PROGRAMMING BASICS

OVERVIEW

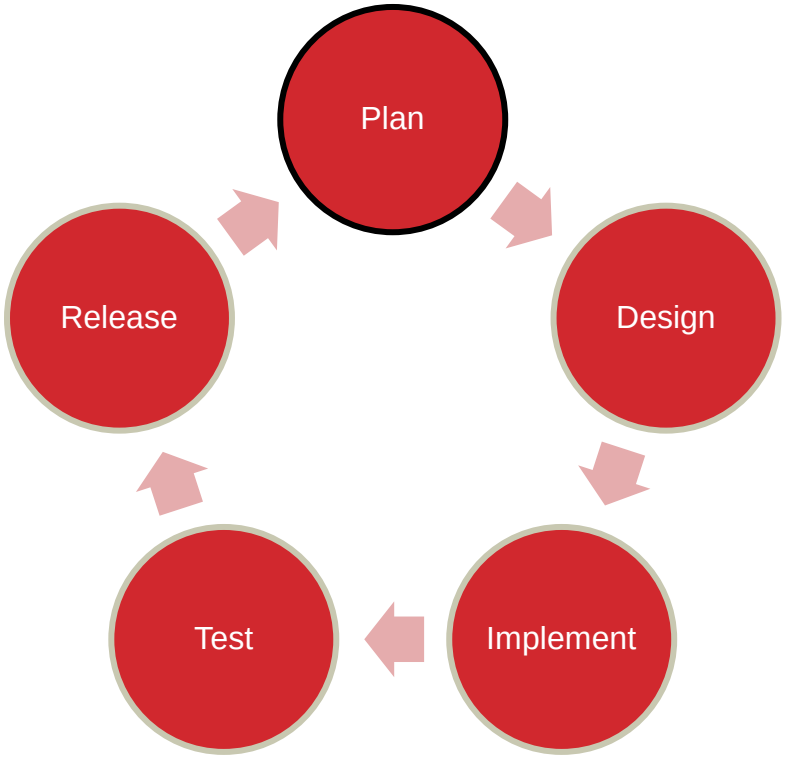
OVERVIEW

- **What is computer programming?**
  - The objective of programming is to give the computer detailed instructions to solve a desired problem
  - Computers have to read and process these instructions so they have to be written clearly and unambiguously
  - Hundreds of programming languages have been invented for this purpose over last 50 years
  - This class will use the programming language C++ because it is very powerful and widely used in industry

OVERVIEW

- **How do we write programs?**
  - Tools and techniques for writing programs have evolved over the last 50 years, and continue to evolve today
  - The goal is to convert abstract goals (what we want the program to do) into clear and unambiguous instructions for the computer (in our case C++ code)
  - The classic software development cycle we will be using has five stages: plan, design, implement, test, and release

OVERVIEW

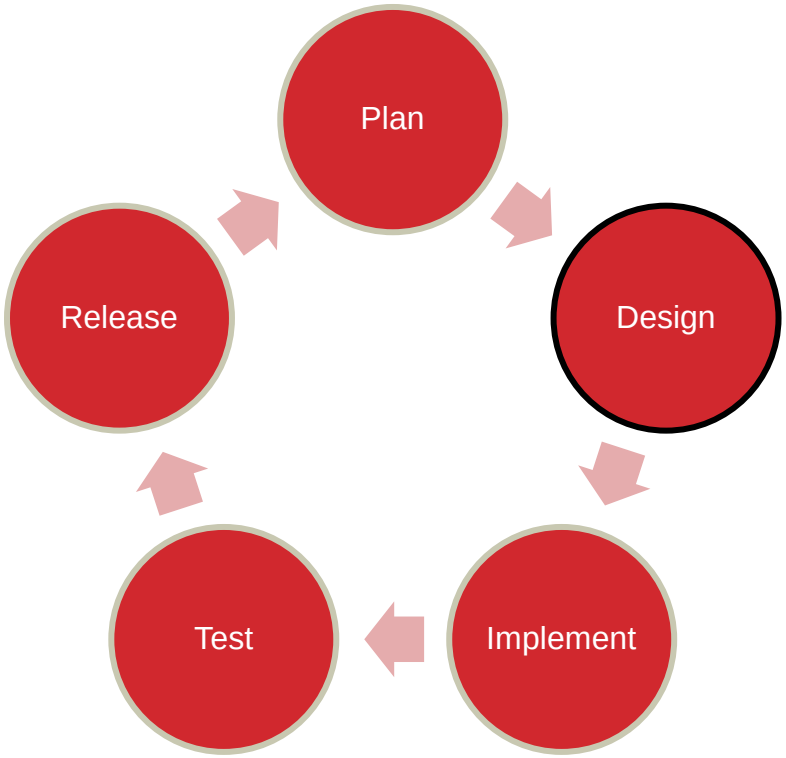


The classic software development cycle

Plan:

- Decide what problem we are trying to solve
- What are program inputs?
- What should the program output or do?

OVERVIEW

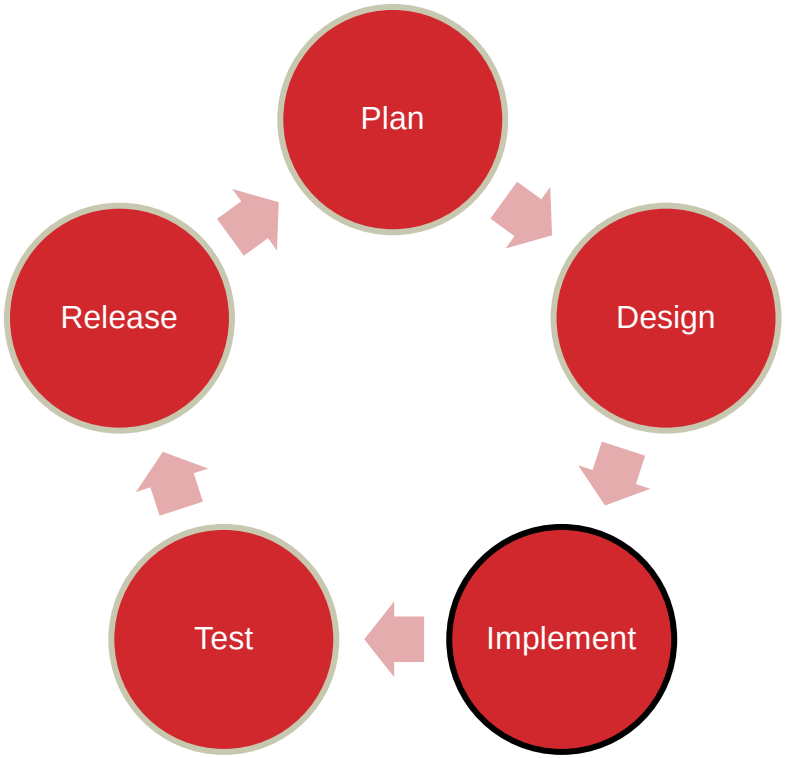


The classic software development cycle

Design:

- Break the problem into smaller steps we know how to solve
- Describe how these steps should be combined to solve the problem

OVERVIEW

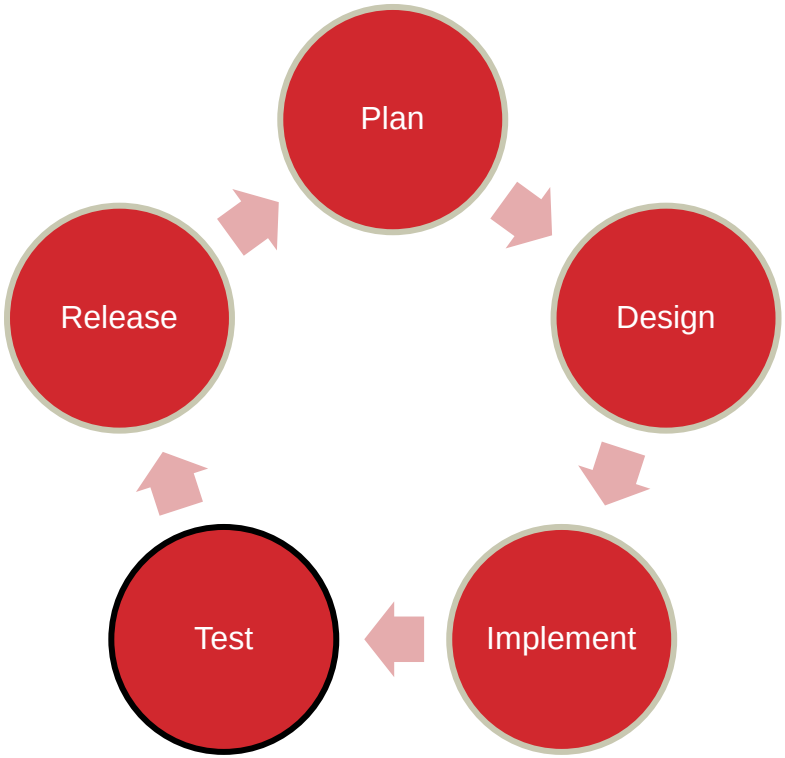


The classic software development cycle

Implement:

- Write code that performs the steps needed to solve the problem
- Use existing code and software libraries whenever possible

OVERVIEW

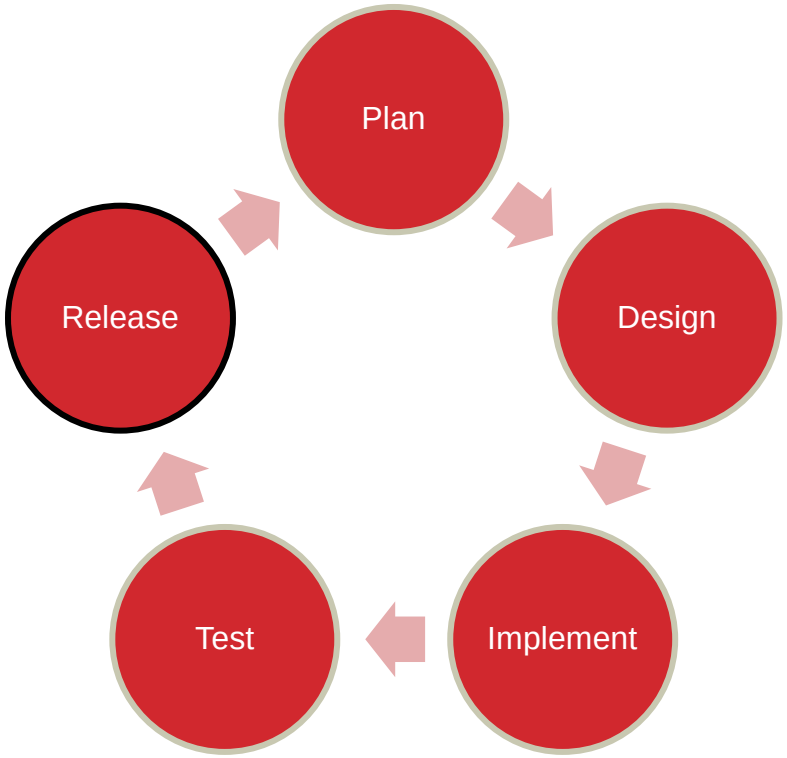


The classic software development cycle

Test:

- Run the program with normal inputs to see if it produces correct outputs
- Run the program with incorrect inputs to check the error handling

OVERVIEW

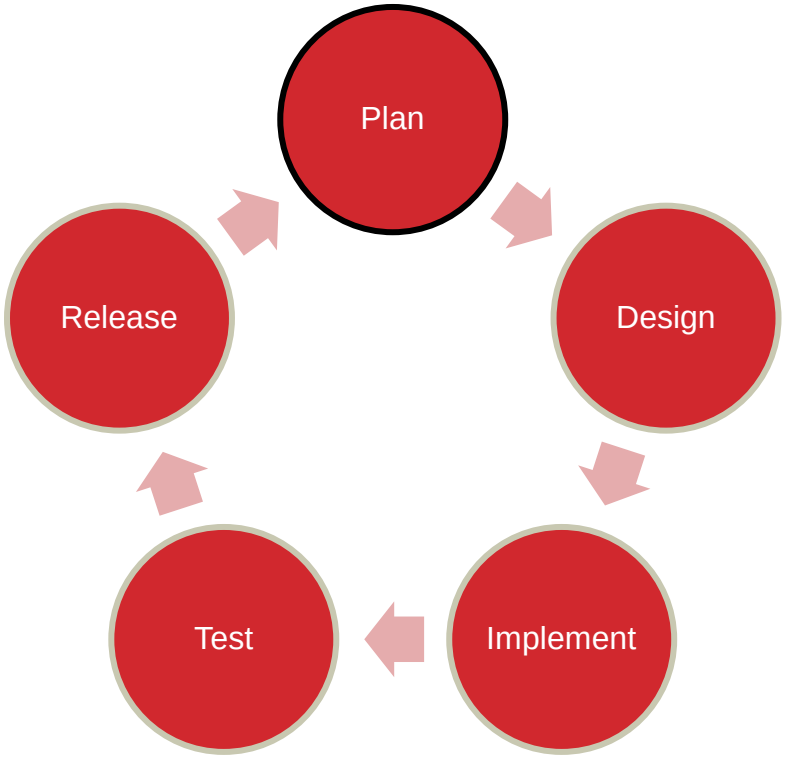


The classic software development cycle

Release:

- Distribute the working program to users
- Collect user feedback to identify problems to fix and new features to add

OVERVIEW



The classic software development cycle

Plan:

- Decide what to do next with the program
- What new features to add
- What problems/bugs to fix

OVERVIEW

- **There are many ways to create programs**
  - Manager: Buy all or part of solution from someone else
  - Mimic: Extend or improve solution to similar problem
  - Inventor: Create new solution from scratch
  - We must be part manager, part mimic, part inventor

- **How can we become great programmers?**
  - Learn programming tools by looking at libraries
  - Learn programming patterns by looking at examples
  - Learn programming skills by writing a lot of code

OVERVIEW

- **How will we learn to program?**
  - We will learn the syntax of the language
    - How to write instructions
  - We will learn semantics of the language
    - What the computer does with instructions
  - We will learn problem solving techniques
    - How to break problems into smaller pieces to solve
  - We will learn how to test and evaluate programs
    - How to find and fix bugs

OVERVIEW

- **Lesson objectives:**
  - Learn the structure of C++ programs
  - Learn how program input / output works
  - Learn about C++ variables and data types
  - Study example program using programming basics
  - Complete online lab on programming basics
  - Complete programming project on programming basics

PROGRAMMING BASICS

PART 1

WHAT MAKES A PROGRAM?

WHAT MAKES A PROGRAM?

- **A program is a sequence of instructions to a computer**
  - Every programming language has its own “rules” describing how these instructions should be written
  - These rules define the “syntax” of the language
  - When the program runs, it will execute your written instructions one line at a time
  - For us to understand what a program will do, we need to know the meaning or “semantics” of each instruction

- **In this section, we will focus on the basic layout of a C++ program and fundamental C++ instructions**

WHAT MAKES A PROGRAM?

- **All C++ programs have the following structure:**
  - Introductory comments – explain the purpose of program
  - Include statements - access to existing function libraries
  - Global data structures - used to store information (later)
  - User defined functions - used to decompose problem (later)
  - Main function - variables and statements for program

- **The following example C++ program prints the message "Hello Mom" to the screen**

WHAT MAKES A PROGRAM?

```
// This program prints a message
#include <iostream>
using namespace std;


int main()
{
   cout << “Hello Mom\n”;
   return 0 ;
}
```

This C++ comment line starts with a // and describes the purpose of the program

WHAT MAKES A PROGRAM?

```
// This program prints a message
#include <iostream>
using namespace std;

int main()
{
   cout << “Hello Mom\n”;
   return 0 ;
}
```

These instructions tell the C++ compiler that we want to use the standard C++ input output library

WHAT MAKES A PROGRAM?

```
// This program prints a message
#include <iostream>
using namespace std;

int main()
{
   cout << “Hello Mom\n”;
   return 0 ;
}
```

This is the main function where the program begins executing C++ instructions

This is the line of code that prints the “Hello Mom” message on the screen

WHAT MAKES A PROGRAM?

```
// This program prints a message
#include <iostream>
using namespace std;


int main()
{
   cout << “Hello Mom\n”;
   return 0 ;
}
```

This C++ command ends the program, so it should be the last line in the main function

HOW TO CREATE AND RUN A PROGRAM

- **Step 1 – Type your C++ program using a text editor and save as a file on disk**
  - % gedit hello.cpp
  - % represents the Linux command prompt
  - hello.cpp is a human-readable file with your C++ program
  - hello.cpp is called your “source code” file
  - the filename for C++ code must end in .cpp

HOW TO CREATE AND RUN A PROGRAM

- **Step 2 – Translate your source code into machine code using a C++ compiler**
  - % g++ -Wall hello.cpp –o hello
  - g++: the name of the C++ compiler
  - -Wall: parameter to compiler to turn all warnings on
  - hello.cpp: the name of the source code file
  - -o hello: the name of the output machine code file
  - hello is called the “executable file”

HOW TO CREATE AND RUN A PROGRAM

- **Step 3 – Execute your program from the Linux command**
  - % ./hello
  - ./ is the name for the current directory
  - hello is the name of the file you want to execute

- **Step 4 – Examine your program output on the screen**
  - If the output is not what you expected
  - Use your editor to modify the source code
  - Recompile your program
  - Run the program again
  - Repeat until program is working correctly

SUMMARY

- **In this section we have studied what a program is and what the basic parts of a C++ program are:**
  - Comments describing the goals of the program
  - Include statements that let us use the input/output libraries
  - The main function containing the code we want to run
  - The return statement at the end of the program

PROGRAMMING BASICS

PART 2

STORING DATA

VARIABLES AND DATA TYPES

- **The most common C++ data types are:**
  - int – stores positive or negative integers (32 bit)
  - float – stores positive or negative real numbers (32 bit)
  - char – stores single character like 'A' .. 'Z'
  - string – stores sequences of characters like “hello mom”
- **Other C++ data types include:**
  - long – stores larger integer values (64 bit)
  - double – stores larger real numbers (64 bit)
  - bool – stores Boolean values (true/false)

VARIABLES AND DATA TYPES

- **We allocate space in the computer memory for data by declaring variables in our program**
  - This memory is **not** automatically initialized

- **The C++ syntax for variable declaration is: “data_type name;”**
  - data_type: This specifies what kind of data can be stored
  - name: We refer to variables by name to perform operations

- **Example:**

```
int Age;          // Can store age in years
float Height;     // Can store height in meters
char Gender;      // Can store 'M' or 'F' for gender
string Name;      // Can store “John” or “Susan” for name
```

VARIABLES AND DATA TYPES

- **Syntax rules for variable names:**
  - Names may contain upper or lower case characters
  - Names may also contain the digits 0..9 and the underscore character, but NO other characters are allowed
  - Names must start with an upper or lower case character

- **Incorrect variable declarations**

```
int float;      // Can not use reserved word 'float' as a name
float 2pi;      // Can not start the name of a variable with digit
int num         // Semi-colon at end of line is missing
```

VARIABLES AND DATA TYPES

- **Make your variable names meaningful**
  - “the_persons_middle_name” is a bit much to type
  - “n” is just to short to have any meaning
  - “per_mid_nme” is too cryptic
  - “middle_name” is about right

- **There are several programming conventions for variables with multi-part names**
  - Use underscore characters: “person_age”
  - Use capital letters for each part: “PersonAge”
  - Use capital letters for all but first part: “personAge”

VARIABLES AND DATA TYPES

- **It is possible to save space in your program by declaring several variables of the same data type on one line**
  - Generally these variables logically belong together

- **The C++ syntax for this is: "type name1, name2, name3;"**

```
float x, y, z;                  // Coordinate of 3D point
int height, length, width;      // Dimensions of a box
string first_name, last_name;   // Student's full name
```

VARIABLES AND DATA TYPES

- **It is a good programming practice to initialize all variables when they are declared**
  - This way we know for sure what the variables contain
  - Otherwise, the compiler will give variables a **random** value

- **The C++ syntax for this is: "data_type name = value;"**

```
int Answer = 42;          // Answer to ultimate question
float Height = 0.0;       // Height in meters
char Gender = 'F';        // Gender of person
string Name = "Susan";    // Name of person
```

CONSTANTS

- **Constants are like variables but they never change value**
  - For example, the quantity PI = 3.14159265… should remain unchanged throughout the program
  - We define constants in C++ by adding the reserved word “const” before a variable declaration
  - We must provide the value of constant at declaration time
  - Constants can be of any variable data type

CONSTANTS

- **Example:**

```
const int SILLY = 42;          // My favorite number
const float PI = 3.14159;      // My second favorite number
const char YES = 'Y';          // Example of character constant
```

- **Conventions when using constants:**
  - Constant names are normally written in upper case
  - Constants are added just below the include statements in a program so they can be used by the whole program

ASSIGNMENT STATEMENTS

- **The operator "=" is used to assign data into a variable**

- **The C++ syntax for assignment is: "name = value;"**
  - name: the variable we wish to copy data into
  - value: the data we want to store in the variable
  - Be sure to put a semicolon at end of the statement

ASSIGNMENT STATEMENTS

- **C++ will automatically convert data types if possible**
  - If variable and value are same type – no conversion
  - If variable is more accurate – no data loss will occur
  - If variable is less accurate – conversion will lose data (most compilers will give you a warning message)

- **Example:**

```
int data1 = 42;          // int value 42 is stored
float data2 = 42;        // float value 42.0 is stored
int data3 = 4.2;         // int value 4 is stored (0.2 is discarded)
float data4 = 4.2;       // float value 4.2 is stored
int data5 = “hello”;     // will not compile
```

ASSIGNMENT STATEMENTS

- **Example:**

```
int Value, Number;
float Data;

Data = 2.158;          // Data variable now equals 2.158
Value = 17;            // Value variable now equals 17
Number = Value;        // Number variable now equals 17
Data = 42;             // Data variable now equals 42.0
Number = 3.14159;      // Number variable now equals 3
```

The floating point value will be truncated and the 0.14159 will be discarded

SUMMARY

- **In this section, we have studied how C++ variables are declared and to store information**
  - Basic data types of the language
  - Rules for choosing variable names
  - How to initialize variables
- **Next we showed how constants can be created**
- **Finally, we described the C++ assignment statement**
  - What happens if we store integer values in float variables
  - What happens if we store float values in integer variables

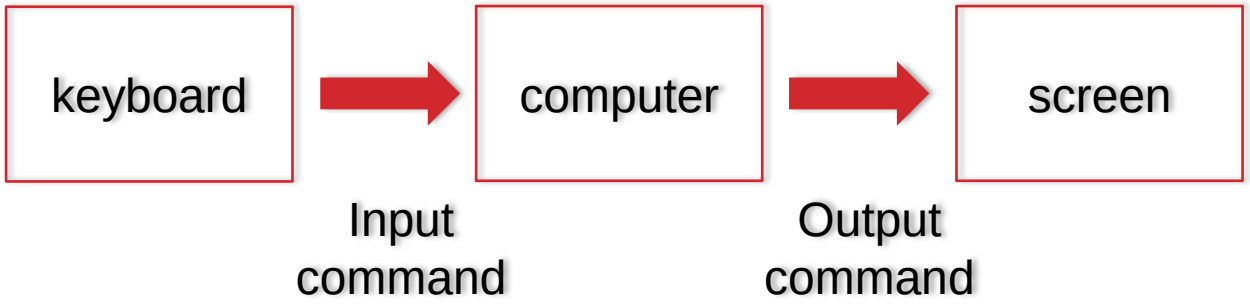
PROGRAMMING BASICS

PART 3

PROGRAM INPUT / OUTPUT

PROGRAM INPUT / OUTPUT

- **We need some way to get data in and out of program**
  - Input commands read values entered on the keyboard
  - Output commands write values onto the screen



PROGRAM INPUT / OUTPUT

- **Many C++ programs have the following pattern:**
  - Print a message to the user with input instructions
  - Read the input typed by the user
  - Print the input values just read by program
  - Do some calculations with the input
  - Print the results of the calculations

- **Next we will go over C++ input / output commands**
  - Cin command for input
  - Cout command for output

PROGRAM INPUT

- **The C++ input command is: cin >> variable;**
  - The “cin” part tell the computer to read from the keyboard
  - The “>>” part tells the computer to read something
  - The “variable” tells the computer where to store the data

- **How is this done?**
  - First, cin will skip over spaces or return characters
  - Then, cin will read characters from the keyboard
  - Then, cin will convert characters to desired data type
  - Finally, cin will store a value in the variable
  - Read and convert steps will **vary** for different data types

PROGRAM INPUT

- **Integer input example:**

```
int number1;
cin >> number1;
```

  - The user types in a sequence of characters “123”
  - The system skips over leading spaces or carriage returns
  - Then the system reads all characters that are digits
  - Then the system converts “123” into an integer 123 and stores this value in the variable number1

PROGRAM INPUT

- **Float input example:**

```
float number2;
cin >> number2;
```

  - The user types in a sequence of characters “3.14159”
  - The system skips over leading spaces or carriage returns
  - Then the system reads all characters that are digits then it reads the “.” then it reads more digit characters
  - Then the system converts “3.14159” into a float value 3.14159 and stores this value in the variable number2

PROGRAM INPUT

- **More on reading float variables…**

- **The user can omit the digits after the decimal point and the cin command will assume they are 0**
  - User input “42.” will be treated like “42.0”

- **The user can omit the digits before the decimal point and the cin command will assume they are 0**
  - User input “.125” will be treated like “0.125”

PROGRAM INPUT

- **Character input example:**

```
char ch;
cin >> ch;
```

  - The user types in a single character ‘y’
  - The system skips over leading spaces or carriage returns
  - Then the system reads a single character ‘y’
  - Then the system stores this character ‘y’ in the variable ch

PROGRAM INPUT

- **String input example:**

```
string str;
cin >> str;
```

  - The user types in a sequence of characters “hello”
  - The system skips over leading spaces or carriage returns
  - Then the system reads sequence of characters “hello”
  - Then the system stores this string in the variable str

PROGRAM INPUT

- **How can we read multiple values from the user?**

- **Solution 1: Use several cin statements**

```
int num1, num2;
cin >> num1;
cin >> num2;
```

First value user types goes in num1
Second value entered goes in num2

- **Solution 2: Use a sequence of >> within the cin statement**

```
float val1, val2;
cin >> val1 >> val2;
```

First value user types goes in val1
Second value entered goes in val2

PROGRAM INPUT

- **Common input errors:**

- **Not enough user input**
  - Cin command will cause program to stop and wait for the user to enter more data
- **Too much user input**
  - Cin will read only the characters it needs to assign a value to the input variable, the rest is left unread
- **Invalid input**
  - Cin will not read any characters, and the input variable will be unchanged by the cin command

PROGRAM INPUT

- **Examples of not enough user input:**

- **User types nothing when input variable is a float**
  - cin >> val;
  - Nothing is read and stored in the variable
  - The program will just sit and wait for input

- **User types “42” when cin is expecting two integers**
  - cin >> num1 >> num2;
  - The value 42 is stored in num1
  - The program will just sit and wait for second input

PROGRAM INPUT

- **Examples of too much user input:**

- **User types “hello mom” when input variable is a string**
  - cin >> str;
  - The string “hello” will be read and stored in the variable
  - The remaining input “ mom” will be unread

- **User types “yes” when input variable is a character**
  - cin >> ch;
  - The character ‘y’ is read and stored in the variable
  - The remaining input “es” will be unread

PROGRAM INPUT

- **Examples of invalid input:**

- **User types “123” when input variable is a string**
  - cin >> str;
  - The string “123” will be read and stored in the variable

- **User types “hello” when input variable is an integer**
  - cin >> num;
  - There are no digits in “hello”, so cin will not read any characters, and the input variable will be set to zero

PROGRAM OUTPUT

- **The C++ output command is: cout << variable;**
  - The “cout” part tell the computer to write to the screen
  - The “<<” part tells the computer to write something
  - The “variable” tells the computer what data to write

- **How is this done?**
  - First, cout will look at variable to get its value
  - Then, cout will convert value to sequence of characters
  - Then, cout will output these characters on the monitor
  - The convert step will **vary** for different data types

PROGRAM OUTPUT

- **Integer output example:**

```
float number1 = 123;
cout << number1;
```

  - The system converts the integer value of the variable 123 to a sequence of ascii characters “123”
  - The system displays the characters “123” on the screen at the current cursor position

PROGRAM OUTPUT

- **Float output example:**

```
float number2 = 3.14;

cout << number2;
```

  - The system converts the float value of the variable 3.14 to a sequence of ascii characters “3.14”
  - The system displays the characters “3.14” on the screen at the current cursor position

PROGRAM OUTPUT

- **Character output example:**

```
char ch = 'y';
cout << ch;
```

  - No conversion to ascii character is needed since the variable is already an ascii character
  - The system displays the character "y" on the screen at the current cursor position

PROGRAM OUTPUT

- **String output example:**

```
string str = “hello mom”;

cout << str;
```

  - No conversion to ascii character is needed since the variable is already a sequence of ascii characters
  - The system displays the character “hello mom” on the screen at the current cursor position

PROGRAM OUTPUT

- **Spaces are NOT automatically written between values**
  - int var1=42, var2=17;
  - cout << var1;
  - cout << var2;
  - This will print “4217” without spaces between values

- **We must print the spaces between values ourselves**
  - int var1=42, var2=17;
  - cout << var1 << “ ”;
  - cout << var2 << “ ”;
  - This will print “42 17 ” with spaces after both values

PROGRAM OUTPUT

- **We can use the reserved word “endl” to print a carriage return after data values**
  - int var1=42, var2=17;
  - cout << val1 << endl;
  - cout << val2 << endl;
  - This will print “42” on one line and “17” on the next line

PROGRAM OUTPUT

- **We can also print out any of the following special characters inside a string to format our output**

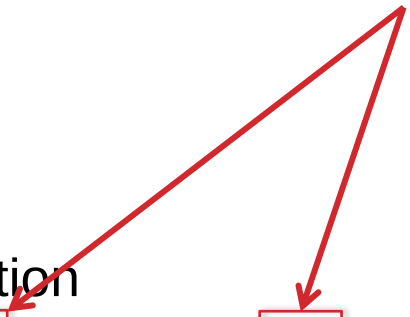
\n	Carriage return
\t	Tab character
\b	Back space
\f	Form feed
\a	Bell sound
\’	Single quote
\”	Double quote
\\	Backslash character

PROGRAM OUTPUT

Example:

```
// Initialize student information
string first = "John";
string last = "Smith";
int age = 21;
float gpa = 3.14;

// Print student information
cout << "First Name:\t" << first << "\n";
cout << "Last Name:\t" << last << "\n";
cout << "Age:\t\t" << age << "\n";
cout << "GPA:\t\t" << gpa << "\n";
```



We are printing tab and carriage return characters to make the output look nice

PROGRAM OUTPUT

Sample program output:

```
First Name:    John
Last Name:     Smith
Age:           21
GPA:           3.14
```

Notice how all output is nicely aligned with each other

COMMENTS

- **Comments are an essential part of all programs**
  - Comments are used to explain the design and implementation of a program
  - They are human readable and are ignored by the compiler
  - Programmers should write comments as the program is being written and when major changes are made

- **Do NOT wait "until the program is finished" to write your comments**
  - Comments are there to help you write the program
  - In real life, programs are never "finished", there are always security updates and new features added

COMMENTS

- **C++ supports two types of comments**

- **C++ style comments are a single line long (recommended)**
  - These comments start with // and go to end of the line

```
// Here is a new C++ style comment
// This is the second line of the comment
```

- **C style comments can span multiple lines (in older code)**
  - These comments start with /* and end with */

```
/* Here is an old C style comment
   This is the second line of the comment */
```

SUMMARY

- In this section we have studied the “cin” command for reading and storing information from users
- We also discussed the “cout” command for writing variables and other information to the screen
- Finally, have described how C++ comments are formed and their importance in writing clear programs

PROGRAMMING BASICS

PART 4

NUMERICAL CALCULATIONS

ARITHMETIC EXPRESSIONS

- **Arithmetic expressions are used to perform numerical calculations using variables and arithmetic operators**

- **Once the values of arithmetic expressions are evaluated, they can be printed out using cout, or stored in variables using the assignment operator**

- **The rules for arithmetic expressions in C++ is very similar to the rules we learn in mathematics, but there are some subtle differences we will discuss below**

ARITHMETIC EXPRESSIONS

- **What is the syntax for arithmetic expressions?**
  - Arithmetic expressions consist of an alternating sequence of values and arithmetic operators
  - Values can be numerical literals, variables, or constants
  - Arithmetic operators include

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo (remainder after integer division)

  - Parentheses ( ) can be used to control the order of evaluation of sub-expressions

ARITHMETIC EXPRESSIONS

- **Examples of valid arithmetic expressions:**
  - 7 + 2 * 5
  - 21 - num / 2
  - (2 + 2 + 2) / (3 - 3 - 3)

- **Examples of invalid arithmetic expressions:**
  - 17 * ← missing value after * operator
  - (num - 9 * 5 ← missing closing parenthesis
  - cin + 42 ← cin is not a valid variable name

ARITHMETIC EXPRESSIONS

- **How are expressions evaluated?**
  - We follow the “natural” rules of mathematics
  - Multiplication, division, modulo have high precedence
  - Addition, subtraction have low precedence
  - The result of high precedence operations are calculated before low precedence operations (i.e. * before +)
  - Operations in the expression are calculated left to right at same precedence level
  - Parenthesized expressions ( ) are calculated first, and are evaluated from the inside out

ARITHMETIC EXPRESSIONS

- **Evaluation examples:**
  - 7 + 2 * 5

    = 7 + 10 ← perform multiplication

    = 17 ← perform addition

  - 21 - num / 2

    = 21 - 10 / 2 ← substitute variable value

    = 21 - 5 ← perform division

    = 16 ← perform subtraction

ARITHMETIC EXPRESSIONS

- **Evaluation examples:**
  - (2 + 2 + 2) / (3 - 3 - 3)

= (4 + 2) / (3 - 3 - 3)	← perform leftmost addition
= 6 / (3 - 3 - 3)	← perform addition
= 6 / (0 - 3)	← perform leftmost subtraction
= 6 / -3	← perform subtraction
= -2	← perform division

ARITHMETIC EXPRESSIONS

- **What happens if we mix data types in expressions?**
- **C++ will look at the data types and choose the most accurate data type for each arithmetic operation**
- **The ordering of data types from least accurate to most accurate is: char, short, int, long, float, double**

int OP int	← int result
char OP int	← int result
int OP float	← float result
float OP double	← double result

ARITHMETIC EXPRESSIONS

- **Mixed type examples:**
  - 3 * 5 + 4.2

    = 15 + 4.2 ← integer multiplication

    = 19.2 ← float addition

  - (16 - num) / 4.0

    = (16 - 10) / 4.0 ← variable substitution

    = 6 / 4.0 ← integer subtraction

    = 1.5 ← float division

ARITHMETIC EXPRESSIONS

- **In C++ there is an important difference between float division and integer division**

- **Float division always returns a float result**
  - 3.0 / 2.0 = 1.5

- **Integer division always returns an integer result**
  - 3 / 2 = 1 ← the 0.5 is discarded !!

ARITHMETIC EXPRESSIONS

- **Integer division examples:**
  - (16 - num) / 4

    = (16 - 10) / 4 ← variable substitution

    = 6 / 4 ← integer subtraction

    = 1 ← integer division (0.5 discarded)

  - (1 + 2) / (3 + 6)

    = 3 / (3 + 6) ← integer addition

    = 3 / 9 ← integer addition

    = 0 ← integer division (0.333 discarded)

ARITHMETIC EXPRESSIONS

- **In C++ modulo operator % is used to calculate the value of the remainder after an integer division**
  - Both arguments to the % operator must be integers
  - If not the compiler will give error messages

- **Modulo operator examples:**
  - 285 % 10
    = 5 ← 285 / 10 = 28, remainder is 5
  - 285 % 100
    = 85 ← 285 / 100 = 2, remainder is 85

TYPE CASTING

- **C++ will do implicit type conversion in assignment statements if the value type does not match variable type**
  - The value is converted to match the variable type
  - Sometimes compilers will warn of possible loss of data

- **Examples:**
  - int num = 4.2; // value 4 is stored
  - float val = 17; // value 17.0 is stored
  - int sum = 1 + 2.0; // value 3 is stored
  - float total = num + sum; // value of 7.0 is stored

TYPE CASTING

- **Type casting in C++ lets us convert a value from one data type to another in the middle of arithmetic expressions**
  - This is very useful if we want to force the expression to use integer operations or float operations

- **There are two equivalent ways do type casting**
  - (data_type) value
  - static_cast<data_type>(value)

- **Type casting has the highest precedence, so the type conversion is done before the next arithmetic operation**

TYPE CASTING

- **Type casting examples:**
  - 2 / 3
    = 0 ← integer division
  - (float) 2 / 3
    = 2.0 / 3 ← converts value to float
    = 0.666 ← float division
  - 1 / static_cast<float>(3)
    = 1 / 3.0 ← converts value to float
    = 0.333 ← float division

SPHERE EXAMPLE

- **Assume we want to calculate the volume and surface area of a sphere of any size**

- **How can we perform this calculation?**
  - Look up formulas for sphere volume and surface area

- **How can we implement this?**
  - Write a program to prompt user for sphere radius
  - Calculate sphere volume and surface area
  - Print the results of these calculations

SPHERE EXAMPLE

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
  // Local variable declarations
  // Read sphere radius and echo input
  // Calculate volume and surface area
  // Print output
  return 0;
}
```

With the first version of the program we just type in comments to describe our approach

The rest of the program is our "standard empty program" boiler plate

SPHERE EXAMPLE

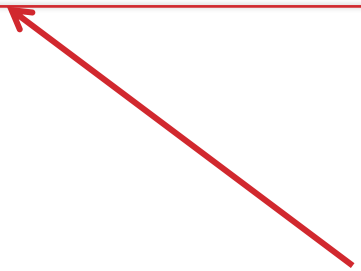
```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
```

```
// Local variable declarations
float Radius = 0.0;
float Volume = 0.0;
float Area = 0.0;
```

Next we add code for the each of the steps in our approach one chunk of code at a time

SPHERE EXAMPLE

```
…
 // Read sphere radius and echo input
cout << "Enter sphere radius: ";
cin >> Radius;
cout << "Radius = " << Radius << endl;
…
```



It is always a good idea to print the values you have read from the user to verify cin worked as expected

SPHERE EXAMPLE

We are using float literals here to force the result to be a float value (using 4/3 would produce incorrect result due to integer division)

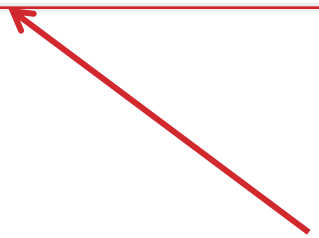
```
…
  // Calculate sphere volume
  Volume = (4.0 / 3.0) * M_PI * Radius * Radius * Radius;

  // Calculate sphere surface area
  Area = 4.0 * M_PI * Radius * Radius;
…
```

M_PI = 3.141592653 is a constant defined in the <cmath> library

SPHERE EXAMPLE

```
…
   // Print output
   cout << "Volume = " << Volume << endl;
   cout << "Area = " << Area << endl;
   return 0 ;
}
```



Finally we add the code to output our answers

SPHERE EXAMPLE

To compile on a Linux system:

```
g++ -Wall sphere.cpp -o sphere
```

To run on a Linux system:

```
./sphere
```

SPHERE EXAMPLE

Sample program output:

```
Enter sphere radius: 1.0
Radius = 1
Volume = 4.18879
Area = 12.5664

Enter sphere radius: 10
Radius = 10
Volume = 4188.79
Area = 1256.64
```

SOFTWARE ENGINEERING TIPS

- **Think about the problem you are trying to solve before you start writing your program**
  - What data do you need to solve problem?
  - What formulas are you going to use?
  - Work out a few examples by hand to be sure you understand the process you are going to use

- **Start your program by writing your comments**
  - Add your name and date at top of program
  - Describe steps in program in point form
  - Add code to your program a little at a time
  - Compile and test program incrementally

SOFTWARE ENGINEERING TIPS

- **Top-down problem solving has the following steps:**
  - Understand the problem to be solved
  - Decompose problem into smaller pieces you can solve
  - Write computer instructions for each piece
  - Combine pieces into a single program
  - Compile, test, and debug program
  - Use program to solve initial problem

SOFTWARE ENGINEERING TIPS

- **Bottom-up problem solving has the following steps:**
  - Understand the problem to be solved
  - Look at similar problems to identify common components
  - Design and implement general purpose components
  - Combine components into a single program
  - Compile, test, and debug program
  - Use program to solve initial problem

SOFTWARE ENGINEERING TIPS

- **Make your program easy to read and understand**
  - Pick variable names that are meaningful to you and others
  - Add blank lines and white space to separate calculations
  - Indent your code using a consistent convention

- **Make sure your program is running correctly**
  - Initialize all variables before you use their values
  - Print out intermediate results as you debug code
  - Test with “normal” and “unexpected” input values
  - Document all known bugs/limitations in the code

SUMMARY

- In this section we have studied the syntax and use of arithmetic expressions to do numerical calculations
- We also showed an example program demonstrating the use of arithmetic expressions and input/output
- Finally, have discussed several software engineering tips for creating and debugging programs