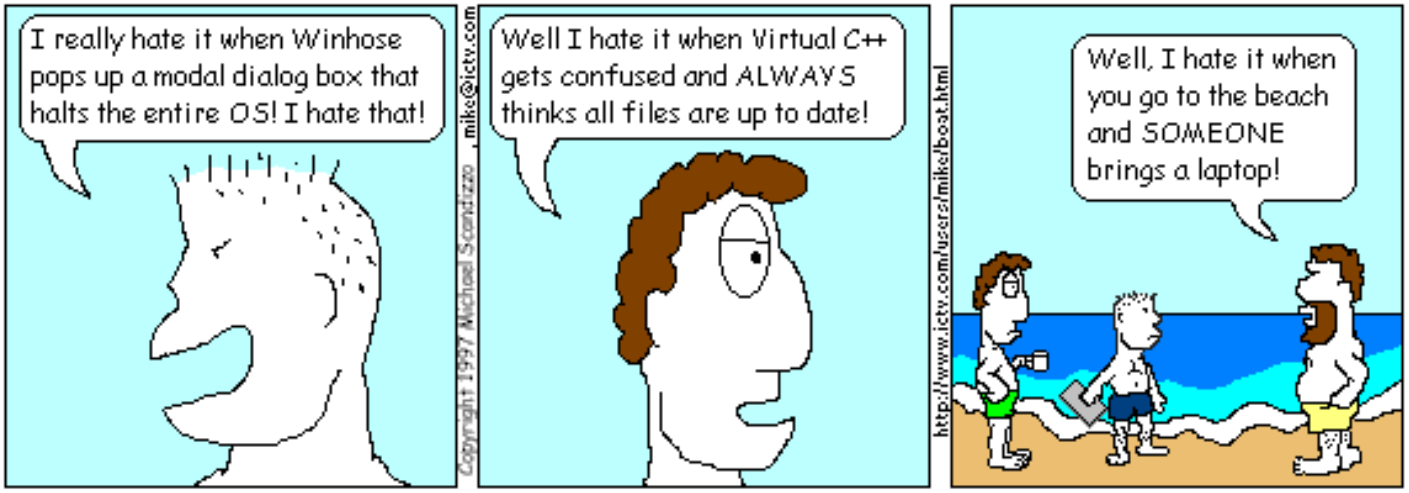
ORF 201

COMPUTER METHODS FOR PROBLEM SOLVING

Lecture 22

C/C++ vs. Java



Most Simple Things Are the Same

```
#include <stream.h>
#include <math.h>

// To compile:  CC sqrt.c -lm -o sqrt

/*************************************************
 * Program to compute square roots using Newton's method.
 *************************************************/

#define EPSILON 1.0e-10

main()
{
    double x, rootx, err;

    while ( cin >> x ) {
        rootx = x;
        err = fabs( x - rootx*rootx );
        while ( err > EPSILON ) {
            rootx = (rootx + (x/rootx))/2;
            err = fabs( x - rootx*rootx );
        }
        cout << form("sqrt(%f) = %f \n", x, rootx);
    }
}
```

Comments are the same.

main is essentially the same.

Basic variable types are the same (except **boolean** is **bool** in C++ and nonexistent in C).

Arithmetic expressions are the same.

Input/output is different (variable-number-of-argument methods).

There is a *preprocessor* for loading library information and setting *constants*.

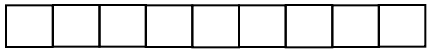
Stuff Needn't be Enclosed in a Class

```
#include <stream.h>

// To compile:  CC power.c -o power

double pow( double a, int n); // This is a function prototype

main()
{
    int i;

    for (i=0; i<10; i++)  {
        cout << form("%2d %15.5f %15.5f \n",
                i, pow(2,i), pow(-3,i));
    }
}

double pow( double a, int n)
{
    int i;
    double p;

    p = 1;
    for (i=0; i<n; i++) {
        p = p * a;
    }
    return p;
}
```

No "enclosing" class - in a sense the file itself does that.

Compiler must see a method's *prototype* or its definition before using the method for the first time.

Methods are called *functions* in C/C++.

Arrays 

Arrays of a fixed predetermined size:

```
double x[100];
```

Dynamic Arrays:

```
int n;
double *x;      // a pointer to an array of doubles
n = 100;
x = (double *) malloc(n*sizeof(double));
n=200;
x = (double *) realloc( x, n*sizeof(double) );
free(x);
```

Classes

C doesn't have classes. Use **`typedef struct`** instead for container-type classes.

```
typedef struct ZIP {
    int zip;
    double lat;
    double lon;
    char *name;   // a pointer to an array of characters
} ZIP;

ZIP z;

z.zip = 90210;
```

C++ has classes. They are similar to Java.

Call-By-Value

Java is call-by-value but you need to remember that object variables are, in fact, pointers to the objects.

In C/C++, struct/object variables are the real thing. Therefore an entire copy gets passed because C is strictly call-by-value (and in C++ that is the default behavior).

To have an "effect above", must explicitly pass a pointer to an object. In C, it's mighty ugly:

```
void above()
{
     double x = 3.14;
     below(&x);  // ampersand means "address of"
     cout << x;
}

void below(double *xx)
{
     *xx = 2.71; // star means "value of"
}
```

C++ is somewhat cleaner:

```
void above()
{
     double x = 3.14;
     below(x);
     cout << x;
}

void below(double &x) // the ugliness shifts to here
{
     xx = 2.71;
}
```

Mimicking Java-Style Objects (in C++)

Still need one "star" - when declaring the pointer.

```
Zip *z;

z = new Zip(08544, 45.2, 76.4);

z->zip = 90210;
z->lat = 39.5;
z->lon = 112.4;

delete z;
```

z->zip is legal shorthand for **(*z).zip**

The "dot" operator works with objects, not with pointers to objects.

No garbage-collection. Every call to **new** needs a later call to **delete**.