Typedefs, Multi-file Development, and Makefiles

Jonathan Misurda
jmisurda@cs.pitt.edu

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;
```

Struct with Instance

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

```
Node *head;
```

Function Pointers

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 3

typedef void (*FP)(int);

void f(int a) { printf("hello world\n"); }
void g(int a) { printf("%d\n", a); }
void h(int a) { printf("The value is %d\n", 2 * a); }

int main() {
        int i;
        FP ar[3] = {f, g, h};        //void (*ar[SIZE])(int) = {f, g, h};

        for(i=0;i<SIZE;i++)
                ar[i](5);

        return 0;
}
```

Multi-file Development

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

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;
```

Will refer to the same memory location

Example

a.c

```
int x = 0;

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

b.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

```
static int x = 0;

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

b.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

Headers and Implementation

mymalloc.h

```
void *my_bestfit_malloc(int size);

void *my_nextfit_malloc(int size);

void my_free(void *ptr);
```

mymalloc.c

```
static MallocInfo *head;

void *my_bestfit_malloc(int size){
   ...
}

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