Computer Programming

Irwan Prasetya Gunawan, Ph.D

Sampoerna University
ipg@ieee.org

00: Introduction: Overview, Code Building, IDE

Version: July 10, 2019

Contents

About the course

- **Course description:** This course introduce students to the fundamental principles of programming for solving engineering problems, using the C programming language. It familiarizes students with the process of computational thinking and the translation of real life engineering problems to computation problems. Further, it describes the basic technique for systematic software design. It provides fundamental knowledge in basic programming concepts such as program flow control, memory management, and elementary data structures.
- **Credit Hours:** 3
- **Objectives:** Upon completion of this course, students should have achieved the following objectives:
  - Conceptualize engineering problems as computational problems
  - Code basic computing problems using the C language
  - Learn fundamental software design principles and commonly used techniques

Textbooks/Instructional Materials

Textbook

- K. N. King. C Programming: A Modern Approach (Second Edition). Norton and Company, 2008.
- George Tselikis and Nikolaos Tselikas. C: From Theory to Practice. CRC Press.
- B. W. Kernighan and D. M. Ritchie. The C Programming Language (Second Edition). Prentice Hall, 1998.

Online Resources

- Zyante's Programming in C interactive textbook, available at: https://www.zyante.com/
- Daniel Weller, and Sharat Chikkerur. *6.087 Practical Programming in C*. January IAP 2010. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu. License: Creative Commons BY-NC-SA.

Evaluation

Grades:

- Mid Test: 25%
- Final Test: 25%
- Quiz: 20%
- Project: 15%
- Assignment: 10%
- Participation: 5%

Note:

- Late submission of assignment/homeworks/etc: 25% discount
- (If there is a) Quiz will not be announced beforehand; so, always be prepared!

Warning

- No other types of grade will be given.
- **There will not be any extra assignments once the Final Test has been conducted**

Required Installations

The required installations are:

- GCC on Windows using MinGW [1]
- MSYS [2]
- (Any) text editors
- Eclipse based IDE [3]

GCC on Windows

GCC

- GCC, formerly known as "GNU C Compiler", supports many languages such as C (gcc), C++ (g++), Objective-C, Objective-C++, Java (gcj), Fortran (gfortran), and many others.
- It is now referred to as "GNU Compiler Collection".
- The mother site for GCC is http://gcc.gnu.org/.
- For the current course, GCC is chosen because it is a cross-compiler that can produce executables on different platforms.
- The other reason is its relatively simple process (during compiling) compared to other C compilers.
- In addition, GCC can also be ported to Windows (by Cygwin, MinGW and MinGW-W64).

Flavours of GCC on Windows

For Windows environment, there are several different flavours of GCC that can be installed:

- *Cygwin GCC*: Cygwin is a Unix-like environment and command-line interface for Microsoft Windows. Cygwin is huge and includes most of the Unix tools and utilities. It also included the commonly-used Bash shell.
- *MinGW*: MinGW (Minimalist GNU for Windows) is a port of the GNU Compiler Collection (GCC) and GNU Binutils for use in Windows. It also included MSYS (Minimal System), which is basically a Bourne shell (bash).
- *MinGW-W64*: a fork of MinGW that supports both 32-bit and 64-bit windows.

Note

For the current course, MinGW is preferred to the other two options because of its lightweight installations.

MinGW Installation Steps

1. Go to MinGW mother site at http://www.mingw.org/ ⇒ Downloads ⇒ Installer ⇒ click on "mingw-get-inst" link to download the installer.
2. Run the downloaded installer.
   1. Set the installation directory. It is recommended to use the default one, i.e., `c:\mingw` although other directory can also be used as long as its name does not have space in it (for example, try not to install under "Program Files" or "Desktop").
   2. In MinGW Installation Manager, select "Installation" ⇒ "Update Catalogue" ⇒ Select all packages in "Basic Setup" ⇒ continue.
3. Setup environment variable PATH to include "<MINGW_HOME/bin>" where "<MINGW_HOME/bin>" is the MinGW installed directory that have been chosen in the previous step.

Notes

The above-mentioned steps are using the automated installer. However, post-install tasks to ensure that MinGW application is running properly must also be done. A detailed explanation on how to install MinGW is given in http://www.mingw.org/wiki/Getting_Started.

MinGW Post Installation Check I

Display the version of GCC via `--version` option by invoking these commands via CMD terminal:

```
C:\Users\irwan>gcc --version
gcc (MinGW.org GCC-8.2.0-3) 8.2.0
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
    There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
    PURPOSE.

C:\Users\irwan>g++ --version
g++ (MinGW.org GCC-8.2.0-3) 8.2.0
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
    There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
    PURPOSE.
```

MinGW Post Installation Check II

More details can be obtained via -v option:

```
C:\Users\irwan>gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=c:/mingw/bin/../libexec/gcc/mingw32/8.2.0/lto-
    wrapper.exe
Target: mingw32
Configured with: ../src/gcc-8.2.0/configure --build=x86_64-pc-linux
    -gnu --host=mingw32 --target=mingw32 --prefix=/mingw --disable-
    win32-registry --with-arch=i586 --with-tune=generic --enable-
    languages=c,c++,objc,obj-c++,fortran,ada --with-pkgversion=
'MinGW.org GCC-8.2.0-3' --with-gmp=/mingw --with-mpfr=/mingw --with
    -mpc=/mingw --enable-static --enable-shared --enable-threads --
    with-dwarf2 --disable-sjlj-exceptions --enable-version-specific
    -runtime-libs --with-libiconv-prefix=/mingw --with-libint
l-prefix=/mingw --enable-libstdcxx-debug --with-isl=/mingw --enable
    -libgomp --disable-libvtv --enable-nls --disable-build-format-
    warnings
Thread model: win32
gcc version 8.2.0 (MinGW.org GCC-8.2.0-3)
```

MinGW Post Installation Check III

Help manual can be invoked via the `--help` option:

```
C:\Users\irwan>gcc --help
Usage: gcc [options] file...
Options:
-pass-exit-codes         Exit with highest error code from a phase.
--help                   Display this information.
--target-help            Display target specific command line
    options.
--help={common|optimizers|params|target|warnings|[^]{joined|
    separate|undocumented}}[,...].
Display specific types of command line options.
(Use '-v --help' to display command line options of sub-processes).
--version                Display compiler version information.
-dumpspecs               Display all of the built in spec strings.
...
```

MSYS

- MSYS is a collection of GNU utilities such as `bash`, `make`, `gawk` and `grep` to allow building of applications and programs which depend on traditionally UNIX tools to be present. It is intended to supplement MinGW and the deficiencies of the Windows' own CMD shell [2].
- In the current course, MSYS installation is primarily intended to support `make` utility in compiling C program under MinGW-based GCC in Windows environment.

MSYS Installation

All the MSYS components are now available as separate downloads managed by `mingw-get`. As a pre-requisite, MinGW should be installed first, preferably under default directory such as `C:\MinGW`

1. Make sure that MinGW is already installed in Windows
2. Get MSYS installer from http://downloads.sourceforge.net/mingw/MSYS-1.0.11.exe. By default, it will be installed in `C:\msys\1.0` although any directories will also be fine.
3. Next, the post install process will ask for the directory where MinGW was installed to. Assuming that MinGW has been installed in the default directory, then we should enter `c:/mingw` (notice the use of 'slash' / instead of 'backslash' \ )
4. Install MSYS Development Tool Kit (DTK) from http://downloads.sourceforge.net/mingw/msysDTK-1.0.1.exe
5. Set the Windows environment variable `HOME` to `C:\msys\1.0\home`

MSYS Post-Installation Checks I

- By now, there should be a cyan "M" link on Windows desktop, which, upon double click will launch a terminal configured to run some of the mostly used GNU utilities (with respect to the current course) traditionally only available on UNIX environment.
- Additionally, it is also convenient to have MinGW installation mounted on `/mingw` since `/mingw` is on MSYS PATH by default. If MinGW installation is not automatically mounted by MSYS installation, then we need to run the following command from the MSYS-launched terminal:

```
mount c:/mingw /mingw
```

MSYS Post-Installation Checks II

- Windows user home drive, e.g., `C:\Users\irwan`, is by default mounted to `/home`. Several others default mount by MSYS, including auto-mounted directories, can be listed using the command `mount` (without any arguments) such as follows:

```
irwan@UB-BENFANO ~
$ mount
C:\Users\irwan\AppData\Local\Temp on /tmp type user (binmode,
    noumount)
C:\msys\1.0 on / type user (binmode,noumount)
C:\msys\1.0 on /usr type user (binmode,noumount)
c:\MinGW on /mingw type user (binmode)
c: on /c type user (binmode,noumount)
d: on /d type user (binmode,noumount)
e: on /e type user (binmode,noumount)
f: on /f type user (binmode,noumount)


irwan@UB-BENFANO ~
$
```

Text editor

For the current course, any text editors would be fine. A recommended Windows-based text editor (or preferably source code editor) that recognizes C/C++ syntaxes is Notepad++ downloadable from
https://notepad-plus-plus.org/download/

Eclipse C/C++ Development Tool (CDT)

- Eclipse is an open-source Integrated Development Environment (IDE) supported by IBM.
- The mother site is `www.eclipse.org`.
- Eclipse is popular for Java project development. It also supports C/C++, PHP, Python, Perl, and other web project developments via extensible plug-ins.
- Eclipse is cross-platform and runs under Windows, Linux and Mac OS.

Eclipse Installation I

There are two ways to install Eclipse CDT, depending on whether we have previously installed an Eclipse:

1. If we have already installed "Eclipse for Java Developers" or other Eclipse packages, we could install the CDT plug-in as follows:
    1. Launch Eclipse ⇒ Help ⇒ Install New Software ⇒ In "Work with" field, pull down the drop-down menu and select "Kepler - http://download.eclipse.org/releases/kepler" (or juno for Eclipse 4.2; or helios for Eclipse 3.7).
    2. In "Name" box, expand "Programming Language" node ⇒ Check "C/C++ Development Tools" ⇒ "Next" ⇒ ... ⇒ "Finish".

Eclipse Installation II

2. If we have not installed any Eclipse package, we could download "Eclipse IDE for C/C++ Developers" from http://www.eclipse.org/downloads, and unzip the downloaded file into a directory of our choice.
   By default, the package for Windows is mostly for the latest version which comes as 64-bit machine. For 32-bit Windows, we can install the slightly outdated version of Eclipse from https://www.eclipse.org/downloads/packages/release/kepler/sr2/eclipse-ide-cc-developers
3. There is NO NEED to perform any configuration as long as MinGW binary is included in the PATH environment variable because CDT will search the PATH to discover the C/C++ compilers.

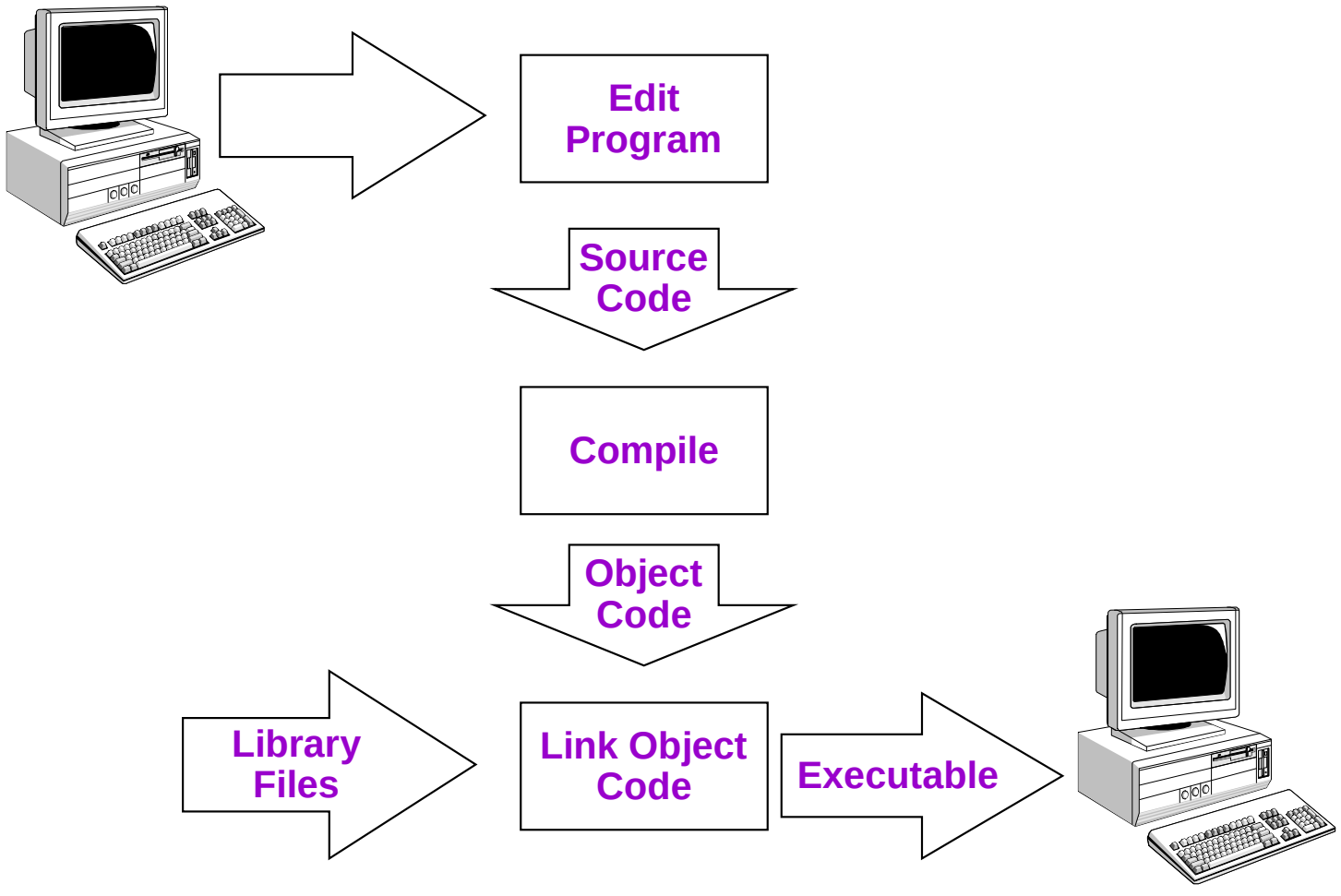
A Brief History

- Created by Dennis Ritchie at AT&T Labs in 1972
- Originally created to design and support the Unix operating system.
- There are only 27 keywords in the original version of C.
  - for, goto, if, else ……
- Easy to build a compiler for C.
  - Many people have written C compilers
  - C compilers are available for virtually every platform
- In 1983 the American National Standards Institute (ANSI) formed a committee to establish a standard definition.
  - Called ANSI Standard C.
  - As opposed to K&R C (referring to the general “standards” that appeared in the first edition of Brian Kernighan and Ritchie’s influential book: *The C Programming Language*)

Why use C?

- C is intended as a language for programmers
  - BASIC was for nonprogrammers to program and solve simple problems.
  - C was created, influenced, and field-tested by working programmers.
- C is powerful and efficient
  - You can nearly achieve the efficiency of assembly code.
  - System calls and pointers allow you do most of the things that you can do with an assembly language.
- C is a structured language
  - Code can be written and read much easier.
- C is standardized
  - Your ANSI C program should work with any ANSI C compiler.

The C Development Cycle



“Hello World”

- Everyone writes this program first

```
#include <stdio.h>
int main ( )
{
    printf ("Hello, World!\n");
    return 0;
}
```

Compilation (1)

- Compilation translates your source code (in the file hello.c) into object code (machine dependent instructions for the particular machine you are on).
  - Note the difference with Java:
    - The javac compiler creates Java byte code from your Java program.
    - The byte code is then executed by a Java virtual machine, so it's machine independent.
- Linking the object code will generate an executable file.
- There are many compilers for C under Unix
  - SUN provides the Workshop C Compiler, which you run with the cc command
  - There is also the freeware GNU compiler gcc

Compilation (2)

- To compile a program:
  - Compile the program to object code.

    obelix[2] > cc –c hello.c
  - Link the object code to executable file.

    obelix[3] > cc hello.o –o hello
- You can do the two steps together by running:

  obelix[4] > cc hello.c –o hello
- To run your program:

  obelix[5] > ./hello

  Hello World!

If you leave off the -o, executable goes into the file a.out

Compilation (3)

- Error messages are a little different than you may be used to but they can be quite descriptive.
- Suppose you forgot the semi-colon after the printf

```
obelix[3] > cc hello.c –o hello
"hello.c", line 5: syntax error before or at: return
cc: acomp failed for hello.c
```

- Notice that the compiler flags and informs you about the error at the first inappropriate token.
  - In this case, the return statement.
- Always try to fix problems starting with the first error the compiler gives you - the others may disappear too!

Example 1

```
#include <stdio.h>
int main ()
{
    int radius, area;

     printf ("Enter radius (i.e. 10) : ");
     scanf ( "%d", &radius);
     area = 3.14159 * radius * radius;
     printf ("\nArea = %d\n\n", area);
     return 0;
}
```

Example 2

```
#include <stdio.h>
int main ()
{
    int i, j;
    for (i = 0; i < 10; i++)
    {
            printf ("\n");
            for (j = 0; j < i+1; j++ )
                printf ( "A");
    }
    printf("\n");
    return 0;
}
```

Example 3

```
/* Program to calculate the product of
     two numbers */

#include <stdio.h>

int product(int x, int y);

int main ()

{

   int a,b,c;

   /* Input the first number */

   printf ("Enter a number between 1
              and 100: ");

   scanf  ("%d", &a);


   /* Input the second number */

   printf ("Enter another number
              between 1 and 100: ");

   scanf  ("%d", &b);

   /* Calculate and display the product */

   c = product (a, b);

   printf ("%d times %d = %d \n", a, b, c);

   return 0;

}


/* Functions returns the product of its
     two arguments */

int product (int x, int y)

{

   return (x*y);

}
```

Makefiles

Multiple Source Files (1)

- Obviously, large programs are not going to be contained within single files.
- C provides several techniques to ensure that these multiple files are managed properly.
  - These are not enforced rules but every good C programmer know how to do this.
- A large program is divided into several modules, perhaps using abstract data types.
- The header (.h) file contains function prototypes of a module.
- The (.c) file contains the function definitions of a module.
- Each module is compiled separately and they are linked to generate the executable file.

Multiple Source Files (2)

- C programs are generally broken up into two types of files.
  - .c files:
    - contain source code (function definitions) and global variable declarations
    - these are compiled once and never included
  - .h files:
    - these are the “interface” files which “describe” the .c files
      - type and struct declarations
      - const and #define constant declarations
      - #includes of other header files that must be included
      - prototypes for functions

Example - Main Program sample.c

```
#include <stdio.h>
#include "my_stat.h"
int main()
{
    int a, b, c;
    puts("Input three numbers:");
    scanf("%d %d %d", &a, &b, &c);
    printf("The average of %d %d %d is %f.\n",
            x,y,z,average(a,b,c));
    return 0;
}
```

Example - Module my_stat

```
* my_stat.h */
#define PI 3.1415926
loat average(int x, int y, int z);
loat sum( int x, int y, int z);
```

```
/* my_stat.c */
#include "my_stat.h"
float average(int x, int y, int z)
{
    return sum(x,y,z)/3;
}

float sum(int x, int y, int z)
{
    return x+y+z;
}
```

Example - Compile the Sample Program

- You need my_stat.c and my_stat.h to compile the my_stat module to object code

  cc –c my_stat.c

- You need my_stat.h and sample.c to compile sample.c to object code

  cc –c sample.c

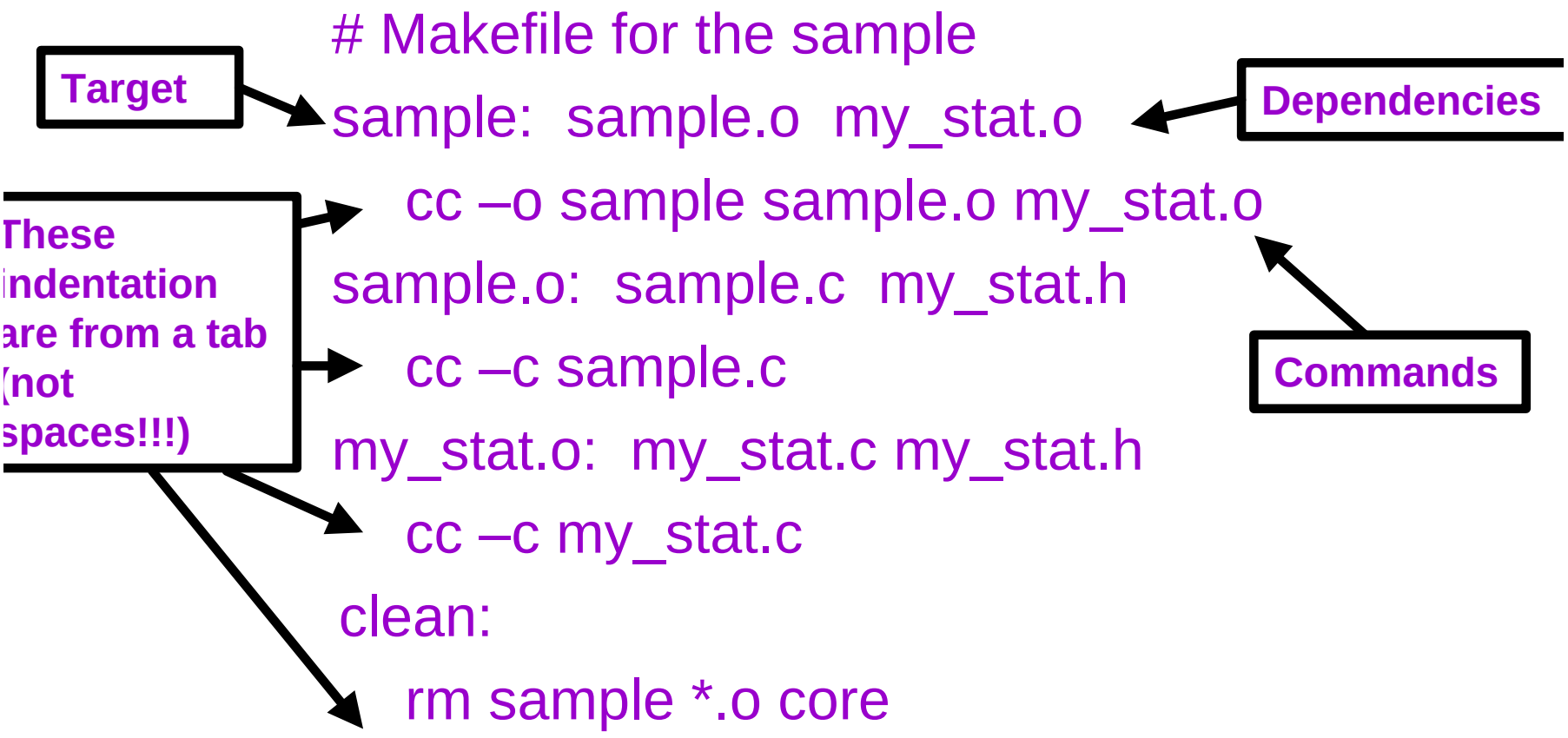
- You need my_stat.o and sample.o to generate an executable file

  cc –o sample sample.o my_stat.o

- Therefore, the module my_stat can be reused just with the my_stat.o and my_stat.h. In fact, this is how the standard libraries work. (Libraries are just collections of object code, with headers describing functions and types used in the libraries.)

The make Utility (1)

- Programs consisting of many modules are nearly impossible to maintain manually.
- This can be addressed by using the make utility.



The make Utility (2)

- Save the file with name "Makefile" (or "makefile") at the same directory.
- For every time you want to make your program, type

  make
- The make utility will
  - Find the Makefile
  - Check rules and dependencies to see if an update is necessary.
  - Re-generate the necessary files that need updating.
- For example:
  - If only sample.c is newer than sample, then only the following commands will be executed:
    - cc –c sample.c
    - cc –o sample sample.o my_stat.o

The make Utility (3)

- To clean all generated files:

  make clean

- To re-compile, you can
  - Remove all generated files and make again.
    - make clean; make
  - Or you can:
    - touch my_stat.h and then make again
    - This changes the time stamp of my_stat.h, so make thinks it necessary to make all the files.

Using make with Several Directories

- As the number of .c files for a program increases, it becomes more difficult to keep track of all the parts.
- Complex programs may be easier to control if we have one Makefile for each major module.
- A program will be stored in a directory that has one subdirectory for each module, and one directory to store *all* the .h files.
- The Makefile for the main program will direct the creation of the executable file.
- Makefiles for each module will direct the creation of the corresponding .o files.

A Makefile Example (1)

- Consider a C program that uses a Stack ADT, a Queue ADT and a main module.
- Suppose that the program is in seven files: StackTypes.h, StackInterface.h, QueueTypes.h, QueueInterface.h, StackImplementation.c, QueueImplementation.c, and Main.c
- We will build the program in a directory called Assn that has four subdirectories: Stack, Queue, Main, and Include.
- All four .h files will be stored in Include.

A Makefile Example (2)

- Stack contains StackImplementation.c and the following Makefile:

```
export: StackImplementation.o

StackImplementation.o: StackImplementation.c \
                       ../Include/StackTypes.h \
                       ../Include/StackInterface.h
        gcc -I../Include -c StackImplementation.c
# substitute a print command of your choice for lpr below
print:
        lpr StackImplementation.c
clean:
        rm -f *.o
```

A Makefile Example (3)

- Queue contains QueueImplementation.c and the following Makefile:

```
export: QueueImplementation.o

QueueImplementation.o: QueueImplementation.c \
                       ../Include/QueueTypes.h \
                       ../Include/QueueInterface.h
        gcc -I../Include -c QueueImplementation.c
# substitute a print command of your choice for lpr below
print:
        lpr QueueImplementation.c
clean:
        rm -f *.o
```

A Makefile Example (4)

- Note: The -I option (uppercase i) for cc and gcc specifies a path on which to look to find .h files that are mentioned in statements of the form #include "StackTypes.h" in .c files.
- It is possible to specify a list of directories separated by commas with -I.
- By using -I, we can avoid having to put copies of a .h file in the subdirectories for every .c file that depends on the .h file.

A Makefile Example (5)

- Main contains Main.c and the following Makefile:

```
export: Main

Main: Main.o StackDir QueueDir
        gcc -o Main Main.o ../Stack/StackImplementation.o \
                ../Queue/QueueImplementation.o
Main.o: Main.c ../Include/*.h
        gcc -I../Include -c Main.c
StackDir:
        (cd ../Stack; make export)
QueueDir:
        (cd ../Queue; make export)

#continued on next page
```

A Makefile Example (6)

```
print:
        lpr Main.c
printall:
        lpr Main.c
        (cd ../Stack; make print)
        (cd ../Queue; make print)

clean:
        rm -f *.o  Main  core
cleanall:
        rm -f *.o  Main  core
        (cd ../Stack; make clean)
        (cd ../Queue; make clean)
```

A Makefile Example (7)

- Note: When a sequence of Unix commands is placed inside parentheses (), a new subprocess is created, and the commands are executed as part of that subprocess.
- For example, when (cd ../Stack; make export) is executed, the subprocess switches to the Stack directory and executes the make command; when the subprocess terminates, the parent process resumes in the original directory. No additional cd command is needed.

Using Macros in Makefiles

- Macros can be used in Makefiles to reduce file size by providing (shorter) names for long or repeated sequences of text.
- Example: The definition name = text string creates a macro called name whose value is text string.
- Subsequent references to $(name) or ${name} are replaced by text string when the Makefile is processed.
- Macros make it easier to change Makefiles without introducing inconsistencies.

Makefile Example Revisited (1)

- The Makefile for Stack can become:

```
CC = gcc
HDIR = ../Include
INCPATH = -I$(HDIR)
DEPH =  $(HDIR)/StackTypes.h  $(HDIR)/StackInterface.h
SOURCE = StackImplementation
export:  $(SOURCE).o

$(SOURCE).o:  $(SOURCE).c  $(DEPH)
        $(CC)  $(INCPATH)  -c  $(SOURCE).c
print:
        lpr  $(SOURCE).c
clean:
        rm  -f  *.o
```

Makefile Example Revisited (2)

- The Makefile for Queue can become:

```
CC = gcc
HDIR = ../Include
INCPATH = -I$(HDIR)
DEPH =  $(HDIR)/QueueTypes.h  $(HDIR)/QueueInterface.h
SOURCE = QueueImplementation

export:  $(SOURCE).o

$(SOURCE).o:  $(SOURCE).c  $(DEPH)
        $(CC)  $(INCPATH)  -c  $(SOURCE).c
print:
        lpr  $(SOURCE).c
clean:
        rm  -f  *.o
```

Makefile Example Revisited (3)

- The Makefile for Main.c can become:

```
CC = gcc
HDIR = ../Include
INCPATH = -I$(HDIR)
DEPH =  $(HDIR)/QueueTypes.h  $(HDIR)/QueueInterface.h
OBJ = ../Stack/StackImplementation.o   \
          ../Queue/QueueImplementation.o

export: Main

Main: Main.o  StackDir  QueueDir
          $(CC)  -o  Main  Main.o  $(OBJ)

#continued on next page
```

Makefile Example Revisited (4)

```
Main.o: Main.c  $(HDIR)/*.h
        $(CC)  $(INCPATH)  -c  Main.c
StackDir:
        (cd ../Stack; make export)
QueueDir:
        (cd ../Queue; make export)
print:
        lpr  Main.c
printall:
        lpr  Main.c
        (cd ../Stack; make print)
        (cd ../Queue; make print)

# continued on next page...
```

Makefile Example Revisited (5)

```
clean:
        rm  -f  *.o  Main  core
cleanall:
        rm  -f  *.o  Main  core
        (cd ../Stack; make clean)
        (cd ../Queue; make clean)
```

A Makefile Exercise

- Rewrite the Makefiles for the previous example so that the command make debug will generate an executable Maingdb that can be run using the debugger gdb.

More Advanced Makefiles

- Many newer versions of make, including the one with Solaris and GNU make program gmake include other powerful features.
  - Control structures such as conditional statements and loops.
  - Implicit rules that act as defaults when more explicit rules are not given in the Makefile.
  - Simple function support for transforming text.
  - Automatic variables to refer to various elements of a Makefile, such as targets and dependencies.

- See the following web page for more details on gmake: http://www.gnu.org/manual/make/html_mono/make.html

References I

[1] MinGW Installation Notes Wiki Page, May 2007. URL http://www.mingw.org/wiki/Getting_Started. Accessed 18 June 2019.

[2] MSYS Wiki Page, January 2008. URL http://www.mingw.org/wiki/MSYS. Accessed 18 June 2019.

[3] Eclipse Foundation. Eclipse Project. URL https://www.eclipse.org/.

[4] Chua Hock-Chuan. Yet another insignificant . . . programming notes, March 2019. URL https://www3.ntu.edu.sg/home/ehchua/programming/index.html. Accessed 18 June 2019.

[5] Marc Moreno Maza. Lecture notes: Software tools and systems programming. University of Western Ontario, 2011.

[6] Daniel Weller and Sharat Chikkerur. 6.087 Practical Programming in C. Massachusetts Institute of Technology: MIT OpenCourseWare, January 2010. URL https://ocw.mit.edu. License: Creative Commons BY-NC-SA.