Today’s lecture

• Start C/C++
• Basic language features
C History and Background

• Origins 1973, Bell Labs
• Public K&R C “The C Programming Language”, [Kernighan 1978]
• ANSI C – standardized 1989, X3.159-1989
• Ritchie “C is quirky, flawed and an enormous success”
  – http://cm.bell-labs.com/cm/cs/who/dmr/chist.html
• Compiled language ( gcc, cc )
  – Good runtime performance, more control e.g memory utilisation
  – Portability, licensing, versatility
  – C apps: Matlab, Mathematica, + Linux netscape, IE, …
• C++ superset of C i.e. C plus some additional concepts – more on these later
C Variables (and C++)

• Variable names
  – Lower or upper case + lower, upper, digit, _ …
  – e.g. x, CO2, DENSITY, area_of_polygon
  – Names ARE case sensitive: CO2 and co2 not same
  – Keywords are reserved (also case sensitive)
    • if, for, while, return, int, float …….
Data types and basic arrays

- int, float, double, char, short, uint, long int
- int – 4 byte integer (long = 8 byte), short – 2 byte integer, float 32-bit, double 64-bit, char – 1 byte
- [] for arrays
- Examples
  - int a [10], b[10][10];
  - char c[20];
  - double x, area_of_circle, radius;
- Also macros
  - #define PI 3.14159
- Everything must be declared
- /* */ comments
Executable Statements 1

- Statement terminator is the ;. All C-statements end with this character (common compile error is to forget to put ; at end of a statement.
- Assignment
  - \#define PI 3.14159
  double x, radius, area_of_circle;
  radius=2.;
  area_of_circle = PI*radius*radius;
- Assignment operators:
  \textit{variable op= expression} is equivalent to \textit{Variable = variable op epression}
- Operators are: = += -= *= /= %= >>= <<= &= ^= |=
- Example: \textit{k *= 3+x} is the same as \textit{k=k*(3+x)}
- Some of the operators above (\texttt{>>} <\texttt{&} \texttt{|} are bit operators and rarely seen. \% is the modulus operator (\textit{a\%b} is a modulus b; remainder after removing as many b’s are possible from a e.g. \textit{7\%3 = 1})
- Multiple = and be used on a line e.g., \textit{a=b=c-0}; right to left evaluation
Executables: Conditionals

- Conditional statements are like fortran except no endif statement. The code to be executed is contained in {}'s unless it is just one statement.
  - if ( radius == 0. ) {
    inv_radius = 0.;
  } else {
    inv_radius = 1./radius;
  }
  - We could above used ‘} else inv_radius = 1./radius; ‘
  - If( radius == 0. ) { code }
    else if ( condition ) { code }
  - It is allowed to have to an empty statement by just having ; after the if or in a sequence of if else if statements.
Executable Statements 2

- Increment int type by 1 methods in C:
  - Postfix evaluated after expression
  - Prefix evaluated before expression
    ```
    int i;
    i = i+1.;
    ++i;    /* prefix mode */
    i++;    /* postfix mode */
    ```
  - When used in an expression prefix mode increments first e.g.,
    ```c = ++a + ++b;``` gives different answer to ```c = a++ + b++;```
  - These commands are used because increment by 1 is a
    machine instruction (faster than load 1 to register and add to
    another register)

- Changing variable type: cast
  - ```double x; int i;```  
  - ```x = (double) i; /* changes integer i to double type)```
Executable Statements 3

• Loops using the “for” construction.

```c
int i, j, k;
double b[10][10];
k=0;
for (j=0; j<10; ++j) {
    for (i=0; i<10; ++i) {
        b[j][i] = (double) k++;
    }
}
```

• Fortran style “do while structure” but the while appears at the end of the construction

```c
do { statements; } while (condition);
```
Standard libraries

• no math functions, no I/O functions etc are included in standard code. Header files are need to define constants and functions.

```c
#include <math.h>
x = cos(y);
z = cos(PI);
```

```c
#include <stdio.h>
printf("Hello\n");
fprintf(stdout,"Hello\n");
```

```
<math.h> == /usr/include/math.h – C source files
<stdio.h> == /usr/include/stdio.h
```
#include <stdio.h>
#include <math.h>
int i=1;
main()
{
    int j;
    j = 2;
    printf("Hello\n");
    fprintf(stdout,"Hello\n");
    fprintf(stdout,"pi == %f\n",M_PI);
    fprintf(stdout,"i == %d\n",i);
    fprintf(stdout,"j == %d\n",j);
}
Functions

• Definition method. All modules are functions in c and may or may not return a result (type void if no return).
  
  type fname(type arg1, type arg2)
  {
    /* Local variables and executable code */
  }

• Calling a function
  
  fname(arg1, arg2); /* type void call */
  result = fname( arg1, arg2); /* result and fname same type*/

• Prototype defines how a function should be called
  
  type fname(type, type);

• In C, none of the arguments passed to a functions can be changed -- call by value. Addresses can be passed and the values stored at these addresses can be changed.
Function Example

```c
int mymax(float, float);     /* Prototype */
main ()
{
    float a,b; int ans;
    a=b=2.;
    ans= mymax(a,b) /* returns 1 if a > b, 2 if b > a, 0 otherwise */
}
int mymax(float a, float b)
{
    if ( a > b ) return 1;
    if ( b > a ) return 2;
    return 0;
}
```
Call by reference

int mymax(*float, *float); /* Prototype. The *float is a pointer to (address of) a floating point number */
main ()
{
    float a,b; int ans;
    a=b=2.;
    ans= mymax(&a,&b); /* 1 if a > b, 2 if b > a, 0 otherwise */
    /* set a and b = to max. value */
}
int mymax(float *a, float *b)
{
    if ( *a > *b ) {*b=*a;return 1;}
    if ( *b > *a ) {*a=*b;return 2;}
    return 0;
}
Addresses - *, &

- C allows very explicit addressing of memory locations with the concept of “pointers” (points to memory location)

```c
short a; short *ptr_to_a;
a = 1;
ptr_to_a = &a;
```

Computer Memory

0x00 0xFF

&\(a\)  a (value stored at &\(a\))
Summary

• C programming language. Similar to fortran in many ways but with:
  – Somewhat less rigid syntax
  – More explicit memory addressing methods
  – “short-cut” ways of doing operations that can be very fast on some CPU’s.

• Next lecture we go into more detail in pointers and call by reference and call by value.