Practical C Issues:

Preprocessor Directives, Typedefs, Multi-file Development, and Makefiles

Jonathan Misurda
jmisurda@cs.pitt.edu

#include

- Copies the contents of the specified file into the current file
- < > indicate to look in a known location for includes
- " " indicate to look in the current directory or specified path

```
#include <stdio.h>
#include "myheader.h"
```

#define

- **Textual** Symbol Replacements

```
#define PI 3.1415926535
#define MAX 10

float f = PI;
for(i=0;i<MAX;i++) …
```

#define Macros

- Textual replacements with parameters:
- Good:
  - `#define MAX(a, b) (a > b) ? a : b`
- Not so good:
  - `#define SWAP(a,b) {int t=a; a=b; b=t;}`

#if

- #if <condition that can be evaluated by the preprocessor>
- What does preprocessor know?
  - Values of #defined variables
  - Constants

Example

```
#include <stdio.h>

int main()
{
  #if 0
     printf("this is not printed\n");
  #endif
  printf("This is printed\n");
  return 0;
}
```

Example 2

```
#include <stdio.h>
#define VERSION 5

int main()
{
  #if VERSION < 5
     printf(“this is not printed\n”);
  #endif
  printf(“This is printed\n”);
  return 0;
}
```

#else

```
#if
  …
#elif
  …
#else
  …
#endif
```

#ifdef

- #if defined
  - Checks to see if a macro has been defined, but doesn’t care about the value
  - A defined macro might expand to nothing, but is still considered defined

Example

```
#include <stdio.h>
#define MACRO

int main()
{
  #if defined MACRO
     printf(“this is printed\n”);
  #endif
  printf(“This is also printed\n”);
  return 0;
}
```

#undef

- Undefines a macro:

```
#include <stdio.h>
#define MACRO
#undef MACRO

int main()
{
  #if defined MACRO
     printf(“this is not printed\n”);
  #endif
  printf(“This is printed\n”);
  return 0;
}
```

Shortcuts

- #if defined → #ifdef
- #if !defined → #ifndef

Uses

- Handle Operating System/Architecture specific code
- Handle differences in compilers
- Build program with different features
  - Debugging:

```
#ifdef DEBUG
   printf(…)
#endif
```

Notes

- Can define variables from the commandline with –D
  - `gcc –o test –DVERSION=5 test.c`
  - `gcc –o test –DMACRO test.c`

Other Preprocessor Details

- # - quotes a string
- ## - concatenates two things
- #pragma
- #warn
- #error

Defined Constants

Macro	Meaning
__FILE__	The currently compiled file
__LINE__	The current line number
__DATE__	The current date
__TIME__	The current time
__STDC__	Defined if compiler supports ANSI C
…	Many other compiler-specific flags

typedef

```
typedef type-declaration synonym;
```

Examples:

```
typedef int * int_pointer;
typedef int * int_array;
```

Type Clarity

```
void takes_int(int_pointer x)
{
   *x = 3;
}
```

```
void takes_array(int_array x,
   int n)
{
   int i;
   for(i=0; i<n; i++)
        printf(“%d\n”, x[i]);
}
```

Structures

Typedef

```
typedef struct node {
  int i;
  struct node *next;
} Node;

Node *head;
```

Struct with Instance

```
struct node {
  int i;
  struct node *next;
} Node;
```

Function Pointers

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

typedef void (*FP)(int, int);

void f(int a, int b) {
   printf("%d\n", a+b)
}

void g(int a, int b) {
   printf("%d\n", a*b)
}

int main() {
        FP ar1 = f;
        FP ar2 = g;

        ar1(2,3);
        ar2(2,3);
        return 0;
}
```

Function Pointers As Parameters

```
void qsort (
  void *base ,
  size_t num ,
  size_t size ,
  int (*comparator)(const void *, const void *)
);
```

Comparator

```
int compare_ints(const void *a,const void
  *b)
{
  int *x = (int *)a;
  int *y = (int *)b;

  return *x - *y;
}
```

Multi-file Development

- Want to break up a program into multiple files
  - Easier to maintain
  - Multiple authors
  - Quicker compilation
  - Modularity

Static Local Scope

- Scope: **Local**
- Lifetime: **"Global"** (life of program)

```
void f(…) {
  static int x;
  …
}
```

File Scope

- “Global Variables” are actually limited to the file
- extern maybe be used to import variables from other files

File A int x;

File B extern int x;

$$\underbrace{\texttt{int x;} \qquad\qquad \texttt{extern int x;}}_{\text{Will refer to the same memory location}}$$

Example

a.c

```c
int x = 0;

int f(int y)
{
        return x+y;
}
```

b.c

```c
#include <stdio.h>

extern int x;
int f(int);

int main()
{
        x = 5;
        printf("%d", f(0));

        return 0;
}
```

Compiling

```
gcc a.c b.c

./a.out
5
```

Static

a.c

```c
static int x = 0;

static int f(int y)
{
        return x+y;
}
```

b.c

```c
#include <stdio.h>

extern int x;
int f(int);

int main()
{
        x = 5;
        printf("%d", f(0));

        return 0;
}
```

Compiling

```
gcc a.c b.c

/tmp/cccyUCUA.o(.text+0x6): In
  function `main':
: undefined reference to `x'
/tmp/cccyUCUA.o(.text+0x19): In
  function `main':
: undefined reference to `f'
collect2: ld returned 1 exit status
```

Header Files

- Usually only contain declarations
  - Variables
  - Functions
  - #defined macros
- Paired with an implementation file

Including a Header File Once

```
#ifndef _MYHEADER_H_
#define _MYHEADER_H_

…Definitions of header to only be included once


#endif
```

Headers and Implementation

mymalloc.h

```
void *my_nextfit_malloc(int size);

void my_free(void *ptr);
```

mymalloc.c

```
static MallocInfo *head;

void *my_nextfit_malloc(int size){
   static MallocInfo *current;
   ...
}

void my_free(void *ptr) {
   ...
}
```

Driver

- Driver program:

  #include "mymalloc.h"

- Can now use those functions
- Compile:

```
gcc –o malloctest mymalloc.c mallocdriver.c
```

Makefiles

- Express what files depend upon others
- If any are modified, build smallest set required

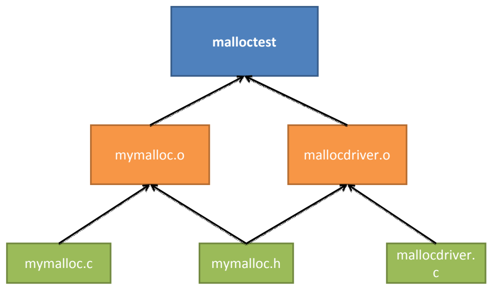
Makefile

```
malloctest: mymalloc.o mallocdriver.o
   gcc –o malloctest mymalloc.o mallocdriver.o

mymalloc.o: mymalloc.c mymalloc.h
   gcc –c mymalloc.c

mallocdriver.o: mymalloc.h mallocdriver.c
   gcc –c mallocdriver.c
```

Dependency Graph


malloctest
mymalloc.o
mallocdriver.o
mymalloc.c
mymalloc.h
mallocdriver.c