Computer Programming: Skills & Concepts (CP1)
Programming Languages

22nd November 2010
Varieties of Programming Language

- Procedural/imperative (like C)
  - Language consists of statements which *act on the state space of variables*.
  - Functions, procedures common.

- Functional Languages (eg Haskell, Lisp)
  - Specify *what* is computed, but abstract away from *how*.
  - The concept of an evolving state space (of program variables) is *not* explicit.

- Object-Oriented Languages
  - Focus is on the organisation and representation of the state space.
Fibonacci in Lisp

(defun fibonacci (n)
  (if (or (= n 0) (= n 1))
      1
      (+ (fibonacci (- n 1)) (fibonacci (- n 2))))
)

- defun - define a function.
- In functional programming almost *everything* is a function, even at basic level
- Notice in fibonacci (n) that we have no variables to store n-1 or n-2. Instead we apply the function – to the arguments n and 1 (and 2 respectively)
Compilation versus Interpretation

C is usually a compiled language:
- Programming cycle is write/compile/run.
- Compiler generates code to run on the hardware of the machine.
- Fast, compact and efficient (once compiled).

Sometimes languages (especially functional) may be interpreted:
- The encoding into machine code is done on a step-by-step basis.
- Allows for dynamic creation of variables and data structures.
- Can be good for debugging.
- Slower execution, requires interpreter.
Imperative/procedural languages

C
- Need to be careful with array bounds (as we know!).
- Allows direct access to memory.
- Good for direct interfacing to hardware and writing device drivers.
- Pointers get you into trouble.

Fortran
- Bit old fashioned, but still used (good for numerical work).
- UK Met Office Unified Model - millions of lines of Fortran.
- Limited feature set - less to go wrong.
- Easy to make a fast compiler.
Features of Fortran

- No explicit pointers (special case in F90)
  ▶ Easier to automatically optimise code.
- Very stable numerical libraries available.
- In F77, no dynamic storage allocation.
  - *Cannot do recursion* (but can in F90).
- All variables passed by reference.
  ▶ Faster than by value.
- Variable dimension array arguments to functions.
  ▶ Required by many numerical algorithms.
- Built-in complex numbers.
Functional languages

What are they?

- Emphasis is the evaluation of expressions, rather than the execution of commands – \texttt{comp.lang.functional}

- Important in theoretical computer science, not used so often in practice.

- Haskell is perhaps the most popular functional language.
Sum integers from 1-10

C

total = 0;
for (i=1; i<=10; ++i)
    total += i;

Functional language.

sum [1..10]

- sum is a function to compute the sum of a list of values.
- [1..10] is an expression representing the list containing the numbers from 1 to 10
Object-Oriented Languages
typedef struct {
  float re, im;
} Complex_t;

Complex_t ComplexSum(Complex_t z1, Complex_t z2) 
/* Returns the sum of z1 and z2 */
{
  Complex_t z;
  z.re = z1.re + z2.re;
  z.im = z1.im + z2.im;
  return z;
}
Used in practice

```c
int main(void)
{
    Complex_t z, z1, z2, z3, z4;
    z1 = MakeComplex(1.0, -5.0);
    z2 = MakeComplex(3.0, 2.0);
    z3 = MakeComplex(2.0, -7.0);
    z4 = ComplexMultiply(z1, z2);
    z = ComplexSum(z4, z3);

    printf("The modulus of z is \%f\n", Modulus(z));
    return EXIT_SUCCESS;
}
```

Evaluating the expression \( z = (z1 \times z2) + z3 \).
C++ and objects

C groups similar data into a struct:
- Functions which operate on those data are separate from the data itself.

C++ groups the functions operating on some struct type:
- C++ calls these classes.
- An instance of a class is called an object.
- The ‘functions’ don’t exist until the object is created.

Complex c1,c2,c3 ;
c3 = c1.multiply(c2);
Operator overloading

C++ allows re-definition of standard operators
- eg Complex number multiplication with *.
- Also could define * for matrices etc

Complex c1,c2,c3 ;
c4 = c1 * c2 + c3;
Common OO Languages

C++
- Extension to the C language.
- Has objects, but also still C pointers and memory access.
- Compiles directly on the machine, like C.

Java
- Cleaner than C++ - no pointers.
- ‘Compiles’ onto a virtual machine.
- Portable across platforms - and web applets.
- Slower than C++ - and less efficient.
Inheritance

- Can define generic classes with general properties.
- Then subclasses can be *derived* from this base class.
- For example generic class (in C++) for a vehicle:
  ```cpp
  char colour[50] ;
  int numWheels ;
  int start() ;
  int stop() ;
  ```
- Derived class for a car:
  ```cpp
  char typeOfFuel ;
  ```
Object-Oriented design

- What are classes? - sometimes obvious - complex numbers.
- Some tasks fit the model very well.
- Sometimes difficult to see where the objects are in a design.
  - Some tasks are just a sequence of functions.
Common Data Structures

Queue - a dynamic list of items
  - first-in is first-out.

Stack - first-in, last-out.

Both these structures have implementations with faster access than arrays (because no need for *random* access).
Implementing a Queue

You are implementing a queue for an accounting system.
You implement a queue for customer records.
Now you need a queue for messages too?

You have to re-write the queue to work with the new ‘message’ type?
C++ templates

- Way of writing objects (eg data structures) that is independent of the type that it works with.
- Write a generic queue \textit{template} with a type parameter T.
  - T can be replaced with any data type.
  - (eg) Our queue can be used with any data type.
- Change details of template \implies all queues automatically change.
- Very useful for common operations.
  - lists, sorting, searching, queues etc.
- Useful set of templates provided in the \textit{Standard template library}.
Examples: vectors in C++ (using arrays)

```c
void f(int a[], int s) {
    /* do something with a; the size of a is s */
    for (int i = 0; i<s; ++i)
        a[i] = i;
}

int arr1[20];
int arr2[10];

void g() {
    f(arr1,20);
    f(arr1,20); /* CRASH !! */
    f(arr2,20); /* CRASH !! */
}
```
#define S 10;

void f(int s) {
    int a1[s]; /* error */
    int a2[S]; /* ok */
    /* Arrays have to be declared at compile time. */
    * ...
}

Using arrays
const int S = 10;
void g(int s) {
    vector<int> v1(s); /* ok */
    vector<int> v2(S); /* ok */

    v2.resize(v2.size() * 2);
    /* Can resize arrays during runtime. */
}


Vector template

```cpp
void f(vector<int>& v) {
    /* do something with v */
    for (int i = 0; i<v.size(); ++i)
        v[i] = i;
}

vector<int> v1(20);
vector<int> v2(10);

void g() {
    f(v1);
    f(v2);
}
```

Equivalent code with C++ vectors
Summary

C
▶ Good general purpose language.
▶ Good for interfacing with hardware.
▶ Not good for big projects (organisationally).

Fortran
▶ Good for numerical computation.
▶ Stable, well-supported.

C++
▶ Use with the standard template library.

Java
▶ Widely used, good for web applets, and neater than C++
▶ Not as fast or efficient as C++.