

# Writing and Understanding C++

- **Writing programs in any language requires understanding the syntax and semantics of the programming language as well as language-independent skills in programming.**
  - **Syntax is similar to rules of spelling and grammar:**
    - *i before e except after c*
    - The relationship between a command and a quote, “*this is a fact,*” or “*this is a fact*”,
  - **Semantics is what a program (or English sentence) means**
    - You ain’t nothing but a hound dog.
    - La plume de ma tante est sur la porte.
- **At first it seems like the syntax is hard to master, but the semantics are much harder**
  - **Natural languages are more forgiving than programming languages.**

# Toward an Understanding of C++

- **Traditional first program, doesn't convey power of computing but it illustrates basic components of a simple program**

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

// traditional first program

int main()
{
    cout << "Hello world" << endl;
    return 0;
}
```

- **This program must be edited/typed, compiled, linked and executed.**
- **Other languages don't use compile/link phase, examples?**

# Anatomy of a C++ Program

- **#include statements make libraries of classes and functions accessible to the program**
  - **Compiler needs access to interface, what the functions look like, but not to implementation**
  - **Linker/Loader needs access to implementations**
  - **Helps programmers develop code independently**
- **Comments make programs readable by humans**
  - **The cost of program maintenance is often far greater than the cost of program development**
  - **Use comments liberally, but make them meaningful**

# More C++ Anatomy

- **Programmer-defined functions**
  - Functions are abstractions that help you to reuse ideas and code
  - The square root key on a calculator invokes a function
  - The chorus of a song is a similar abstraction
  - One word, e.g., “chorus”, takes the place of many or represents a concept
- **A program is a collection of functions and classes**
- **Programs may be implemented in more than one file, but there is only one main function**
  - Execution of the program begins with `main`
  - The `main` function returns a value to the operating system or environment

# Dennis Ritchie

- Developed C and Unix
- Shared 1983 Turing award and National Medal of Science in 1999

**“We wanted to preserve not just a good environment in which to do programming, but a system around which a fellowship could form”**

- **Unix was**
  - Free to Universities
  - Expensive originally
  - Linux precursor?



# Execution and Flow Control

- **Execution of C++ programs is organized around statements**
  - **A statement executes, it may cause another statement to execute**
  - **Statements execute sequentially, or as governed by control that repeats a group of statements or selects one of several groups to execute**
    - **Control statements covered later; for now sequential flow**
- **Syntax determines what's in a statement, semantics determines construction of program from statements**
- **Output will be part of our programs**
  - **cout is the output stream, objects are placed on the stream**
  - **Objects are strings, numbers, many other *types***

# Stream output

- **cout is the standard output stream, use cerr for errors and other streams later. Accessible via `#include<iostream>`**

- **Objects inserted onto stream with insertion operator `<<`**
- **Different objects separated by insertion operator `<<`**

```
cout << "yadda yadda yadda" << endl;  
cout << " gross = " << 12*12 << endl;  
cout << 5 << " in. = "  
    << 5*2.54 << " cm. " << endl;
```

- **String *literals* in quotes, other *expressions* are evaluated before being output.**
  - **endl is the “end of line” object (IO manipulator)**
  - **Can also output `"\n"` or `"\t"` or `"\""` (escape sequences)**

# More about streams and syntax

- **C++ statements are terminated by a semi-colon**

```
cout << 3.14159*10*10 << " = area "  
      << " of circle with radius = "  
      << 10 << ", circ = " << 2*10*3.14159  
      << endl;
```

- **Thinking ahead:**
  - Repetition of radius, problems?
  - Repetition of  $\pi$ , problems?
  - What's better, several statements, or one long statement?
  - Evaluating expressions: rules of arithmetic?
  - Differences between  $2*3$  and  $2*3.0$  ?

# Toward Using Functions

```
#include <iostream>
using namespace std;
int main()
{
    cout << " ||||| ||||| ||||| ||||| ||||| " << endl;
    cout << " | " << endl;
    cout << " | o o | " << endl;
    cout << " _| _ " << endl;
    cout << " | _| " << endl;
    cout << " | |_____| | " << endl;
    cout << " | " << endl;
    return 0;
}
```

- Prints head, but not as *modular* as program using functions
  - Harder to modify to draw differently

# Programmer-defined Functions

```
#include <iostream>
using namespace std;
// functions appear here

int main()
{
    Hair();
    Sides();
    Eyes(); Ears(); Smile();
    Sides();
    return 0;
}
```

- **What are advantages of this main over one in which several output statements appear in main.**
  - **New hair style? Stretched head?**
  - **Are these advantages?**
  - **How is width of head determined? Drawbacks? Solutions?**

# Advantages of Functions

```
#include <iostream>
using namespace std;
// functions appear here

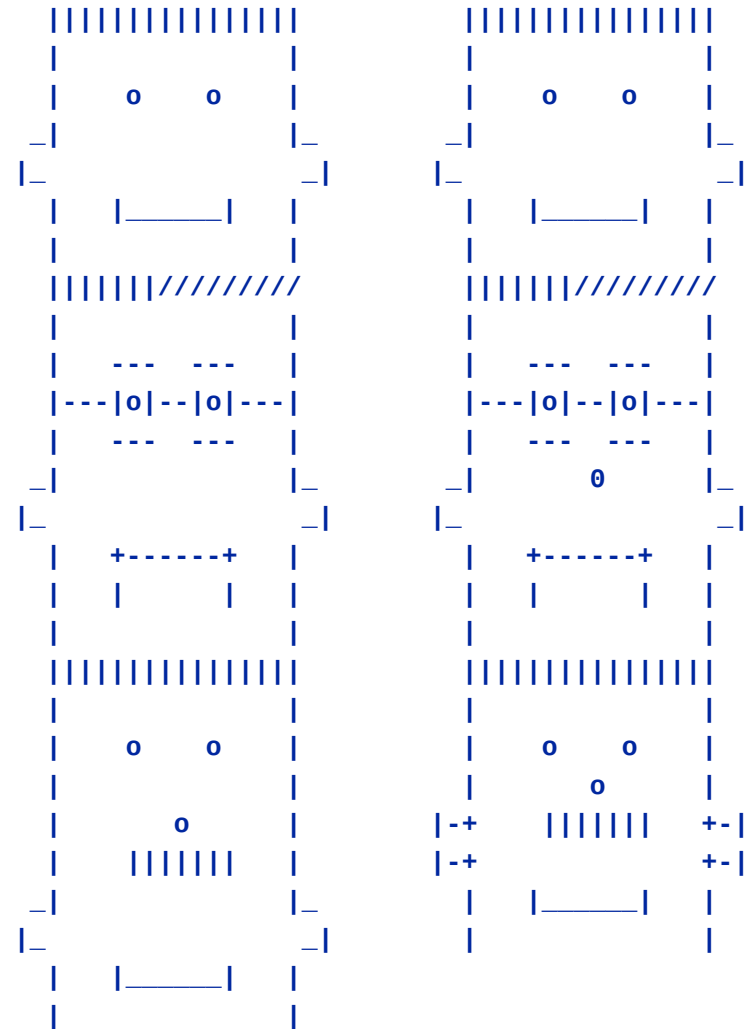
int main()
{
    Hair();
    Sides();
    Eyes(); Ears(); Smile();
    Sides();
    return 0;
}
```

- **What about eyeglasses? Mustache? Big nose? Frown?**
  - **Advantages in extending program rather than modifying program.**
  - **Multiple heads (totem poles)**

# Totem Functions

```
int main()
{
    Head1();
    Head2();
    Head3();
    return 0;
}
```

- What changed between the two runs of the program?
- Can you write Headxx( ) ?
  - Is Head1 a good name?
  - Does Headxx call other functions?
  - Suppose we used graphics instead of cout << ?



# Parameterized Functions

- **A square root function that only returns square root of 2 isn't very useful**
  - **F = sqrt(2), so 2 is a parameter/argument to the function**
  - **Useful parameter to head-drawing functions?**
    - 
    -
  - **What about happy birthday printing argument/parameter?**
- **Functions have parameters, *arguments* are *passed* to functions**

```
Birthday("Fred");    // sing to fred  
Birthday("Ethel");    // sing to ethel
```

# Functions and Parameters (continued)

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

void WinBigMoney(string name)
{
    cout << "Hello " << name << " you may have "
          << " won $1,000,000" << endl;
    cout << name << ", please call 1-900-IMN-IDIOT"
          << endl;
}

int main()
{
    Hello("owen"); Hello("susan");
    Hello("bill gates");
    return 0;
}
```

- **Parameter list provides *type* and *name* of parameter**
  - **Argument type must match parameter type**
  - **Function's *prototype* based on types only, not names**

# Parameterized Functions for Songs

- **On his farm Old MacDonald had a X that that says Y**
  - pig, oink
  - cow, moo

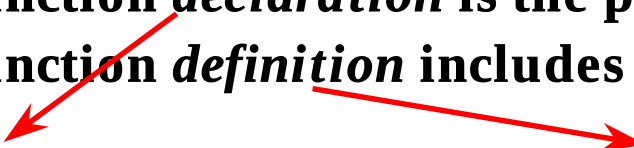
```
void Verse(                );
```
- **Five bottles of Z on a wall, five bottles of Z**
  - cola
  - lemonade

```
void Verse(                );
```
- **Mama's going to buy you a X, and if that X Y**
  - Mocking bird, don't sing
  - Looking glass, get's broke

```
void Verse(                );
```

# Calling Functions: where, when, how?

- **Some functions are imported from libraries**
  - Function prototypes specified in header files, implementations linked later
  - Compiler “sees” prototype before client code calls function
- **Some functions are in the same file in which they’re called**
  - Function *declaration* is the prototype only
  - Function *definition* includes the implementation



```
void Verse(string name);      void Verse(string name)
                               {
                               cout << "hi " << name << endl;
                               }
```

- **Declaration or definition must appear before call**
  - Ok to put declaration before, definition after call
  - Ok to put main last, all definitions first (problems?)

# Ada Lovelace, 1816-1853

- Daughter of Byron, advocate of work of Charles Babbage, designer of early “computer” (the Analytical Engine)
  - Made Babbage’s work accessible
- *“It would weave algebraic patterns the way the Jacquard loom weaved patterns in textiles”*
- Tutored in mathematics by Augustus de Morgan
- Marched around the billiard table playing the violin
- Ada is a notable programming language

# Program Style

- **People who use your program don't read your code**
  - **You'll write programs to match user needs**
- **People who maintain or modify your program do read code**
  - **Must be readable, understandable without you next door**
  - **Use a consistent programming style, adhere to conventions**
- **Identifiers are names of functions, parameters, (variables, classes, ...)**
  - **Sequence of letters, numbers, underscore \_\_ characters**
  - **Cannot begin with a number (we won't begin with \_\_)**
  - **big\_head vs. BigHead, we'll use ALTeRnAtInG format**
  - **Make identifiers meaningful, not droll and witty**