Programs that Respond to Input

- Programs in chapters one and two generate the same output each time they are executed.
  - Old MacDonald doesn’t get new animals without editing and recompiling the program
    - Drawbacks in editing and recompiling?
  - Allow the user to input values that generate output
    - Calculators respond to buttons pressed by users, programs respond to values entered by users
- Sequential model of programming: input, process, output
  - Interactive model of programming: entities communicate with each other continuously
  - We’ll start with IPO, input, process, output

C++ Review, Programming Process

- C++ programs begin execution in main
  - Statements are executed (can you identify a statement?)
  - Sometimes expressions are evaluated:
    ```cpp
    cout << "gpa = " << grades/totalCourses << endl;
    ```
  - Function calls execute a group of statements that embody an abstraction (e.g., Verse, EiEiO, ...)
- C++ programs must import needed declarations via #include directives (not statements, why not?)
  - Streams in `<iostream>`, used for ???
  - Strings in `<string>`, used for ???
  - Built-in types include int (integer), double (real number) and many operators like +, -, *, ... are NOT imported

C++ and Programming Review

- Functions have prototypes (or signatures) that indicate to both the compiler and the programmer how to use the function
  - Later functions will return values, like square root
  - For now, `void` means no value is returned
  - Every function has a parameter list, but it’s possible to have no parameters
    ```cpp
    Hello(); Verse("pig","oink");
    ```
    - What do prototypes look like for these calls?
- Function must appear before it’s called, either the function declaration (prototype only) or definition (implementation)

Programming Review

- You’ll design and implement C++ programs
  - Written in a high-level language, should run on many platforms, e.g., Windows, Unix, Mac, ...
  - Compiler translates C++ into low-level machine language
  - Different compilers generate different low-level programs
    - Efficiency concerns, portability concerns, proprietary...
- To execute, programs must link libraries — implementations of what’s imported via #include directives
  - `<iostream>`, `<string>`, many more “standard”
  - Tapestry library
- Errors can result if when programs use libraries incorrectly
  - Fail to include, fail to link, fail to use properly
Toward a User-controlled Barnyard

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

void Verse(string animal, string noise)
{
    ... cout << "on his farm he had a " << animal << endl;
}

int main()
{
    Verse("pig","oink");
    Verse("elephant","hrruyaahungh");
    return 0;
}
```

What can we do to allow user to enter animal and noise?

Desired Program Behavior

- We want the user to enter/input values

Enter animal name: sheep
Enter noise: baah
Old MacDonald had a farm, Ee-igh, Ee-igh, oh!
And on his farm he had a sheep, Ee-igh, ee-igh, oh!
With a baah baah here
And a baah baah there
Here a baah, there a baah, everywhere a baah baah
Old MacDonald had a farm, Ee-igh, Ee-igh, oh!

- We'll pass the user-entered values to the Verse function
  - The input stream cin takes input from the keyboard using operator `>>`
  - Values that are input are stored in variables (aka objects)

Input values are stored in variables

```cpp
void Verse(string animal, string noise)
{
    // this function doesn't change
}

int main()
{
    string animal;    // defined, no value supplied
    string noise;     // defined, no value supplied
    cin >> animal;
    cin >> noise;
    // what goes here??
    Verse(animal,noise);
    return 0;
}
```

- Each variable has a type, a name/identifier, and a value

Variables and Parameters

- Both are placeholders for values. Each has a type and a name
  - Parameters are given values when arguments passed in a function call:
    ```cpp
    void Verse(string animal, string noise){...}
    Verse("duck", "quack");
    ```
  - Variables are given values when initially defined, or as a result of executing a statement
    ```cpp
    string animal;    // defined, no value supplied
    cout << "enter animal ";
    cin >> animal;    // user-entered value stored
    ```
Define variables anywhere, but …

- Two common conventions for where to define variables.
  - At the beginning of the function in which they’re used:
    ```
    { string animal, noise; 
    cout << "enter animal "; 
    cin >> animal; 
    cout << "enter noise a " << animal << " makes "; 
    cin >> noise;
    }
    ```
  - Just before the first place they’re used:
    ```
    string animal; 
    cout << "enter animal "; 
    cin >> animal; 
    string noise; 
    cout << "enter noise a " << animal << " makes "; 
    cin >> noise;
    ```

Defensive programming

- When your program fails, you want to be able to find the cause quickly and without tearing your hair out
  - Give each variable a value when it is defined
    ```
    string animal = “UNASSIGNED”; 
    cout << "enter animal "; 
    cin >> animal; 
    //…
    ```
  - If, for some reason, the extraction >> fails, animal will have an identifiable value.
  - What is the value if no initial assignment and extraction fails?
- Read << as "puts-to" or "inserts", read >> as "extract" ???

Using numbers in a program

```
#include <iostream>
using namespace std;
int main()
{
    double degrees;
    cin << "enter temperature in degrees F. ";
    cin >> degrees;
    cout << "degrees = ";
    cout << (degrees-32) * 5 / 9 << endl;
    return 0;
}
```

- User can enter 80 or 80.5
  - There are two types for numbers, double and int, why?
  - Are parentheses needed in (degrees-32)? Why?

Some arithmetic details

- C++ adheres to traditional order of operations
  - * and / have higher precedence than + and –
    ```
    int x = 3 + 5 * 6;    int y = (3 + 5) * 6;
    ```
  - Parentheses are free, use them liberally
- Arithmetic expressions are evaluated left-to-right in the absence of parentheses
  ```
  int x = 3 * 4 / 6 * 2;    int y = (3*4)/(6*2);
  ```
- There are limits on int and double value, be aware of them.
Variables and Parameters for Numbers

- The type string is not a built-in type, technically it’s a class
  - What must you do to use strings in your programs?
  - What alternatives are there if strings not supported?

- There are many numerical types in C++. We’ll use two
  - int, represents integers: {...,-3,-2,-1,0,1,2,3,...}
    - Conceptually there are an infinite number of integers, but the range is limited to $[-2^{31}, 2^{31}-1]$ (on most systems)
    - Alternatives? Why is range limited?
  - double, represents real numbers like $\pi$, $\sqrt{2}$
    - Not represented exactly, so expressions like $100 * 0.1$ may yield unexpected results
    - Double precision floating point numbers, another type float exists, but it’s a terrible choice (generates poor results)

GIGO: program as good as its data?

- In calculations involving floating point numbers it’s easy to generate errors because of accumulated approximations:
  - What is $10^{23} + 1$?
  - When is $(x + y) + z$ different from $x + (y + z)$?

- The type int is severely constrained on 16-bit computers, e.g., running DOS, largest value is $32,767$ ($2^{15}-1$)
  - Even on 32-bit machines, how many seconds in a millennium? $60 * 60 * 24 * 365 * 1000$, problems?
  - On UNIX machines time is measure in seconds since 1970, problems?
  - What was Y2K all about?

What arithmetic operations exist?

- Syntax and semantics for arithmetic operations
  - Addition, subtraction: + and −, int and double
    - $23 + 4 \quad x + y \quad d - 14.0 + 23$
  - Multiplication: *, int and double
    - $23 * 4 \quad y * 3.0 \quad d * 23.1 * 4$
  - Division: /, different for int and double
    - $21 / 4 \quad 21 / 4.0 \quad x / y$
  - Modulus: %, only for int
    - $21 \% 4 \quad 17 \% 2 \quad x \% y$

- Mixed type expressions are converted to “higher” type
- Associativity of operators determines left-to-right behavior
- Use parentheses liberally
  - Without () use operator precedence, *, /, % before +, -

Comparing Dominos to Pizza Hut to ...

void SlicePrice(int radius, double price)
// compute pizza statistics
{
    // assume all pizzas have 8 slices
    cout << "sq in/slice = ";
    cout << 3.14159*radius*radius/8 << endl;

    cout << "one slice: $" << price/8 << endl;
    cout << "$" << price/(3.14159*radius*radius);
    cout << " per sq. inch" << endl;
}

- How can we call this several times to compare values?
- Are there alternatives to the 8 slices/pie convention?
- What about thickness?
Parameter, compiler, warning, trouble

- What if argument types don’t match parameter types?

```c
void SlicePrice(int radius, double price);
```

- Consider the calls below, which are ok?
  - `SlicePrice(12, 18.99);`
  - `SlicePrice(12, 18);`
  - `SlicePrice(18.99, 12);`
  - `SlicePrice(12, "18.99");`

- What is a compiler warning as opposed to an error? Should you pay attention to warnings?

Compiling and linking, differences

```c
#include <string>
int main()
{
    string s = "hi";
}
```

A Question of Style

Coding style can be a personal thing but there do exist a number of good guideline

Elements

- Indentation / White space
- Meaningful identifiers
  - Variables: *What is it?*
  - Functions: *What does it do?*
- Use case to indicate what type of identifier (e.g. variables lowercase and functions uppercase)
- Commenting
  - Abstraction comments: *What does it do?*
  - Implementation comments: *How does it do it?*

*Choose one style that is comfortable to you and be consistent!*

Think about it

Puzzle: Toggling Frogs

- You have 100 light switches, numbered 1-100, and 100 frogs, also numbered 1-100.
- Whenever a frog jumps on a light switch, it toggles a light between on and off. All lights are initially off.
  - frog #1 jumps on every light switch (ie turning them all on).
  - frog #2 jumps on every 2nd light switch, toggling some of them back off.
  ...
  - frog #k jumps on every kth light switch.
- After 100 frogs, which lights are on?

Game: Don’t be last

- You and a friend have a stack of 10 coins.
- On each person's turn, they remove either 1 or 2 coins from the stack.
- The person who removes the last coin wins.
- What is a winning strategy? Should you go first or second?