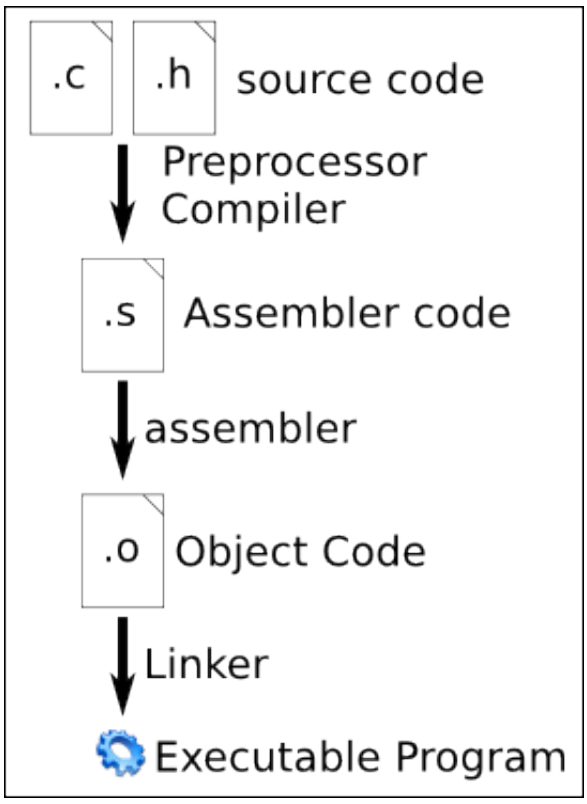
Compiling C language source code

Compiling Steps



• The Preprocessor stage: The C preprocessor is a macro processor that is used automatically by the C compiler to transform your program before actual compilation. It is called a macro processor because it allows you to define macros, which are brief abbreviations for longer constructs.

• The Compiler stage: C code is converted to low level machine instructions (assembly).

• The Assembler stage: The assembly language code is then converted into object code which are fragments of code which the computer understands directly. It is not necessarily executable directly without linking to other modules.

• The Linker stage: Linking the object code to code libraries which contain certain ”built-in” functions, such as printf. This stage produces an executable program.

The GNU Compiler Collection (GCC)

GCC is the most common Unix C compiler; but it is available on many non-Unix systems as well: VMS, DOS, various version of MS windows, etc... Because of this, many "portable" applications are written using it.

GCC command line compilation

Simple Example: main.c

```c
#include <stdio.h>

int main(void)
{
        printf("Hello World\n");
        return 0;
}
```

Browse in your shell to the folder where your source file is located.

Run the follow command:

$ gcc -Wall -o HelloWorld main.c

gcc : invokes the gcc compiler

-wall option: specifies that you want all warning messages to be reported to you.

-o <filename> option: The compiled binary is saved with specified filename. If not specified, the default name is 'a.out'.

main.c : the source file you want to compile

result:

an executable file called “HelloWorld.exe”

to run : $ ./HelloWorld

GCC options:

GCC has many options or command switches, we will only show the ones most important to you.

$ gcc -c : gcc will compile and assembles your source WITHOUT linking.

→ result: *.o (object files)

$ gcc -o <filename> : here you specify a filename instead of the default a.out

$ gcc -g : includes debugger info for GDB (GNU Debugger)

$ gcc -Wall : show compiling warnings and errors.

$ gcc -O : With '-O', the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

$ gcc -O1 $ gcc -O2 $ gcc -O3 : various degrees of optimization targeted for speed.

$ gcc -O0 : no optimization

$ gcc -Os : optimization for generated code size

For your assignments, We strongly suggest to always include the -Wall option.

Preprocessor options:

The -D*NAME* option:

```
#include <stdio.h>

int main (void)
{
#ifdef TEST
        printf ("Test mode\n");
#endif
        printf ("Running...\n");

return 0;
}
```

when compiling with the '-D*NAME*' option
e.g.:$ gcc -Wall -D*TEST -o preprocessor preprocessor.c*

Program Output:

```
Test mode

Running...
```

Macros are generally undefined, unless specified on the command line with the option '-D', or in a source file (or library header file) with #define.

The -E option:

The '-E' option causes gcc to run the preprocessor, display the expanded output, and then exit without compiling the resulting source code.

Compilation of a multiple source project

The more common way of compiling executables using gcc (at least for big projects) is to *compile* each C file separately and then *link* them together afterwards. This is usually done using Makefiles, but can also be done manual. Use the -c switch to compile each .c file to a .o file. Afterwards, use a gcc line referring to the .o files to link them together. When you have modified one file e.g.: stack.c you do not need to recompile main.c. You only need to re-excute lines 2 and 3. This delivers a great time saving advantage when creating/compiling larger projects.

Example using gcc to compile two files and then link them separately:

1. $ gcc -Wall -o main.o -c main.c
2. $ gcc -Wall -o stack.o -c stack.c
3. $ gcc -Wall -o stack_version3 main.o stack.o

The GNU Debugger (GDB)

A debugger is a program that runs other programs, allowing the user to exercise control over these programs, and to examine variables when problems arise. The most popular debugger for UNIX systems is GDB.

Compiling your program for GDB: use the -g option e.g.: $ gcc -Wall -g -o HelloWorld main.c

Starting GDB: attach the debugger to your program: $ gdb HelloWorld

GDB command line functions

(GDB)break main.c:5	*Placing Breakpoints:* (GDB) break <*filename*>:*line number*
(GDB)run	Your program will start until it reaches the first breakpoint
(GDB)delete N	*Delete Breakpoints:* N = the line number where the breakpoint is placed Leave off *N* to remove all breakpoints
(GDB)info break	Lists information about all your breakpoint
(GDB)step	Executes the current line of the program and stops on the next statement continue
(GDB)continue	Continues regular execution of the program until a breakpoint is hit or the program stops
(GDB)print *var*	*P*rints the value of var *(variable used in your code)* in the current frame in the program
(GDB)quit	Leave GDB

The GNU Profiler (GPROF)

Profiling allows you to learn where your program spent its time and which functions called which other functions while it was executing.

Compile your program to link Gprof: use the -pg option : $ gcc -Wall -g -pg -o HelloWorld main.c

Run your program in the normal way: $ ./HelloWorld

an extra output file is produced by gprof: 'gmon.out'

Run gprof : gprof *options* [*executable-file* [*profile-data-files*...]] [> *outfile*]

e.g.: $ gprof -p -s HelloWorld gmon.out > result.txt

Gprof Options

-a : Do not display statically declared functions
-b : Do not display information about each field in the profile
-i : Print a summary information on datafiles afterwards exit
-m *n* : Don't print count statistics for symbols executed less than *n* times.
-n : Propogate time statistics in call graph analysis.
- p : Print profile statics
-q : Print call graph analysis.
-s : Summarize profile information in the file *gmon.sum*.
-v : Print version and exit.
-z : Include zero-usage calls.
-P : Don't print profile statistics
-Q : Don't print call graph analysis.

GCC for Windows

1) Goto http://www.mingw.org/ download + install

2) Open your cmd shell and type in "gcc –help"

doesn't work?

" 'gcc' is not recognized as an internal or external command, operable program or batch file"

Solution : Add the location of the gcc binary to your PATH environment variable.

3) First check if this is the location of gcc.exe → C:\MinGW\bin ?

4) Add this to the PATH environment variable 'path = %PATH%;C:\MinGW\bin'

5) Check this with 'echo %PATH%' or 'gcc --help'