Australian National University

Structure of a c program

Eric McCreath

Basic structure

The basic structure of a c program includes:

- comments
- includes
- prototypes
- global variables
- constants
- functions
- main function

Comments

Comments provide a way for those reading your code to understand the codes: context, purpose, structure, and explain aspects of the code which may be hard to understand.

At a minimum it is always good practice to give source code a name, description, and authorship details.

Comments can be included:

- within `/* */` (they don't nest, so if you have a comment within a comment you can run into problems) or
- everything on a line after //

```
/* MyCoolProgram - reduces the temperature in the room
   Eric McCreath 2013 */

float currentTemp;  // stores the current temperature
```

includes

Includes can be thought of as simply copying the content of a header file into your source code. They shouldn't generate any code, but rather provide prototypes, struct, constants, and names for "extern"ally accessable global variables.

You will often include the header files of libraries you make use of. Note the library itself is "linked" in after your code is compiled.

For system headers you use <> e.g.

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

For local headers you use "" e.g.

```
#include "myheader.h"
```

prototypes

Prototypes provide the signatures of the functions you will use within your code. They are somewhat optional, if you can order your code such that you only use functions that are previously defined then you can get away without defining them.

Below a prototype for a function that sums two integers is given.

```
int add(int a, int b);
```

global variables

Variables that are defined outside of functions are global and can be access from any part of your code.

It is always good practice to minimize the scope of any variable. Hence care should be taken when using global variables.

Below is an example global variable which is an integer:

```
int count;
```

constants

A "#define", which are basically macro replacement rules, can be used to define constants within your code.

```
#define LOOPCOUNT 10
```

If you have literals that you use multiple times then you should replace them with a constant.

A common style that is used with *c* is to have all upper case for constant names.

functions

Functions are defined with their signatures followed by the code within curly brackets.

Below is the example of a function that adds two numbers.

```
int add(int a, int b) {
   int sum;
   sum = a + b;
   return sum;
}
```

The statements in *c* are ended by a ";'.

main function

Every program has a single "main" function. This is the function that is executed when the program starts running.

In c the parameters of the executed program are provided as parameters to the main function.

```
int main(int argc, char *argv[]) {
   // code of the main function
}
```

- argc - is the number of arguments given to the program when it is executed.
- argv - is an array of strings that references these arguments.

Note the first one in this list will be the name of the program (something that the executing code may not know).

Good Style

The below program is valid c, yet, it not only has poor style it is obfuscated code! see http://en.wikipedia.org/wiki/Obfuscation_%28software%29

```
/* LEAST LIKELY TO COMPILE SUCCESSFULLY:
  Ian Phillipps, Cambridge Consultants Ltd., Cambridge, England */
#include <stdio.h>
main(t,_,a)
char *a; {
        return!
0<t?t<3?main(-79,-13,a+main(-87,1-_,main(-86, 0, a+1 )+a)):1,
t<_?main(t+1, _, a ):3,main ( -94, -27+t, a )
&&t == 2 ?_<13 ?main ( 2, _+1, "%s %d %d\n" ):9:16:
t<0?t<-72?main( _, t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+,/+#n+,/#;\
#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l q#'+d'K#!/+k#;\
q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# ){nl]!/n{n#'; \
r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#\
\
n'wk nw' iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c ;;\
{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;\
#'rdq#w! nr'/ ') }+}{rl#'{n' ')# }'+}##(!!/")
:t<-50?_==*a ?putchar(31[a]):main(-65,_,a+1):main((*a == '/') + t, _, a + 1 )
:0<t?main ( 2, 2 , "%s"):*a=='/'||
main(0,main(-61,*a, "!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghir
,a+1);}
```

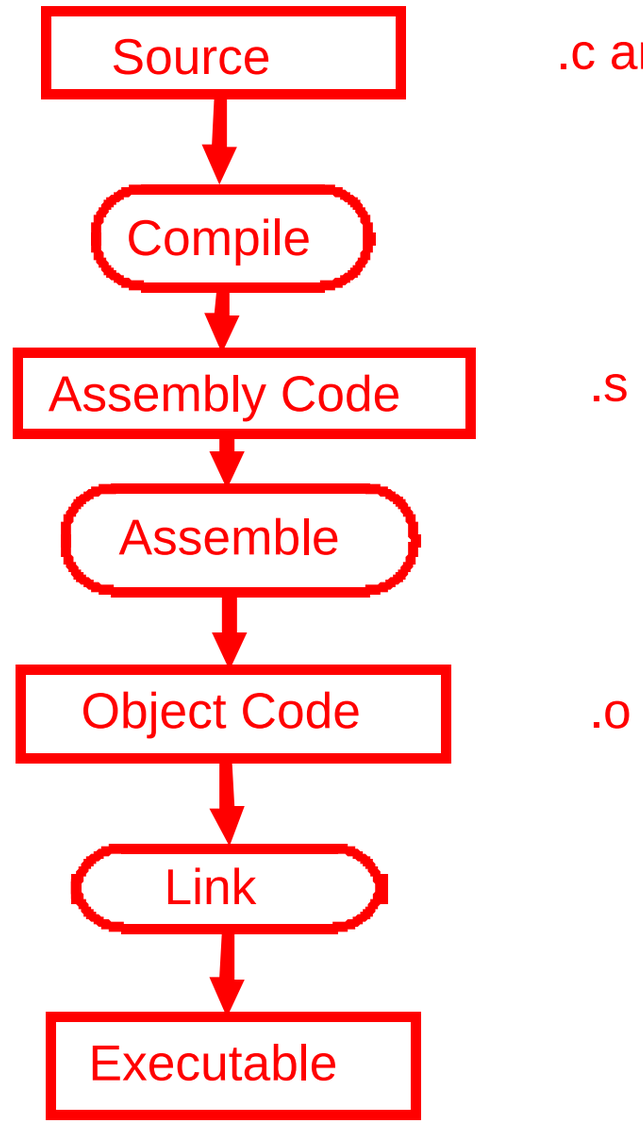
Good Style

The objective of a good program is to use a consistent standard style that makes it easier for a person reading your code to understand it.

Well styled code will have:

- easy to understand variable, constant, and function names,
- consistent indentation and line spacing,
- coherent functions,
- comments as required and helpful for the reader,
- use named constants rather than magic numbers,
- reduce duplication in code, and
- prioritize readability over often small perceived gains in performance.

Source to Executable



.c and .h

.s

.o

As programs get larger

As programs get larger you will often wish to partition your code into a number of modules.

This will involve separating the code into the source and header files. Header files are like the public interface into your code. They have .h file name extensions and share the same root name as the .c file. Header files would normally contain:

- prototypes of functions that are used outside that .c code (the public functions).
- constants
- structures
- the declaration of external global variables.

Exersizes

- Find both a small and a large c program from the internet and have a look how they are structured. Do they use prototypes? What is in the header files?

- Take a program that you have written and use the -S option view how that program is translated into assembly language.

- What does the 'static' keyword do in c?