12.010 Computational Methods of Scientific Programming
Fall 2008

12.010 Computational Methods of Scientific Programming
Lecture 9

Today's lecture

- C in more detail

Summary

- LAST LECTURE
- Basic C
  - Syntax v. Fortran
- THIS LECTURE
  - Examined C-pointers
  - File Input/Output and the routines for formatted reads and writes
  - Compiling C routines
  - The C preprocessor cpp.
  - Structures in C
  - Memory management

Call by reference

- In call by reference, the address of a variable (called a pointer) is passed to the function. The value stored at this address can be changed but not the address itself (arguments to C functions can never be changed).
- Example:

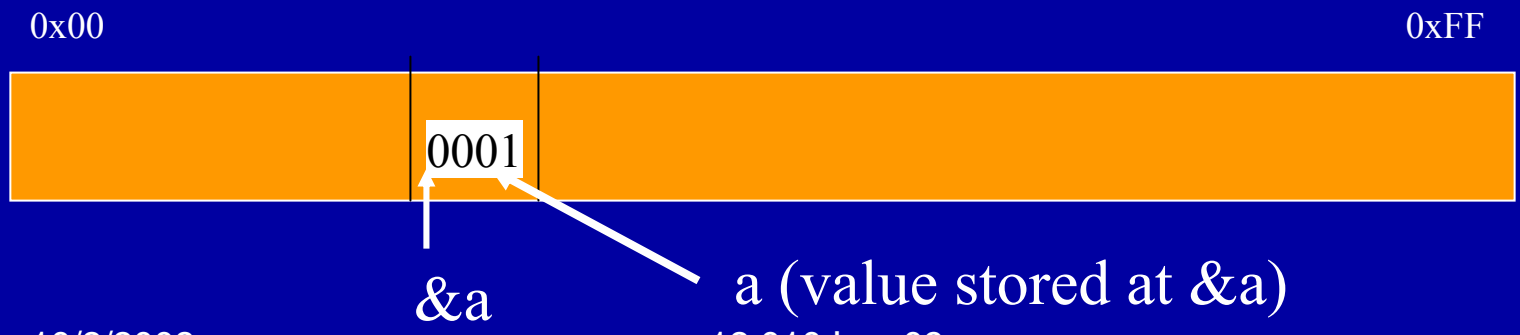
```
int mymax(*float, *float);     /* Prototype.  The *float is a pointer to (address of)
     a floating point number */
main ()
{
   float a,b; int ans;
   a=b=2.;
   ans= mymax(&a,&b); /* 1 if a > b, 2 if b > a, 0 otherwise */
                                 /* set a and b = to max. value       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

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

Computer Memory



Example of pointer use

- The following code examines how pointers can be used.

```
main ()
{
  char c='A', *p, s[100], *strcpy();
  p = &c ;
  printf("\n%c %c %c", *p, *p+1, *p+2);
  s[0] = 'A' ; s[1] = 'B'; s[2] = 'C'; s[3] = '\0';
  p = s;
  printf("\n%s %s %c %s",s, p, *(p+1), p+1);
  strcpy(s,"\nshe sells seas shells by the seashore");
  printf("%s",s);
  p += 17;
  for ( ; *p != '\0' ; ++p ){
      if ( *p == 'e' ) *p = 'E';
      if ( *p == ' ' ) *p = '\n';
  }
  printf("%s\n",s);
}
```

Output of Program

```
A B C
ABC ABC B BC
she sells seas shells by the seashore
she sells seas shElls
by
thE
sEashorE
```

File input/output

- To use files in C, the stdio.h header needs to be included. This contains a structure called FILE.
- Code for file use contains
  FILE *fp, *fopen();
  fp = fopen("file name","r");
- fp will return NULL if file could not be opened.
- The options for open are "r" read; "w" write; "a" append
- The file name is a variable would be declared
  char file_name[100];
- With stdio.h included, stdin stdout and stderr are pointers to the keyboard, screen and error output (direct output to screen with little or no buffering).
- fclose(fp) will close the file (needed if written in one part of program and read in another). Automatically happens when program stops.

Reading/writing files

- To read files:
  - getc(fp) : Gets next character in file
  - fgetc(fp) : Same but function not macro
  - getchar() : Similar but reads from stdin
  - fgets(s,n,fp) : Gets string of n-1 characters or until a newline character is read (\n)
  - gets(s) : Similar but reads from stdin
  - putc(c,fp) : Outputs a character (putchar to stdout)
  - fputs(s, fp) : null terminated string sent to file. (puts goes to stdout).
- fseek and other functions allow more control of moving through file.

Reading/writing

- The main reading/writing routines are:
  printf, fprintf, sprintf : Output formatted lines to stdout, a file pointer and string
  scanf, fscanf, sscanf : Input formatted lines stdin, a file pointter or a string.
- Format used:
  - %nc - prints character in n-width right justified; %-nc is left justified.
  - %n.ms - n character string into m width right justfied, %-n.ms is left justified, %s whole string to \0
  - %n.md int ouput (%-n.md left justified)
  - %n.mf floating point
  - %n.me exponential format
  - Others include o for octal, x for hexidecimal, g for e/f combination

Compiling and linking

- Source code is created in a text editor.
- To compile and link:

  cc <options> prog.c funcs.c -llibraries -o prog

Where prog.c is main program plus maybe functions

  funcs.c are more subroutines and functions

  libraries.a are indexed libraries of subroutines and functions (see ranlib)

  prog is name of executable program to run.

- <options> depend on specific machine (see man cc or cc --help)
- -llibraries refers to precompiled library in file `lib`libraries`.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).
- #undef PI - stops substitution

```
#ifdef 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
```

C preprocessor (CPP)

- Macros with args

```
#define _getaddress(a) (&a)  /* This macro returns address of a */
main() {   double n; double *ptrToN;
              ptrToN = _getadress(n); }
```

- Compiler actually sees code below

```
main()  {   double n; double *ptrToN;
              ptrToN = &n; }
```

- Often used for debuging

```
#ifdef debug
 #define _D(a) a
#else
 #define _D(a)
#endif
```

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;

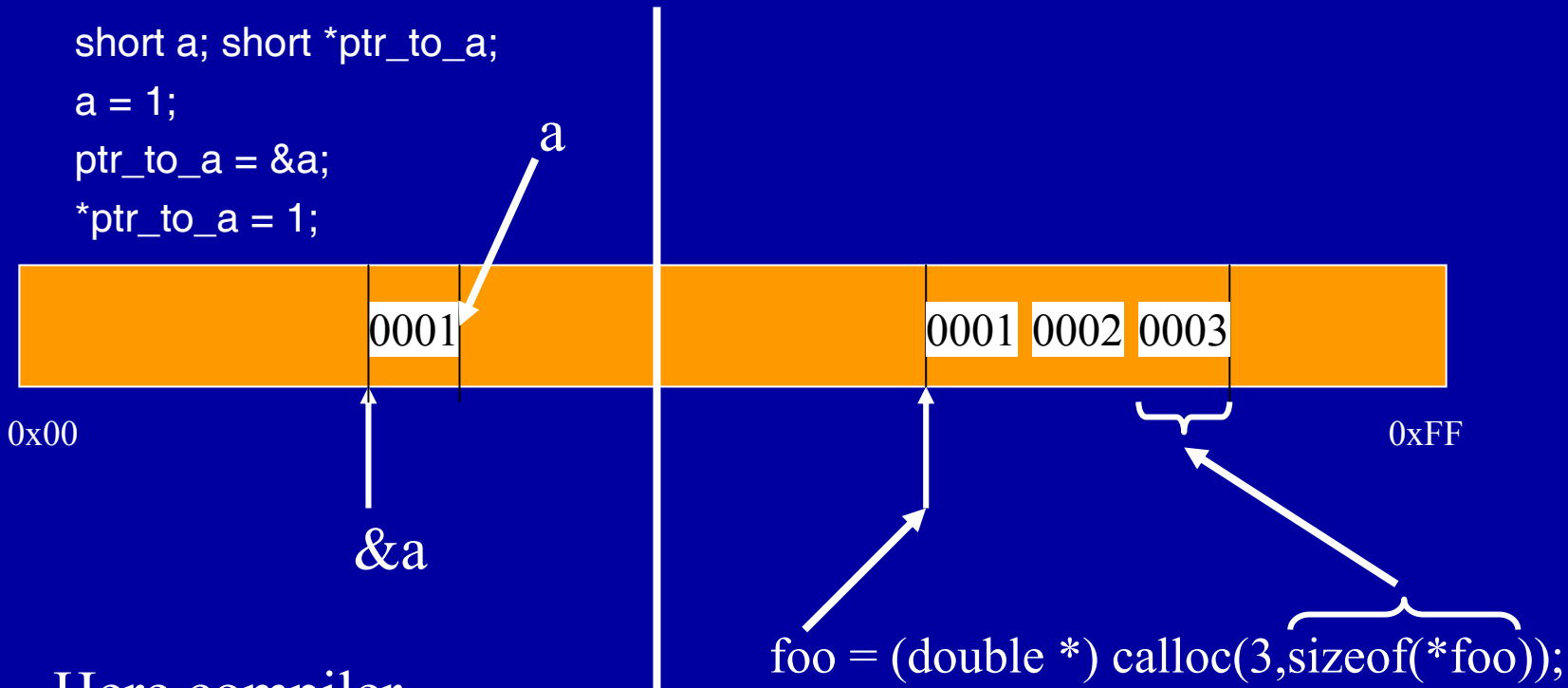
```
main() {
 t_point point[10]; int i;
for (i=0;i<10;++i) {
 point[i].cx = 3.; point[i].cy=3.; point[i].cz=(double)i; }
for (i=0;i<10;++i) {
 plot(point[i]); }
}
```

Memory Management

- Application code creates variables and arrays at runtime
- <stdlib.h> - malloc, calloc, free, realloc + sizeof
- e.g

```
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);
}
```

Remember - *, &



Here compiler allocated memory for you

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

Summary

- Examined C-pointers
- File Input/Output and the routines for formatted reads and writes
- Compiling C routines
- The C preprocessor cpp.
- Structures in C
- Memory management