UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Programming Technique II

GNU Compiler Collection

GNU Compiler Collection : Tools

- Compilers
  - C : *gcc*
  - C++ : *g++*
- Debuggers
  - Text : *gdb*
  - GUI : *emacs*
- Integrated Development Environments (IDE)
  - *Anjuta*
  - *KDevelop*
  - *source-navigator*

UTM
UNIVERSITI TEKNOLOGI MALAYSIA
www.utm.my

GCC / G++ : Introduction

- GNU is a free software / operating system.
- The plan for the GNU operating system was announced in September 1983 by Richard Stallman and software development work began in January 1984.
- The project to develop GNU is known as the GNU Project, and programs released under the auspices of the GNU Project are called GNU packages or GNU programs.
- Licensing
  - In order to ensure that GNU software remains free, the project released the first version of the GNU General Public License (GNU GPL) in 1989.
  - The GNU Lesser General Public License (LGPL) is a modified version of the GPL, intended for some software libraries.
  - It gives all recipients of a program the right to run, copy, modify and distribute it, while forbidding them from imposing further restrictions on any copies they distribute.

GCC / G++ : Introduction

- The GNU Compiler Collection (GCC) is a set of programming language compilers produced by the GNU Project.
- It is free software distributed by the Free Software Foundation (FSF) under the GNU GPL and GNU LGPL, and is a key component of the GNU toolchain.
- It is the standard compiler for the free software Unix-like operating systems, and certain proprietary operating systems derived therefrom such as Mac OS X.
- Originally named the GNU C Compiler, because it only handled the C programming language, GCC was later extended to compile C++, Java, Fortran, Ada, and others.

GCC / G++ : Introduction

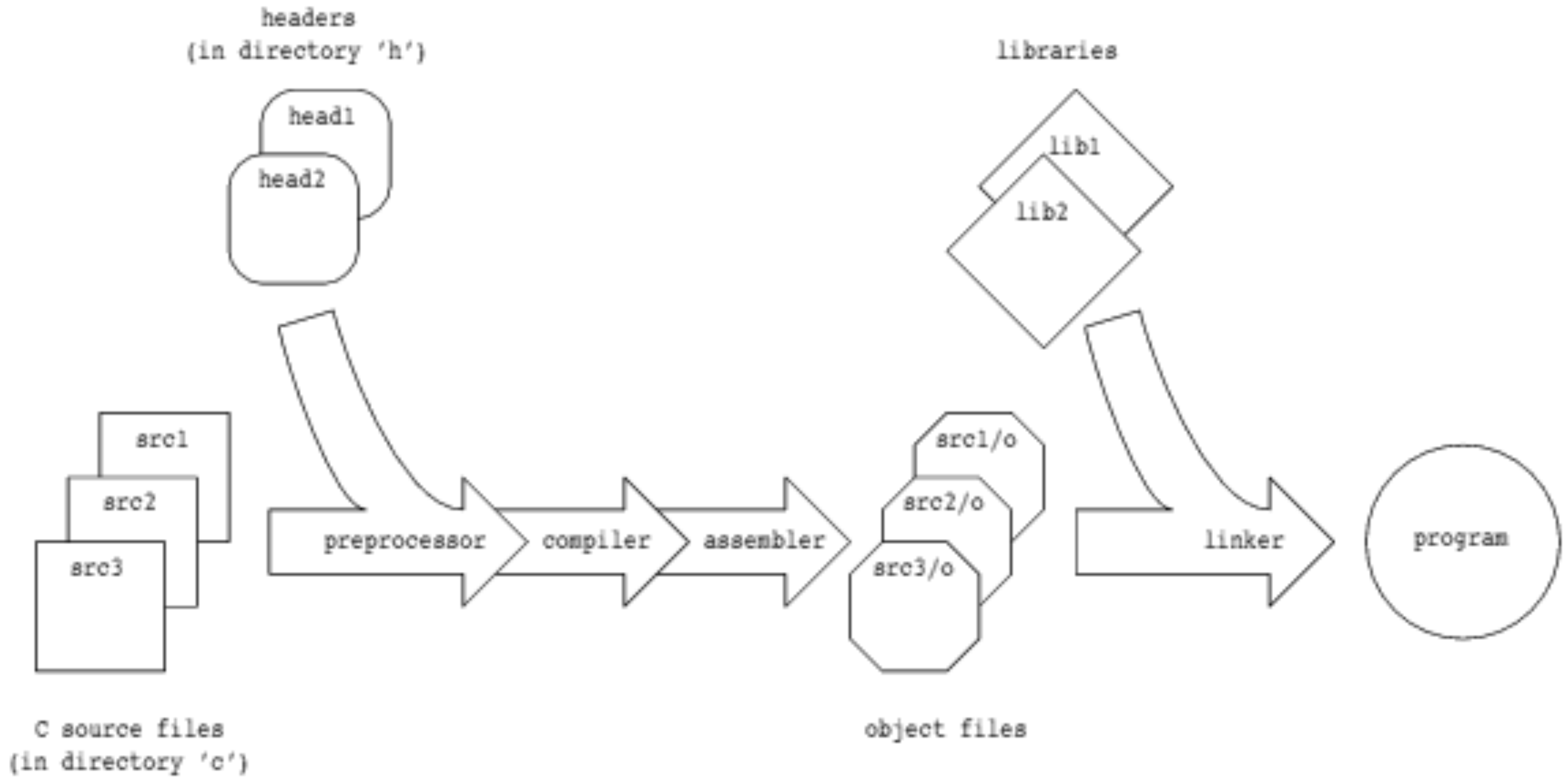
- Standard compiler on:
  - GNU/Linux
  - Other free operating systems
  - OS X
- Used on almost every other platforms:
  - Windows
  - Embedded systems
  - UNIX and UNIX-like systems (Linux / BSD)
- Support for a very wide variety of architectures.
  - Same behavior on all systems.
  - Support for kernel/embedded programming.
- Good code generation.
- Standards compliance.

GCC / G++ : 4 Stages of Compilation

Stage	Input	Output
preprocessor	source code	pre-processed source code
compiler	pre-processed source code	assembly source code
assembler	assembly source code	object file
linker	object files	executable file

UTM
UNIVERSITI TEKNOLOGI MALAYSIA
www.utm.my

GCC / G++ : 4 Stages of Compilation



GCC / G++ : 4 Stages of Compilation

- The compiler can be stopped at any stage using the appropriate flag:
  - -E -- stops after the preprocessing stage. It outputs source code after preprocessing to standard out (the terminal).
  - -S -- stops after the compile stage. It outputs the assembly for each source file to a file of the same name but with a ".s" extension.
  - -c -- stops after the assemble stage. It outputs an object file for each source file with the same name but with an ".o" extension.

GCC / G++ : Compilation Flags

- There are also many other useful compiler flags that can be supplied. We will cover some of the more important ones here but others can be found in manual g++.
  - **-g** : includes debug symbols
    This flag must be specified to debug programs using gdb.
  - **-Wall** : show all compiler warnings.
    This flag is useful for finding problems that are not necessarily compile errors. It tells the compiler to print warnings issued during compile which are normally hidden.
  - **-o** : the name for the compiled output.
  - **-v** :Give details of what gcc is doing. Use this to help track down problems.
  - **-ansi** : Force ANSI compliant compilation

GCC / G++ : Compilation Flags

- **-O** or **-O1** - optimizes programs.

  This tells the compiler to reduce the code size and execution time of the program it produces. This may cause unexpected behavior when run in debug, since GCC is taking every opportunity to eliminate temporary variables and increase efficiency. The process generally takes much more time and memory for compilation, but the resulting executable may be drastically faster.
- **O2** - optimize even more.

  This performs almost all supported optimizations except loop unrolling, function inlining and register renaming.
- **-O3** - all optimizations
- **-pg** - generates extra code to write profile information suitable for the analysis program "gprof".
- **-fprofile-arcs** - used to generate coverage data or for profile-directed block ordering.

  During execution the program records how many times each branch is executed and how many times a branch is taken. When the compiled program exits it saves this data to a file called *sourcename.da* for each source file.
- **-ftest-coverage** - create data files for the “gcov” code-coverage utility.

GCC / G++ : Example: A basic compile

- *g++* syntax:

  g++ [<options>] <input files>

- Some useful options:

-g	produce debugging info (for gdb)
-Wall	"*Warnings all*"
-ansi	force ANSI compliant compilation
-D<sym>	define <*sym*> in all source files
-o <name>	output filename

- Example:

```
g++ -g -Wall -ansi -o hello hello.cpp
```