note on Typecasting

- While typecasting should be avoided as much as possible it may not be always avoided
  - Unsafe typecasting is allowable in if the loss of data is not a concern
  - Example: typecasting an int to a float, when the int is known to be small

More Data Types

Data Type	Explanation
short	variable is smaller or equal to the default size
long	variable is greater than or equal to the default size
long long	variable is greater than or equal to a long
unsigned	the variable does not contain a sign, always positive
signed	variable contains a sign, positive or negative

Example

unsigned short int x = 5;

- Variable x cannot be negative
- On a 32 bit system a typical int is 32 bits
  - by declaring a short it is likely a short int will be half that size
  - short int ~16 bits
- In most Windows/Linux Apps memory is ubiquitous and not typically a concern
- In embedded applications (microcontrollers) memory is at a premium, therefore assigning shorts instead of ints can allow for a program to be implemented on cheaper hardware

Constants

Example:

```
const float pi = 3.141593;
```

- const (constant) is a piece of data which cannot be altered during runtime
- Benefits
  - Variable cannot be accidentally changed at some point in the program
  - Programmer does not have to repeatedly type 3.141593 every time they wish to use pi saving time.

Calculating Area Revisited

```
float radius, area;
const float pi = 3.14;
cout << “Enter the radius of the circle: “ << endl;
cin >> radius;
area = radius * radius * pi;
cout << “The area is: “ << area << endl;
```

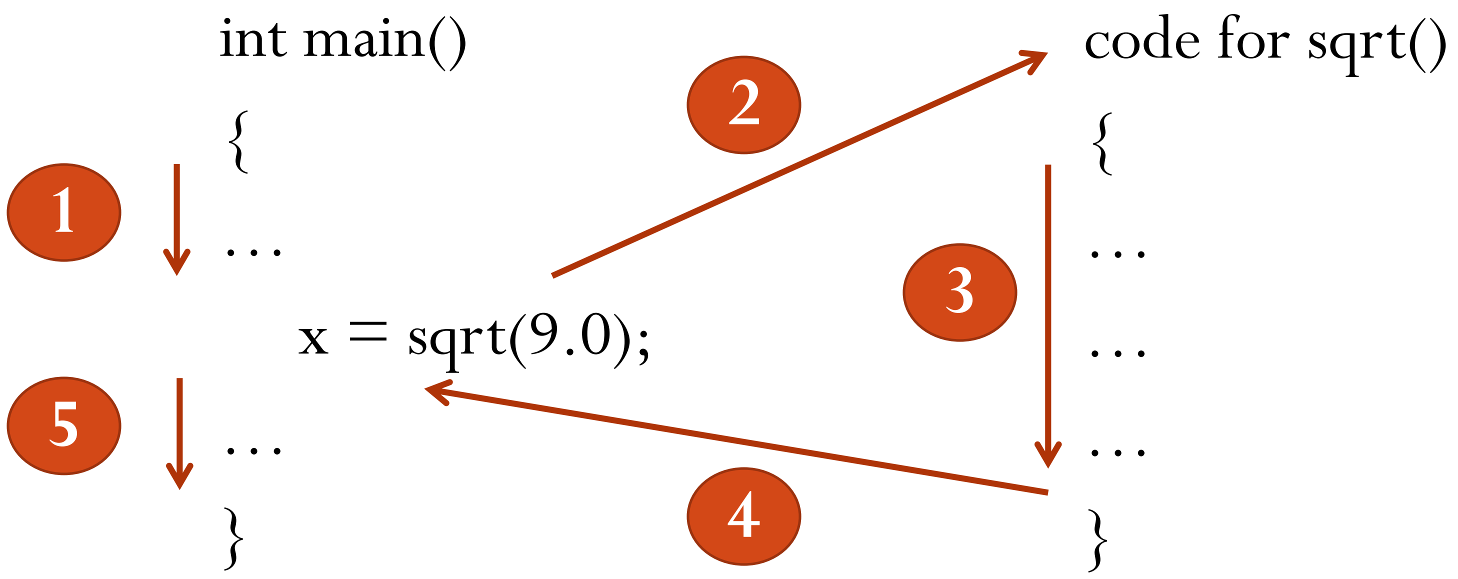
Integer Literals

- Numbers can be written in a variety of ways
- Decimal
  - int x = 10;
- Hexidecimal
  - int x = 0x3a;
  - 0x dictates the number to be hex
- Octal
  - int x = 0123;
  - leading 0 dictates the number to be octal
- Float
  - float x = 2.52e+8;
  - inclusion of e dictates floating point number

Functions

- C++ Library has many built in functions
  - pow
  - sqrt
  - cos, sin
- Functions provide two primary advantages
  - Code Reuse
    - A function can be given a different inputs which in return produce different outputs
  - Organization
    - A large piece of code can be hidden away in a function. When the code is called ("used") the name of the function is used as one line of code.

Functions Cont.



```
int main()              code for sqrt()
{                       {
…                       …
   x = sqrt(9.0);       …
…                       …
}                       }
```

1. Code executes sequentially until function is encountered
2. Execution jumps to sqrt code with the input of 9.0 sent to the function
3. sqrt code is executed sequentially
4. Upon exiting the function, the value of the calculation is returned and stored into x
5. Execution of the Main resumes on the line after the function

Function sqrt example.

```
int main()
{

        // pythagorean theorem
        // c = (a^2 + b^2)^(1/2)
        float a, b, c, midCalc;
        cout << "Enter a, b, c:";
        cin >> a >> b;
        midCalc = b * b + a * a;
        c = sqrt(midCalc);
        cout << "C is " << c << endl;

        return 0;
}
```

- The code to the left calculates the Pythagorean theorem
- The square root of midCalc is calculated and returned
- It is then stored into c
- The act of using the function, sqrt(midCalc), is known as a *function call*

User Defined Functions

- As a programmer you can define (create) your own functions
- Below is a simple example of a user defined function

```
void printHello( )
{
        cout << “Hello Functions”  << endl;
}
```

- The code above merely defines what a function does, in this case print *Hello Functions*, it does not specify when to run the function

Function header

Header

```
void printHello( )
{
        cout << "Hello Functions"  << endl;
}
```

Body

- Functions are composed of two parts
  - Header: defines the returned datatype, the input variables, and the function name
  - Body: defines the code to be executed when the function is called, and the data being returned/outputted from the function

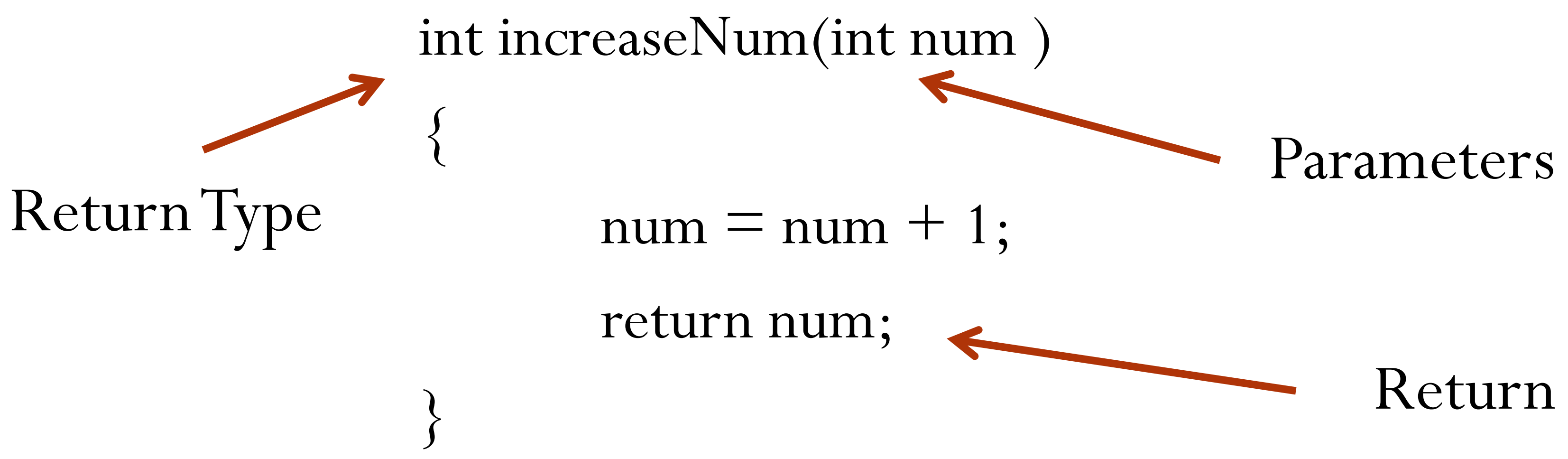
Using User Defined Function

```
void printHello();  // Prototype
int main()
{
        printHello();
        return 0;
}

void printHello( )
{
        cout << "Hello Functions"  << endl;
}
```

- To use the function we call the function just as a standard library function
- For the function to be recognized we prototype it at the top
- For now, prototype by copying the function header to the top of the source file and place a semicolon at the end of the statement

User Function with a Return



```
int increaseNum(int num )
{
        num = num + 1;
        return num;
}
```

- Parameters: define the inputs to the function (the data being passed to the function)
- Return Type: datatype of the data being returned, this case num
  - return type of void means no data is being returned
- Return: defines the exit of the function, and outputs the argument variable
  - **You can only return one variable**

Using User Function with a Return

```
int increaseNumber();  // Prototype
int main()
{
        int x  = increaseNumber(2);
        cout << x << endl;
        return 0;
}

int increaseNumber(int num )
{
        num = num + 1;
        return num;
}
```

- increaseNumber function has a return therefore the output, num can be stored into x
- the cout will result in 3 being printed

Other Basics of Functions

- int main() is a function, and obeys all rules of a function
- You can declare and use variables inside a function
  - These variables only exist inside the body of the function, i.e. you cannot access a variable declared in your function in the main function or vice versa
- Variables passed to a function are passed by value (as copies)
  - Any modification made to a parameter inside the function will not be reflected in the calling function
  - Example:

```
int main()
{
        int x = 4;
        increaseNumber(x);
        cout << x << endl; // This will print 4, not 5
        return 0;
}
```

Other Basics of Functions

- Multiple variables can be passed as parameters using the comma operator

  int triArea(int base, int height)

  - The order of the variables passed in the function call must match the order of the defined parameters
- **Only one variable can be returned!!!**
  - Methods to get around this will be discussed in future lectures

auto Declaration

- Datatypes of variable declarations can be deduced by using the keyword auto

auto x = 0.0;

- above example would be declared as a double
- Typically used from complex data types
- GPP: explicitly typecast your data
  - Below is an example of how one can accidently auto cast to the wrong data type

  auto x = 0; // typecasted to int, though you may have wanted float