Control, Functions, Classes

- We've used built-in types like `int` and `double` as well as the standard class `string` and the streams `cin` and `cout`.
  - Each type supports certain operations and has a specific range of values.
    - What are these for the types we've seen so far?
  - We need more than these basic building blocks, why?
- We've used void functions to encapsulate concepts/statements with one name, avoid repeated code, help develop programs.
  - Functions with parameters are useful.
  - We need functions that return values to solve more problems than we're currently able to solve.

Types of control

- Selection: choose from among many options according to criteria the programmer codes (from which the user chooses).
  - If response is yes do this, else do that.
  - If year is a leap year number of days is 366, else 365.
  - If PIN is incorrect three times, keep banking card.
  - If 10th caller, we have a winner.
- Repetition (next chapter), repeatedly execute statements until criteria met.
  - Print twelve months of a calendar.
  - Allow three attempts at PIN entry.
  - Make moves in game until game is over.

Problem solving leads to programming

- Which is the better value, a 10 inch, $10.95 pizza or a 12 inch $15.95 pizza?
  - Details needed to solve the problem (no computer)?
  - What's missing from programming repertoire?
  - Print two price/sq. in values, let user make conclusions.
  - Program should determine best value after calculating.
- We need selection (why?) and we'd like a function to return a value for comparison (what's the function?)

```cpp
if ( PizzaValue(10,10.95) > PizzaValue(12,15.95) )
    cout << "10 inch pizza is better value" " endl;
```

First step, the assignment operator

- Avoid repeated calculations.

```cpp
void SpherePizza(double radius, double price) {
    double volume;
    volume = 4.0/3*radius*radius*radius*3.1415;
    double area;
    area = 4*radius*radius*3.1415;
    cout " area = " << area << endl;
    cout " volume = " << volume << endl;
    cout " $/cu.in " << price/volume << endl;
}
```

- Assign a value to a variable to give it a value.
  - We have used input stream to enter values for variables.
  - Read the assignment operator as `gets`, "area gets ..."
    - Avoids confusion with equality operator we'll see later.
Calculating change (see change.cpp)

```cpp
int main()
{
    int amount;
    int quarters, dimes, nickels, pennies;
    cin >> "make change in coins for what amount: ";
    amount = amount - quarters*25;
    amount = amount - dimes*10;
    amount = amount - nickels*5;
    amount = amount - pennies;
    // more code here, see the full program
}
```

Problems with code in change.cpp?

```cpp
// previous code for entering value, doing quarters
dimes = amount/10;
amount = amount - dimes*10;
nickels = amount/5;
amount = amount - nickels*5;
pennies = amount;
```

Control via selection, the if statement

```cpp
void Output(string coin, int amount)
{
    if (amount > 0)
    {
        cout << "# " << coin << " =" << amount << endl;
    }
}
```

Selection using if/else statement

```cpp
int main()
{
    string name;
    cin >> name;
    if (name == "Ethan")
    {
        cout << "that's a very nice name" << endl;
    }
    else
    {
        cout << "might be a nice name" << endl;
    }
    return 0;
}
```
More Operators: Relational

- The guard/test in an if statement must be a Boolean expression (named for George Boole)
  - Values are true and false
  - bool is a built-in type like int, double, but some older compilers don't support it (very old)
```cpp
int degrees;
bool isHot = false;
cout << "enter temperature: ";
cin >> degrees;
if (degrees > 95) isHot = true; // or below
isHot = degrees > 95;
```
- Relational operators are used in expressions to compare values: <, <=, >, >=, ==, !=, used for many types
  - See Table 4.2 and A.4 for details, precedence, etc.

Details of Relational Operators

- Relational (comparison) operators work as expected with int and double values, what about string and bool?
  - 23 < 45  49.0 >= 7*7  "apple" < "berry"
- Strings are compared lexicographically (alphabetically) so that "ant" < "zebra" but (surprisingly?) "Ant" < "zebra"
  - How do lengths of strings compare?
  - Why does uppercase 'A' come before lowercase 'a'?
  - (Actually "Ant" < "zebra" doesn't work, need string)
- Boolean values have numeric equivalents, 1 is true, 0 is false
  - cout << (23 < 45) << endl;
  - cout << ("guava" == "Guava") << endl;

Logical operators

- Boolean expressions can be combined using logical operators: AND, OR, NOT
  - C++ equivalents are &&, ||, and !, respectively
  - (standard requires and, or, not, most compilers don't)
```cpp
if (90 <= grade)
    if (grade < 95)
        cout << "that's an A" << endl;
    } 
else
    cout << "goodbye" << endl;
```
- What range of values generates 'A' message? Problems?
  - if (90 <= grade && grade < 95)
    cout << "that's an A" << endl;
  }
Short-circuit Evaluation

- Subexpressions in Boolean expressions are not evaluated if the entire expression’s value is already known.

```c
if (count != 0 && scores/count < 60)
   { cout << "low average warning" << endl; }
```

- Potential problems if there are no grades to average? What happens in this case?

- Alternatives in absence of short-circuit evaluation:

```c
if (count != 0)
   {  if (scores/count < 60)
       {   cout << "low average warning" << endl; }
   }
```

- Examples when OR short-circuits?

---

Donald Knuth (b. 1938)

- Scholar, practitioner, artisan
  - Has written three of seven+ volumes of The Art of Computer Programming
  - Began effort in 1962 to survey entire field, still going

- Strives to write beautiful programs
  - Developed TeX to help typeset his books, widely used scientific document processing program

- Many, many publications
  - First was in Mad Magazine
  - On the Complexity of Songs
  - Surreal Numbers

---

It’s all relative and it depends

- I make the best bread in the city
- I make the best bread in the world
- I make the best bread on the block

---

Richard Stallman (born 1953)

- Described by some as “world’s best programmer”
  - Wrote/developed GNU software tools, particularly g++
  - Believes all software should be free, but like “free speech”, not “free beer”
  - Won MacArthur award for his efforts and contributions
  - League for Programming Freedom

- Gnu/Linux is a free operating system and computing environment
  - Heavy industry/web use
  - Wintel killer??

- Local tie-in: Red Hat Linux,
  - headquartered in Durham, NC
  - IPO in 1999 at $14
  - One month later at $110+
  - Markets “free” product
Functions that return values

- Functions we’ve written so far allow us to decompose a program into conceptual chunks: void functions
  - Each function call is a statement, not used in an expression
    
    ```c
    DoThis();
    DoThat();
    Sing("cow", "moo");
    WriteHTMLHeader();
    ```

- Perhaps more useful are functions that return values:
  
  ```c
  double hypotenuse = sqrt(a*a + b*b);
  int days = DaysIn("September");
  string userID = GetCurrentUser();
  ```

Functions that return values

- Function prototype indicates return type
  - Nearly any type can be returned, all types we’ll use can be
    - A function call evaluates to the return type, the call must be part of an expression, not a stand-alone statement
      - Yes: double hypotenuse = sqrt(a*a + b*b);
      - No: sqrt(a*a + b*b);
      - ?: cout << sqrt(100) << endl;
      - ?: double adjacent = cos(angle)*hypotenuse;
      - ?: if ( sqrt(x*x + y*y) > min) {};
      - ?: cos(3.1415) == -1;

- The math functions are accessible using #include<cmath>,
on older systems this is <math.h>

Anatomy of a function

- Function to calculate volume of a sphere
  
  ```c
  double SphereVol(double radius)
  {
    return 4.0*radius*radius*radius*acos(-1)/3;
  }
  ```

- Function prototype shows return type, void functions do not return a value
- The return statement alters the flow of control so that the function immediately exits (and returns a value)
- A function can have more than one return statement, but only one is executed when the function is called (see next example)

Functions can return strings

- What function call looks like?
  
  ```c
  string dayName;
  int dayNum;
  cout << "enter day (0-6): ";
  cin >> dayNum;
  dayName = WeekDay(dayNum);
  ```

- Which is/are ok? Why?
  
  ```c
  cout << WeekDay(5) << endl;
  int j = WeekDay(0);
  cout << WeekDay(2.1) << endl;
  string s = WeekDay(22);
  WeekDay(3);
  ```
Another version of WeekDay

```cpp
string WeekDay(int day) {
    // precondition: 0 <= day <= 6
    // postcondition: return "Sunday" for 0, ...
    "Monday" for 1, ...
    "Saturday" for 6

    if (0 == day) return "Sunday";
    else if (1 == day) return "Monday";
    else if (2 == day) return "Tuesday";
    else if (3 == day) return "Wednesday";
    else if (4 == day) return "Thursday";
    else if (5 == day) return "Friday";
    else if (6 == day) return "Saturday";
}
```

- Every occurrence of else can be removed, why?
- Why aren’t the braces [...] used in this version?

Function documentation

- Functions usually have a precondition
  - What properties (e.g., of parameters) must be true for function to work as intended?
  - If there are no parameters, sometimes no precondition
  - Some functions work for every parameter value
    ```cpp
double sqrt(double val);
    // precondition:
    string LoginID(string name)
    // precondition:
```

- Functions always have a postcondition
  - If precondition is satisfied what does the function do, what does the function return?

Free functions and member functions

- The functions in `<cmath>` are free functions, they aren’t part of a class
  - C++ is a hybrid language, some functions belong to a class, others do not
  - Java is a pure object-oriented language, every function belongs to a class

- We’ve used string objects in programs, string is a class
  - String variables are objects, they’re instances of the class
  - A class is a collection having members that have common attributes (from American Heritage Dictionary)
  - Strings share many properties, but have different values
  - My little red corvette, her 1958 corvette, his 1977 corvette

String member functions

- The function `length()` returns the number of characters
  ```cpp
  string s = "hello";
  int len = s.length(); // value of len is 5
  s = "";
  len = s.length(); // what is value of len here?
  ```

- Member functions are applied to objects using `dot` notation
  ```cpp
  cannot use length() without an object to apply it to
  not valid  int x = length(s);
  valid?  double y = sqrt(s.length());
  ```
Finding substrings

- A substring is part of a string. Substrings can be extracted from a string using member function `substr(…)`

```cpp
string s = "theater";
int len = s.length(); // value of len is ??
string t = s.substr(0,3); // t is "the", s is ??
t = s.substr(1,4); // t is now ??
s = s.substr(3,3); // s is ?? t is ??
```

- Function prototype for `substr`

```cpp
string substr(int pos, int len);
// pre: 0 <= pos < s.length()
// post: returns substring of len characters
//       beginning at position pos
//       ok if len too big, NOT ok if pos too big
```

Find pieces of symbolic IP addresses

- Pieces are separated by a period or dot
- Assume at most four pieces, first is the 0-th piece
- Prototype for function is:

```cpp
string NthIP(string IP, int n);
// pre: 0<= n < 4
// post: return n-th piece of IP, return "" if there is no n-th piece
```

What are the values of each variable below?

```cpp
string first = NthIP("cs.duke.edu",0);
string last = NthIP("cs.duke.edu",2);
string xxyy = NthIP("cs.duke.edu",100); // 3?
```

We need `find` to write `NthIP`

- String member function `find` looks for an occurrence of one string in another, returns position of start of first occurrence
- If no occurrence, then string::npos is returned

```cpp
string s = "I am the eggman";
int k = s.find("I"); // k is 0
k = s.find("he"); // k is 6
k = s.find("egg"); // what is k?
k = s.find("a"); // what is k?
k = s.find("walrus"); // what is k?
s = "duke.edu";
k = s.find("."); // what is k?
if (k != string::npos)
{
    s = s.substr(k+1,s.length()); // what is s?
}
```

How to get started writing `NthIP`?

```cpp
string NthIP(string s, int n)
// pre: 0<= n < 4
// post: return n-th piece of IP s, return "" if there is no n-th piece
{
    int len = s.length();
    int pos = s.find(".");
    if (pos == string::npos) return "";
    if (1 == n) // s must have dot, why?
    {
        return s.substr(0,pos);
    }
    s = s.substr(pos, len); // what's value of s?
}
```

```cpp
string s = NthIP("duke.edu",1); // trace the call
```
When is a year a leap year?

- Every year divisible by four is a leap year
  - Except years divisible by 100 are not
    - Except years divisible by 400 are

- Alternatively:
  - Every year divisible by 400 is a leap year
  - Otherwise, years divisible by 100 are not leap years
  - Otherwise, years divisible by 4 are leap years
  - Otherwise, not a leap year

```cpp
bool IsLeap(int year);
// post: return true iff year is a leap year
```

Once more again, into the leap

```cpp
bool IsLeap(int year)
// post: return true iff year is a leap year
{
    if (year % 400 == 0)
    {   return true;
    }
}
```

```cpp
int main()
{
    if (IsLeap(2000)) cout << "millennium leap" << endl;
    else              cout << "Y2K bug found" << endl;
    return 0;
}
```

There’s more than one way to ...

```cpp
bool IsLeap(int year)
// post: return true iff year is a leap year
{
    return ( year % 400 == 0 ) ||
           ( year % 4 == 0 && year % 100 != 0);
}
```

- How does this work?
  - Why isn’t an if/else necessary?
  - What’s the value of an expression formed from Boolean operators?
  - Is this version more efficient?
  - Are these two versions different? From what perspective?

Preview: the class Date

- In addition to `int`, `double`, and `string`, there are several standard C++ classes and several classes standard to A Computer Science Tapestry
  - Most C++ classes designed to be “industrial strength”
    - This often means efficiency at the expense of safety
    - Easy to hang yourself, shoot yourself in the foot, ...
  - Tapestry classes designed for novice programmers
    - Sacrifice some efficiency, but often not noticeable
    - Make it run, make it run, make it fast:
      - it’s better to write correct code than to write fast code

- The class `Date` is accessible using `#include “date.h”, the class represents calendar dates, e.g., June 14, 1999`
What can you do with a `Date`?

```cpp
#include <iostream>
#include <date.h>

int main()
{
  int month, year;
  cin >> month >> year;
  Date d(month, 1, year);
  cout << "that day is " << d << ", it is a 
   " << d.DayName() << endl;
  cout << "the month has " << d.DaysIn() << " days in it " << endl;
  return 0;
}
```

Date member functions

- `Date d(9, 15, 1999);`
- `Construct` a `Date` object given month, day, year
- `Problems in other countries?`
- `Other useful ways to construct a Date?`
- `d.DayName()`
- Returns "Saturday", "Sunday", and so on
- `d.DaysIn()`
- Returns the number of days in the month
- `Other functions you think might be useful?`

DeMorgan’s Law: Boolean operators

- Writing complex Boolean expressions can be tricky
  - Prompt user for a number, print a message if the value entered is anything other than 7 or 11 (e.g., 2, 3, 22, ...)

- Prompt user for "rock", "paper", "scissors", print message if anything else is entered

DeMorgan continued

- Logical equivalents
  - !(a && b)             (!a) || (!b)
  - !(a || b)             (!a) && (!b)

- If 7 and 11 are legal values, what are the illegal values?
  - if (value == 7 || value == 11) // ok here

- How to write a statement for illegal values:
  - if (                ) // not ok