

Guide to C



Multifile projects

Getting it running

Data types

Typecasting

Memory management

Pointers

CS-343 Operating Systems

Multifile Projects

- Give your project a structure
- Modularized design
- Reuse whole modules vs. copy of code
- The longer a file gets the less readable it is
- Static variables accessible only within module

hello.h: The Module Declaration

```
#ifndef __HELLO_H__
#define __HELLO_H__

    /*****Defines and Typedefs*****/
#undef EXTERN
#ifdef __HELLO_IMPL__
#define EXTERN
#else
#define EXTERN extern
#endif

    /*****Function Prototypes*****/
/*****
 * Title: Prints some hello world messages
 * -----
 * Purpose: Greets the whole world.
 * Input: void
 * Output: void
 *****/
EXTERN void hello();

#endif /* __HELLO_H__ */
```

hello.c: The Module Implementation

```
#define __HELLO_IMPL__

/*****System include*****/
#include <stdio.h>

/*****Private include*****/
#include "hello.h"

/*****Defines and Typedefs*****/
/* #defines and typedefs should have their names in all caps.
 * Global variables begin with g. Global constants with k. Local
 * variables should be in all lower case. When initializing
 * structures and arrays, line everything up in neat columns.
 */
#define MESSAGE      "hello world!"

/*****Implementation*****/

void hello()
{
    printf("%s\n", MESSAGE);
}
```

main.c: The Main Function

```
    /*******Private include*****  
#include "hello.h"  
  
    /*******Implementation*****  
  
int main(int argc, char* argv[])  
{  
    hello();  
  
    // exit with exit code 0  
    return 0;  
}
```

Makefile: Compile the project

```
TARGET=hello
SOURCES= hello.c \
        main.c

CC = gcc
CFLAGS = -g -Wall -UHAVE_CONFIG_H -D_GNU_SOURCE

$(TARGET): $(SOURCES)
$(CC) $(CFLAGS) -o $@ $^

debug: $(TARGET)
gdb $(TARGET)

clean:
rm -f $(TARGET)
rm -f *.o
rm -f *~
```

Console Output

```
[[sempi@sempi hello]$ make clean
rm -f hello
rm -f *.o
rm -f *~
[[sempi@sempi hello]$ ls
CVS/  hello.c  hello.h  main.c  Makefile
[[sempi@sempi hello]$ make
gcc -g -Wall -UHAVE_CONFIG_H -D_GNU_SOURCE -o hello hello.c main.c
[[sempi@sempi hello]$ ls
CVS/  hello*  hello.c  hello.h  main.c  Makefile
[[sempi@sempi hello]$ ./hello
hello world!
[[sempi@sempi hello]
```

Date Types

- Basic data types (char, int, float, double)
- Qualifliers (short, long)
`long int counter;`
- **See** `<limits.h>` and `<floats.h>`
- **Arrays**

```
int matrix[1000];  
access: matrix[0] ... matrix[999]
```


Advanced Data Types

- Structures (example from 'The C Programming Language')

```
struct point {  
    int x;  
    int y;  
}
```

...

```
struct point pt;
```

```
struct point maxpt = { 320, 200 };
```

```
printf("%d,%d", pt.x, pt.y);
```

Advanced Data Types (...)

Example from 'The C Programming Language'

```
/* makepoint: make a point from x and y components */
struct point makepoint(int x, int y)
{
    struct point temp;

    temp.x = x;
    temp.y = y;
    return temp;
}
```

Memory Management

- **Allocate memory** `<stdlib.h>`
 - Deallocate what you have allocated
 - DON'T rely on the cleanup at program termination
- **Malloc**
`void *malloc(size_t n)`
- **Calloc (for arrays)**
`void *calloc(size_t n, size_t size)`
- **Free**
`void free(void *ptr)`

Typcasting

- Use with care
 - a good source for bugs in your code
 - if you can avoid casts, AVOID IT!

- Example

```
int *ip
```

```
ip = (int *) calloc(n, sizeof(int));
```

```
free(ip); // free the allocated memory
```

Pointers

- Use with care
 - a good source of bugs
 - sometimes you can't avoid it
- Example ('The C Programming Language')

```
int x = 1; y = 2; z[10];
int ip;      /* ip is a pointer to int */

ip = &x;      /* ip now points to x */
y = *ip;      /* y is now 1 */
*ip = 0;      /* x is now 0 */
ip = &z[0];    /* ip now points to z[0] */
```

Strings

- String constant

`"This is a string"`

- Example

```
char *str;
```

```
str = "Hello, world\n"
```

- A string constant is always terminated with `\0`.
 - str from above points to some place in the memory, where you find:
Hello, world\n\0

Strings

- Array vs. pointers

```
char astr[] = "Hello";
```

```
char *pstr = "Hello";
```

- Array refers always to the same storage (content may change).
- The pointer is initialized to point to a location, where you find the string constant, but the pointer may be modified to point elsewhere.

Tools

- **valgrind**
 - memory allocation/deallocation check
 - finds memory leaks
- **lint (splint)**
 - code check

Bookshelf

- B. Kernighan and D. Ritchie; The C Programming Language, 2nd Edition: 1988
 - This book doesn't teach you programming, but it teaches you C!
- The Internet
 - You learn the most by looking at other people's code (you benefit from good and bad examples).