Computing in C for Science

Lecture 5 of 5

Dr. Steven Capper
steven.capper99@imperial.ac.uk
http://www2.imperial.ac.uk/~sdc99/ccourse/

14th December 2011

Imperial College London

Cryptic C - What do I do?

```
void whatdoIdo(double *A, double * B, double *C,
                   unsigned int L, unsigned int M,
                   unsigned int N)
{
   unsigned int i, k;
   double temp, *bptr, *bend, *cptr;

   for(i=0; i < L; i++)
   {
      for (k=0; k < M; k++)
      {
         temp = A[i*M+k];
         cptr = &C[i*N];
         bptr = &B[k*N];
         bend = &B[(k+1)*N];
         while (bptr < bend)
            *cptr++ += temp*(*bptr++);
      }
   }
}
```


Projects and Makefiles

- It is possible (and encouraged) to build a program from multiple `.c` files.
- This maximises the portability of the code, and
- Speeds up compiling - if we only change one `.c` file we only need to recompile one file...
- Visual Studio manages programs in to so-called *projects*, and everything is done graphically.
- UNIX systems have a very powerful program called `make` which manages projects. Information for building programs is stored in a `Makefile`.

A sample `Makefile`

There are many ways of writing one, this is mine!

```
CFLAGS = -O2 -DNDEBUG -Wall -ansi
LFLAGS = -lm
CC = gcc
CLEANFILES = crypticc.o matrixfunctions.o cryptic cryptic.exe

cryptic: crypticc.o matrixfunctions.o
        $(CC) crypticc.o matrixfunctions.o $(LFLAGS) -o cryptic

clean:
       touch $(CLEANFILES)
       rm $(CLEANFILES)
```

- This compiles `crypticc.c` and `matrixfunctions.c`.
- It then links them to produce `cryptic.exe` (Cygwin) or `cryptic` (UNIX).
- It has two rules `cryptic` (default) to build the program and `clean` to clean up all the compiled output.

Bytes (or chars)

- We've mentioned `char`s briefly so far, and used them extensively whilst drawing little attention to them.
- `char` is the smallest data type in C, `sizeof(char)`=1 byte (this is explicitly stated in the standard).
- We can perform integer computations using `char` and `unsigned char`.
- The most common use for `char` is as part of a string.
- We can assign single letters as follows:

```
char letter = 'a';
```

  where we use single quotes.
- An array of `char`s is specified using double quotes:

```
char * name = "Steven";
```


Demo of char

```
#include <stdio.h>

int main()
{
   char * name = "David";
   printf(name); /* not recommended, but allowed*/
   printf("\nname    = %s\n", name);
   printf("name[0] = %c = %d\n", name[0], name[0]);
   return 0;
}
```

Gives the following output:

```
David
name    = David
name[0] = D = 68
```


Layout of a String in Memory

Given: `char * string = "A string in C!";` In memory this looks like:

%c	A		s	t	r	i	n	g		i	n		C	!	\0
%d	65	32	115	116	114	105	110	103	32	105	110	32	67	33	0

- All strings in C are terminated by 0 (or `'\0'`).
- `char` values of 0-127 correspond to *ASCII* codes, their use should be relatively consistent between different compilers.
- All other values correspond to *extended ASCII* codes, the representations of which vary considerably between compilers.

Some useful char Functions

For chars

`isalpha(c)`	True (non-zero) if c is from `A-Z,a-z`
`isdigit(c)`	True if c if from `0-9`
`isalnum(c)`	`=(isalpha(c) || isdigit(c))`
`islower(c)`	True if c is from `a-z`
`isupper(c)`	True if c is from `A-Z`
`d=tolower(c)`	Convert to lowercase (if `isupper(c)`), otherwise it returns c
`d=toupper(c)`	Convert to uppercase (if `islower(c)`), otherwise it returns c

Some useful String Functions

`strlen(s)`

Returns the number of characters pointed to by `s`, the trailing `NULL` (`'\0'`) is excluded!

`strncpy(dest, source, length)`

Copies a maximum of `length` characters from `source` to `dest`.

`int strncmp(s1, s2, length)`

Compares a maximum of `length` characters of `s1` and `s2`. Note that `strncmp` returns 0 (usually false) for equality!

A String Demo

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

int main()
{
   unsigned int loop;
   char * string = "A string in C!", * copy;
   printf("strlen(string) = %d\n", strlen(string));
   copy = (char *) calloc(strlen(string)+1, 1);
   if (!copy)
   {
      fprintf(stderr, "Couldn't allocate buffer!\n");
      return -1;
   }

   strncpy(copy, string, strlen(string));
   printf("strncmp(string, copy) = %d\n",
         strncmp(string, copy, strlen(string)));

   for (loop = 0; loop < strlen(copy); loop++)
      copy[loop] = toupper(copy[loop]);

   printf("modified copy = \"%s\"\n", copy);
   printf("strncmp(string, copy) = %d\n",
         strncmp(string, copy, strlen(string)));

   free(copy);
   return 0;
}
```


Results from String Demo

The program on the previous slide gives the following output:

```
strlen(string) = 14
strncmp(string, copy) = 0
modified copy = "A STRING IN C!"
strncmp(string, copy) = 32
```

- Note that `strncmp` returns `0` for *equality* and non-zero otherwise.
- Case insensitive string comparisons can be made using: `strnicmp`.

Handling the command-line in C

- So far, we have used the prototype: `int main(void)`.
- UNIX and Windows support command-line arguments to programs, and these need to be passed to main somehow.
- There is another prototype of `main` we are allowed to use:

```c
int main(int argc, char ** argv)
```

The example below prints out the command-line arguments to a program:

```c
#include <stdio.h>

int main(int argc, char ** argv)
{
   int loop;
   for (loop = 0; loop < argc; loop++)
      printf("argv[%d] = %s\n", loop, argv[loop]);
   return 0;
}
```

ASCII Table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	\0	SOH	STX	ETX	EOT	ENQ	ACK	\a	\b	\t	\n	VT	\f	\r	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	' '	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- Characters 0x00 to 0x1f (31) are *non-printable*.
- Characters 0x80 (128) to 0xff (255) are *extended characters*.
- We are going to have problems representing other languages using ASCII!

Internationalisation (i18n)

There are two methods that C employs to overcome the limitations of `char`:

Wide Characters & Wide Strings (used in Windows)

C has a wide character type called `wchar_t`, a capital L is used to denote a wide character or wide string:

```
wchar_t mwchar = L'a';
wchar_t * wideString = L"A wide string";
```

Multibyte Characters (used in Linux)

Another trick is to encode frequently used characters (such as numbers and Roman letters) using ASCII as normal, and for larger character sets (such as Hanzi) combine two or more `char`s together.

Unicode Sample

A sample to convert from wide to multibyte, then print

```
#include <stdio.h>
#include <wchar.h>
#include <stdlib.h>
#include <locale.h>

int main()
{
   wchar_t * unicodeString = L"a name:\x8FEA\x6587\nMaths symbols:"
                             L"x\x220A\x2115\x2204\x222B\x2202\n";
   char * mbString = NULL;
   size_t mbSize;
   /* select native locale */
   char * locale = setlocale( LC_CTYPE, "");
   printf("MB_CUR_MAX = %d\n", MB_CUR_MAX);
   printf("sizeof(wchar_t) = %d\n", sizeof(wchar_t));
   printf("locale = %s\n", locale);

   mbSize = (wcslen(unicodeString)+1)*MB_CUR_MAX;
   mbString = (char *) malloc(mbSize);
   if (!mbString)
   {
      fprintf(stderr, "Unable to allocate multibyte buffer");
      return -1;
   }

   wcstombs(mbString, unicodeString, mbSize);
   printf("Multibyte: %s\n", mbString);
   free(mbString);
   return 0;
}
```


Multibyte Output

- Logging in to a Linux box via PuTTY and using UTF-8 encoding, I get:

```
MB_CUR_MAX = 6
sizeof(wchar_t) = 4
locale = en_GB.UTF-8
Multibyte: a name:迪文
Maths symbols: x∈ℕ ∄∫ ∂
```

- This won't work properly for the Windows text console, as it doesn't allow for any unicode encodings. (If you set both the system and user locales to Chinese, then the Hanzi will display properly).

C Keywords

The following keywords are recognised by all C compilers as special commands. These words should not be used for variable names, function names etc.

```
auto      break     case      char
const     continue  default   do
double    else      enum      extern
float     for       goto      if
int       long      register  return
short     signed    sizeof    static
struct    switch    typedef   union
unsigned  void      volatile  while
```

- Exercise, look up the ones that have not been discussed in lectures.

C Keywords - Discussion

- `auto` is the complement of `static`, it is set by default thus seldom seen in code. *DO NOT USE* - in the latest C++ standard, `auto` has a new meaning.
- `const` specifies that a variable may not be changed once it's initialised. Can also be used in function prototypes to indicate read only arguments (i.e. read only arrays). Proper use of `const` is often referred to as *const correctness*.
- `enum` creates a unique type that assigns numerical values to symbolic names, e.g.

```
enum suit {clubs, diamonds, hearts,
           spades};
```


Preprocessor Keywords

- We also have the following preprocessor keywords:

```
#include  #define  #undef
#if       #ifdef   #ifndef
#elif     #else    #endif
#error    #line    #pragma
```

- The `#pragma` directive is "implementation specific".

Operator Precedence and Associativity

From K&R2:

Operators	Associativity
`() [] -> .`	left to right
`! ~ ++ -- + - * & (type) sizeof`	right to left
`* / %`	left to right
`+ -`	left to right
`<< >>`	left to right
`< <= > >=`	left to right
`== !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`?:`	right to left
`= += -= *= /= %= &= ^= |= <<= >>=`	right to left
`,`	left to right

Operator Precedence and Associativity - Examples

`a - b * c / d`

1. `*` and `/` are carried out before `-` due to precedence.
2. `*` is carried out before `/` due to (left to right) associativity.

`if (x & MASK == 0)`

- == has a higher precedence than & so is executed first!
- To get what we originally intended, parentheses are needed:

```
if ((x& MASK) == 0)
```

If in doubt

Put brackets around things...

More Pointers!

- So far, we have seen examples of pointers to:
  - native types (such as `int*`, `float*`, `double*`...),
  - something unknown (`void*`),
  - structures (such as `FILE*`),
  - and pointers themselves (such as `int**`, `double**`...).
- There is one more pointer type, a pointer to a *function*.

Why point to functions?

C code is often re-used between projects. Mathematical routines such as ones to find roots of functions, become extremely limited if they are tied down to a specific case.

Declaring a pointer to a function

One way is to use a `typedef`:

```
typedef double (* fx)(double x);
```


Function Pointers - An example

```
#include <stdio.h>
typedef double (* fx)(double x);

double newtonSolve(fx func, fx grad, double guess)
{
   unsigned int loop;
   for (loop = 0; loop < 10; loop++) guess -=func(guess)/grad(guess);
   return guess;
}

double sample(double x)
{
   return x*x-2.0;
}

double dsample(double x)
{
   return 2.0*x;
}

int main()
{
   double sol = newtonSolve(sample, dsample, 1.0);
   printf("Solution = %.15lg\n", sol);
   printf("Residue = %lg\n", sample(sol));
   return 0;
}
```

This should get you started with exercise # 5.

Don't Do it All Yourself!

- As you have seen from the exercises, writing functions to solve equations such as cubics can become complicated (especially when rounding error needs to be minimised).
- A lot of people have spent a considerable amount of time attempting to perfect implementations of mathematical functions.
- Routines written by a third party are packaged in *libraries*.

Using Fortran Code

- Good news! Scientific programming has been going on for over 60 years (counting from Colossus) in Britain.
- Unfortunately a lot of this has been carried out in Fortran.

f2c

Bell Labs have released a program called `f2c` which converts Fortran source code to C. The output from `f2c` is usually not very human friendly, but it does at least compile.

GNU Scientific Library - GSL

GNU have released their C Scientific Library:

`http://www.gnu.org/software/gsl/`

- It is managed by scientists at Los Alamos, and is very comprehensively documented.
- GSL requires gcc (for Windows users, Cygwin can be used).

Licensing

"GSL can be used internally ("in-house") without restriction, but only redistributed in other software that is under the GNU GPL."

GSL Example

```
#include <stdio.h>
#include <gsl/gsl_sf_bessel.h>

int main()
{
   double x = 0.0;

   printf("Enter x \n");
   scanf("%lf", &x);
   printf("J0(%g) = %.18e\n", x, gsl_sf_bessel_J0(x));

   return 0;
}
```

Compile this using the command-line:

```
gcc gsl-sample.c -lgsl -lgslcblas -lm -o gsl-sample
```


NAGLIB

The Numerical Algorithms Group, based in Oxford, maintain a software library called NAGLIB.

- Imperial College have signed a site license for NAGLIB.
- It can be installed on departmental machines upon request.
- You are allowed to install it on your home machines.

General Numerical Software

The Netlib Repository contains a lot of very useful numerical codes and papers. It is definitely worth having a route through their website:

`www.netlib.org`

JPEG Manipulation

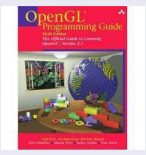
- The Independent JPEG Group, release a freeware JPEG (compressor/decompressor) library.
- Code is available from `http://www.ijg.org`; the most recent version as of writing is 8a (dated 28-Feb-2010).
- To use the code in Visual Studio one needs to rename some project files (please refer to `install.txt` for details).
- Example code is included that demonstrates the proper use of the library, for an example of decompression please refer to `djpeg.c`.

Graphics Programming in C

Avoid it if you can!

Most of the time scientific output can be plotted quite well by GNUplot, Maple and Matlab.

If you still want to...



I recommend that you program in OpenGL as it is portable across architectures and makes use of the graphics hardware in machines. I learnt OpenGL from the “Red Book”; OpenGL Programming Guide: The Official Guide to Learning OpenGL.

Example Code

I have placed an OpenGL sample on my website, it also conveniently demonstrates almost all the C we have covered in this course!

Calling Fortran from C - the C `main` function

```c
#include <stdio.h>

extern double ffunc01_(double *x);

double callfortran(double x)
{
   return ffunc01_(&x);
}

int main()
{
   double x = 1.0;

   printf("Hello from C!\n");
   printf("fsub(%g) = %g\n", x, callfortran(x));
   return 0;
}
```

Calling Fortran from C - The Fortran Function

```
function ffunc01(x)
   real(kind=8), intent(in) :: x
   real(kind=8)             :: ffunc01

   ffunc01 = 10*x + 1
   print *, 'In fortran, received x = ', x
   print *, 'returning', ffunc01
end function
```

- We explicitly specify `kind=8`, i.e. `double` precision.
- In `g95`, all exported functions have a trailing underscore.
- Fortran arguments to functions are pointers.

Calling C from Fortran - Fortran Interface

```
module cmodule
   implicit none
contains

   function my_func(x)
      real, intent(in) :: x
      real             :: my_func
      real(kind=8)     :: tmp

      interface
         function cfunc01(x)
            real(kind=8), intent(in) :: x
            real(kind=8)             :: cfunc01
         end function cfunc01
      end interface

      tmp = x

      my_func = cfunc01(tmp)
   end function my_func
end module cmodule
```


Calling C from Fortran - Fortran entry point

```
program test
   use cmodule
   implicit none

   real :: myx = 26.0980

   print *, 'myx = ', myx
   print *, 'my_funx(myx) = ', my_func(myx)
end program
```

- We must compile the `cmodule` *before* the entry point.
- This is due to Fortran .mod files (think auto-generated .h files).

Calling C from Fortran - The C function

```
#include <stdio.h>

double cfunc01_(double * x)
{
   double retval = (*x)*(*x) - 1.0;
   printf("In cfunc01(): received x = %g, returning: %g\n",
          *x, retval);
   return retval;
}
```

- The arguments to the c function are pointers.
- `cfunc01` is referenced from the `cmodule` interface.
- We add a trailing underscore to the function name.
- Function name translation in object files is known as *name mangling*.

Different compilers have different calling conventions!

Linking Fortran and C - Putting it all together

```
.SUFFIXES: .f90
CFLAGS = -pedantic -ggdb -Wall -ansi
FFLAGS = -ggdb
LFLAGS = -lm
CC = gcc
F90 = g95
CLEANFILES = fmain.o csub.o demo1 fmod.o cmodule.mod demo1.exe \
             cmain.o demo2 demo2.exe ffunc.o

all:    demo1 demo2

demo1:  cmain.o ffunc.o
        $(F90) cmain.o ffunc.o -o demo1

demo2:  fmod.o fmain.o csub.o
        $(F90) fmod.o fmain.o csub.o -o demo2

clean:
        touch $(CLEANFILES)
        rm $(CLEANFILES)

.f90.o:
        $(F90) $(FFLAGS) -c $*.f90
```


Running the codes

Running the first demo:

```
$ ./demo1
Hello from C!
 In fortran, received x =  1.  returning 11.
fsub(1) = 11
```

and the second demo:

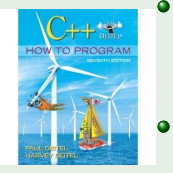
```
$ ./demo2
 myx =  26.098
In cfunc01(): received x = 26.098, returning: 680.106
 my_funx(myx) =  680.1056
```


Moving on to C++

What is it?

- C++ can be thought of very loosely as an object oriented extension to C.
- The C++ standard is over twice as big as the C standard.
- C++ is in the process of changing (at the time of writing C++0x is being adopted by compilers).

References

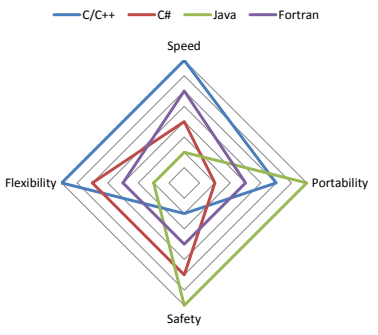


- One good book that has been brought to my attention is “C++ How to Program 7th Edition” by Harvey and Paul Deitel published in 2010.
- Before buying any C++ book, make sure that it is recent, and that you like the writing style.

Moving on to C# or Java?

Java and C# belong to a different class of language.

- Java and C#, both target *virtual machines*.
- Pointers are hidden from the user in C# and completely absent from Java.
- Most memory allocation and de-allocation is done automatically, via *garbage collection*.
- Java code can target multiple platforms, C# is supported only on Microsoft platforms.



C# References

Book

- “Pro C# 2008 and the .NET 3.5 Platform, Fourth Edition” by Andrew Troelsen, appears to be up-to-date and comprehensive.
- I would recommend you skim through a copy before buying. (.Net 4.0 is out now).

MSDN

The definitive (and free) source of information for Microsoft Platforms is the MSDN, the C# documentation is browseable at:

`http://msdn.microsoft.com/en-gb/library/kx37x362.aspx`