Introduction to Scientific Computing

Part II: C and C++

C. David Sherrill

School of Chemistry and Biochemistry

Georgia Institute of Technology

The C Programming Language:

- “Low-level” operators
- Created by Dennis Ritchie for the DEC PDP-11 UNIX operating system, 1970’s
- ANSI standard 1983
- A superset called C++ for object-oriented programming (Stroustrup, 1980+).
- C/C++ currently dominant programming language (used to write, e.g., operating systems)

C and Scientific Computing

Scientific computing was traditionally done with Fortran. C was slow to catch on during the 1980's. C/C++ taken more seriously as scientific programs became more complex.

Ab initio programs:

Gaussian-9x: Maybe 60% Fortran, 40% C??

Q-Chem 2.0: 10% C++, 50% C, 40% Fortran

PSI 3.0: 20% C++, 60% C, 20% Fortran

NWChem: C and Fortran

MPQC: 100% C++ (and Curt++ !)

A Minimal C Program

```
#include <stdio.h>

main()
{
  printf("Hello, world!\n");
}
```

N.b. This is not good ANSI C! Strictly speaking, should have types.

C is a "Typed" Language

All variables (and functions) must have a given *type*. Some allowed types:

int integer

float floating point number, single-precision (bad)

double floating point number, double-precision (good)

char character value

FILE file structure

void A weird catch-all meaning nothing or anything

In C++, you make up your own data types!

An ANSI Approved Hello World

```
#include <stdio.h>

int main(void)
{
  printf("Hello, world!\n");
  return(0);          /* exit w/ success status */
}
```

Compiling a Program

To compile a simple C program in UNIX, you invoke the C compiler (usually named `cc`) like this:

```
cc hello-world.c -o hello-world
```

This would compile a C file called `hello-world.c` (containing the previous example, perhaps) and make an *executable* program called `hello-world`. The executable need not have the same name as the program file. If no name is given by the `-o` switch, the program will be named `a.out` by default.

To compile a C++ program, one would use the C++ compiler [often named cpp, g++ (GNU), or xlC (IBM)].

To compile two C files into one executable, one first compiles the C source into *object* files

```
cc -c hello-world.c other-file.c
```

creating `hello-world.o` and `other-file.o`. The object files are *linked* into the final executable:

```
cc -o hello-world hello-world.o other-file.o
```

Often this can be done all in one step as a shortcut like this:

```
cc -o hello-world hello-world.c other-file.c
```

Linking Libraries

Frequently one wishes to call standard library functions, such as the square root function `sqrt()` from the math library, etc. These libraries are special files with names ending in a `.a` suffix (the `a` stands for "archive"). Names of libraries usually start with the previx `lib`, as in `libm.a`, the C math library.

To link against a library one uses the `-l` flag. The math library can be included by a command like:

```
cc hello-world.c -o hello-world -lm
```

The `-l` flag automatically adds a `lib` prefix and `.a` suffix to determine the library name.

Makefiles for Large Programs

Programs containing more than a few source code files are best compiled using a special program called `make`. The `make` command reads a file called `Makefile` to determine how to compile the program, what libraries to link, etc. An example follows:

```
ROOT = /home/users/sherrill/C
LIBS = -L$(ROOT)/lib -lm -lds_io -lds_str
CFLAGS = -I$(ROOT)/include -O

CC = cc

NOBJ = biblio.o cparse.o format.o
```

```
SRC = $(NOBJ:%.o=%.c)

biblio: $(NOBJ)
        $(CC) $(CFLAGS) $(NOBJ) $(LIBS) -o biblio

clean:
        /bin/rm -f $(NOBJ)

# DO NOT DELETE THIS LINE -- make depend depends on it
biblio.o: biblio.c
cparse.o: cparse.c
format.o: format.c
```

A More Complex Program Example

```
#include <stdio.h>

main()
{
  double x, y;
  double crazy_function(double x);

  x = 4.0;
  y = crazy_function(x);
  printf("The result is %lf\n", y);
}
```

```
double crazy_function(double x)
{
   double z;

   x = x * x;
   z = x + 1.0;
   return(z);
}
```

When run, the program prints

```
The result is 17.000000
```

Pass-by-Value

Suppose we modified the previous example as such:

```
x = 4.0;
y = crazy_function(x);
printf("The result is %lf\n", y);
printf("The value of x is %lf\n", x);
```

What's `x` ? You might think 16.0, but it's 4.0. How could you modify `crazy_function` so `x` would really be changed by it?

Solution I: C Pointers

```
  y = crazy_function(&x);

double crazy_function(double *x)
{
   double z;

   *x = *x * *x;
   z = *x + 1.0;
   return(z);
}
```

Solution I: C++ References

```
  y = crazy_function(x);

double crazy_function(double &x)
{
   double z;

   x = x * x;
   z = x + 1.0;
   return(z);
}
```

Exactly same as original except for declaration of `crazy_function`.

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

main()
{
  double x, y;

  x = 4.0;
  y = sqrt(x);
  printf("The result is %lf\n", y);
}
```

```
The result is 2.000000
```

Why didn’t we need to declare `sqrt`?

Subroutines Also Called Functions

```
main()
{
  double x;
  void dumb_subroutine(double x);

  x = 4.0;
  dumb_subroutine(x);
}
void dumb_subroutine(double y)
{
  printf("The result is %lf\n", y);
}
```

What's So Great About C++ ?

- Retains C as a subset
- Has nice new featues like references, constants, and especially user-defined datatypes or *objects*
- Object-oriented language: seen as big advantage for very large codes
- Not very efficient; may need to do computationally intensive subroutines in C or Fortran (or call optimized math library like BLAS)

Object-Oriented Programming and C++

- Contrasts to *procedural programming*
- The program consists of objects which know how to relate to each other
- Objects "hide" their own data and can only be accessed through their interfaces — keeps others from messing up your beautiful code
- Separation of interface from implementation makes upgrades easier
- C++ programmers tend to write insanely complex code; natural result of taking object ideas to their limit

Suggested Reading

"The C Programming Language, 2nd ed.," Brian W. Kernighan and Dennis M. Ritchie (Prentice Hall, Englewood Cliffs, NJ, 1988).

"The C++ Programming Language, 3rd ed.," Bjarne Stroustrup (Addison-Wesley, Reading, MA, 1997).