C History and Background

- Origins 1973, Bell Labs
- 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**

Do nothing
;

Assignment
- `#define PI 3.14159`
  
  double x, radius, area_of_circle;
  
  radius=2.;
  
  area_of_circle = PI*radius*radius;

Conditional
- if ( radius == 0. ) {
  
  inv_radius = 0.;
  
} else {
  
  inv_radius = 1./radius;
  
}

**Executable Statements 2**

postfix, prefix, infix

int i;

i = i+1.;

++i;

i++;  

cast

double x; int i;

x = (double) i;
Executable Statements 3

Loops

```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++;
    }
}
```

Standard libraries

no math functions, no I/O functions, ...

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

#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
```
A C Program

```c
#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
type fname(type arg1, type arg2)
{
/* Local variables and executable code */
}

Calling a function
fname(arg1, arg2);

Prototype
type fname(type, type);

Functions are call by value
Function Example

```c
int mymax(float a, float b); /* 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

```c
int mymax(float *a, float *b); /* Prototype */
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;
}
```
Adresses - *, &

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

Computer Memory

Compiling and linking

- Source code is created in a text editor.
- To compile and link:
  \[ cc \text{ <options>} \text{ prog.c} \text{funcs.c } -\text{libraries} \text{ -o prog} \]
  Where \text{prog.c} is main program plus maybe functions
  \text{funcs.c} are more subroutines and functions
  \text{libraries.a} are indexed libraries of subroutines and
  functions (see \text{ranlib})
  \text{prog} is name of executable program to run.
- \text{<options>} depend on specific machine (see \text{man cc} or \text{cc --help})
- -\text{libraries} refers to precompiled library in file \text{liblibraries.a}
C Basic Summary

- Origins of C – Compiled language; K&R, ANSI
  – V. versatile i.e Matlab, Mathematica, Fortran compiler,
    Linux, Netscape cores - all are mostly in C.
- Basic Syntax
  – case sensitive, semi-colon required at end of
    statements, loops, conditionals (==).
- Simple program
  – Standard libraries (stdio.h, math.h)
  – Calling a function
- Call by reference v. call by value
  – double a; double *ptrToA; ptrToA = &a;

C preprocessor (CPP)

- precompile macros and options; “compiler” proper does
  not see CPP code.
- Also stand alone cpp; other compilers e.g. .F files fortran –
  (not in java!)
- #include - file inclusion
- #define - macro definition
- #undef - undefine macro
- #line - compiler messages line number (not
  really for general use)
- #if, #ifdef, #ifndef, - Conditional compilation
- #else, #elif, #endif
- __FILE__, __LINE__ (ANSI C).
C preprocessor (CPP)

- Include "fred.h" - includes contents of file fred.h in program. -I cpp flag sets path to search for fred.h
- Define PI 3.14159 - substitutes 3.14159 everywhere PI occurs in program source. (except in quotes).
- Define PI - stops substitution
- If defined PI
  - printf("pi is set to %f in file %s\n",PI,__FILE__);
- Else
  - printf("pi is not set. Line %d file %s\n",__LINE__,__FILE__);
- Endif

Macros with args
#define _getaddress(a) (&a)
main() { double n; double *ptrToN;
  ptrToN = _getaddress(n); }
- Compiler proper sees
main() { double n; double *ptrToN;
  ptrToN = &n; }
- Often used for debugging
#ifdef debug
#define _D(a) a
#else
#define _D(a)
#endif

10/6/05 12.010 Lec 8 17
Structures and Types

• Way to group things that **belong** together
  – e.g. Representing 3d coord \((x,y,z)\)
  – No structures
    
    ```
    double cx, cy, cz;
    cx=3.;cy=3.;cz=2;
    plot(cx, cy, cz);
    ```

  – Structure
    
    ```
    struct { double cx; double cy; double cz; } point;
    point.cx = 3.; point.cy=3.;point.cz=2.;
    ```

---

Structures and Types

• Struct alone is still unclear - typedef
  
  ```
  typedef struct { double cx;
                 double cy;
                 double cz; } t_point;
  ```

  ```
  main() {
    t_point point;
    point.cx = 3.; point.cy=3.; point.cz=2.;
    plot(point);
  }
  ```
Structures and Types

- Derived types just like basic types
  - e.g. can use arrays
- typedef struct { double cx;
    double cy;
    double cz; } t_point;

```c
main() {
    t_point point[10]; int i;
    for (i=0;i<10;++i) {
        point[i].cx = 3.; point[i].cy=3.; point[i].cz=(double)i; }
    for (i=0;i<10;++i) {
        plot(point[i]); }
}
```

10/6/05 12.010 Lec 8 21

Memory Management

- Application code creates variables and arrays at runtime
- `<stdlib.h>` - malloc, calloc, free, realloc + sizeof
- e.g
```c
main(int argc, char *argv[]) {
    double *foo; int nel; int i;
    /* Create an array of size nel at runtime */
    sscanf(argv[1], "%d\n", &nel);
    foo = (double *) calloc(nel,sizeof(*foo));
    if ( foo == NULL ) exit(-1);
    for (i=0;i<nel;++i) { foo[i]=i; }
    free(foo);
}
```

10/6/05 12.010 Lec 8 22
Remember - *, &

Here compiler allocated memory for you

Here application allocates memory explicitly.
 Allows more control but requires careful bookeeping.

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

foo = (double *) malloc(3, sizeof(*foo));