



Correct C Programs

Lawrence Angrave

Lecture 5

Overview

Compile, make, link & run

Importance of writing correct programs

Debug

Summary

Administrative

Reminder:

Turn in your MP0 before Monday 9am

Multiple submits OK

SMP0 Quiz

Comment: Check out R&R Appendix A

UNIX Man Pages

Compilation

Header Files

Linking and libraries

Macros and conditional compilation

Makefiles

Debugging Aids – lint, debugger, truss

Identifiers, Storage Classes and Linkage Classes

The C compiler

```
cc [filename.c]
```

Options include:

- o [output file name]
- lm to include the maths library
- E to get output from preprocessor only (no compilation).

The compiler works in three stages:

1. it **preprocesses** (replacing the #include, #define etc);
2. it **compiles** the preprocessed files;
3. it **links** all of the pieces of code together into the executable program.

For a large program you can compile a small piece at a time, and link it to larger, pre-compiled parts. To compile only, use:

```
cc -c [filename.c]
```

This creates **object files** (whose names end in .o), which can then be linked later by using them instead of the source files.

C Compiler: Libraries

It is possible to create **libraries**, which are (generally large) collections of useful object code ready for linking, so they don't need to be compiled over and over again for different programs.

`ar` = archive

`ruv` = **replace, update, verbose**

To do this with a set of .o files,

`ar`

`ruv`

`g_lib.a`

`g fopen.o g fclose.o g calloc.o.....`

This makes a library file called `g_lib.a`.

`g_lib.a` = output archive file name

object files to go in the library

Follow this by `ranlib g_lib.a`

which organizes the file in a form that is useful for the linker.

To make the functions available to other programs, add the library to the list of compiled files:

```
cc -o pgm main.c file1.c g_lib.a
```

Conditional compilation

The preprocessor can be used to exclude code from compilation:

```
#define DEBUG 1 /*set debug to true at top of file*/
.....
#if DEBUG
    printf("debug: a = %d\n", a);
#endif
```

Because **DEBUG** has value 1, the **printf** statements will be compiled. The statements can be turned off by setting **DEBUG = 0** at the start.

An alternative is

```
#ifdef DEBUG
.....
#endif
```

which will include the lines if **DEBUG** is defined at all at the top.

There is also an **#ifndef**, which includes lines if a name is not defined. There is also an **#undef** function to remove previous definitions, in order to prevent clashes.

To match the **#if**, there is an **#else**; there is also **#elif**, which is short for else if.

Preprocessor macros with arguments

```
#define SQ(x) ((x) * (x))
```

will replace, e.g., **SQ(a + b)** by **((a + b) * (a + b))** .
Note here that all of the parentheses are needed!

```
#define min(x, y) (((x) < (y)) ? (x) : (y))
```

will replace an expression such as

```
m = min(u, v)
```

by

```
m = (((u) < (v)) ? (u) : (v))
```

The arguments of **m** can be arbitrary expressions of comparable type.

Writing large programs

A large program will normally be stored as a collection of .h and .c files in its own directory.

If we are developing a program called *pgm*, we make a header file *pgm.h*, which contains **#includes**, **#defines** and a list of **function prototypes**.

All .c files should then have

```
#include "pgm.h"
```

at the top.

Note the use of quotes rather than in < >, when you include your own header files. The compiler then looks in the current directory rather than in the directory for standard C header files.

Header files in large programs

Often the same header files are included in several different source code files. How to avoid multiple definitions when the header file is compiled repeatedly?

Let us assume we have a header file called MyHeaderFile1.h.:

```
#ifndef MYHEADERFILE1
#define MYHEADERFILE1

.... function prototypes;
.... definitions;

#endif
```

this is only compiled if MYHEADERFILE1 is not defined (i.e. if the header file has not been processed before)

Thus, the material in the file is *only* included if it hasn't been included already.

Program Correctness

You thought that you implement the program correct, but it crashes. Why?

A common problem: **memory**

Do you allocate any buffer?

Do you allocate enough space?

Do you free it only once?



Segmentation Violations and Bus Errors

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

```
main()
```

```
{
```

```
    char *s;
```

(1) What will it happen?

(2) Why?

```
    s = NULL;
```

```
    printf("%d\n", s[0]);
```

```
}
```

Importance of Writing Correct Programs

Buffer Overflow

```
char buf[80];  
  
printf("Enter your first name");  
scanf("%s", buf);
```

Any problem?

Problem: if the user enters more than 79 bytes, the resulting string and the string terminator \0 do not fit in the allocated variable!!

Possible Fix of the Problem?

```
char buf[80];  
  
printf("Enter your first name");  
scanf("%79s", buf);
```

.

Another Example

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

int checkpass(void){
    int x;
    char a[9];
    x = 0;
    fprintf(stderr,"a at %p and\nx at %p\n", (void *)a, (void *)&x);
    printf("Enter a short word: ");
    scanf("%s", a);
    if (strcmp(a, "mypass") == 0)
        x = 1;
    return x;
}
```

What can be the problem?

Library Function Calls

Traditional UNIX – returns 0 successful, -1 unsuccessful, sets errno

POSIX Standard Committee decides that no 'errno' be used. Instead:

all new functions return error code

Good coders handle ALL errors, not just mandatory (in the standard) ones.

Error reporting

perror function outputs to standard error a message corresponding to the current value of *errno*

No return values of errors are defined by *perror*

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

```
void perror(const char * s);
```

strerror function returns a pointer to the system error message corresponding to the error code *errnum*

If successful, *strerror* returns a pointer to the error string

```
#include <string.h>
```

```
char *strerror(int errnum);
```

Handling Errors

Always handle all errors

Either:

- Print error message and exit program (only in main)

- Return -1 or NULL and set an error indicator such as *errno*

- Return an error code

All functions should report errors to calling program

Use conditional compilation to enclose debugging print statements

cc -DDEBUG

Debug Your program

- Discussion: It crashes, what can I do?

- **Tip1: Code review by yourself and others**

- Do you know the problem is?

```
char w[16];
char document[4096];
char docFname[128];
File *fp;
int main (int argc, char **argv ){
    //parsing parameters
    ...
    //reading document
    ...

    for(i=0; i<=docLength; i++){
        if(!strcmp(&(document[i]), w)){
            printf("found!\n"); return;
        }
    }
}
```

Tip2: Print Out More Debugging Info

```
...
int main (int argc, char **argv ){
    //parsing parameters
    ...
    //reading document
    ...

    printf("w=%s, docLength=%d", w, docLength);
    for(i=0; i<=docLength; i++){
        printf("i=%d, document[i]=%c\n", I, document[i]);
        if(!strcmp(&(document[i]), w)){
            printf("found!\n"); return;
        }
    }
}
```

Tip3: Try Different Inputs

Test case 1: the document contains the word

Test case 1a: the document contains the word in the very beginning

Test case 1b: the document contains the word in the very end

Test case 2: the document does not contain the word

Further reduce the document size to see if it changes

Tip4: Interactive Debuggers

Professional engineers use interactive debuggers
(gdb, Microsoft visual studio, etc)

gdb [GoogleSearch w document1](#)

You can stop the program in the middle, examine the state
(variable values, etc)

You can step one statement by one statement

You can change the variable values

Summary

Writing correct programs in C

Program layout

Read R&R : pp: 16-48, pp. 812-814