Translation Process

1. Macro Processor: **source.cpp** is expanded by the macro processor. All macros and **#define**'s are replaced by C++ code.
2. C++ Language Compiler: **source.cpp**, having all macro directives removed, is translated into object representation. Only identifiers that are defined elsewhere remain in plain text form.
3. Linkage Editor **source.o** is combined with C and C++ libraries and other **.o** files. External references are resolved. An executable file is created

Notes:

1. **-g** option directs the compiler to include information for the source debugger
2. **-c** option directs the compiler to produce a relocatable object file. External references are not resolved
3. **-E** option directs the compiler to stop after invoking the macro processor. To view the result of the macro processor phase:
   $ **g++ -E source.cpp > source.m**
4. **-o** option directs the linker to assign the name following the option to the executable file produced. For example,
   $ **g++ -o p09 p09.o**
   directs the linker to name the executable file **p09**.
5. The linker (linkage editor) is invoked when all the input files have a **.o** suffice - when all the input files are relocatable objects.

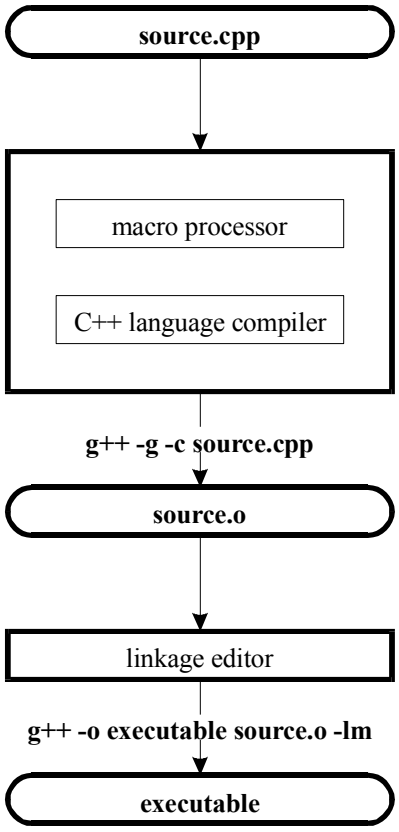

Figure 1. Translation Process

Programs consisting of multiple source files

1. Compile all source files.
   a. **$g++ -c –g p09.cpp**
   b. **$g++ -c –g Random09.cpp**
   c. **$g++ -c –g Card09.cpp**
   d. **$g++ -c –g Hand09.cpp**
   e. **$g++ -c –g Deck09.cpp**
2. Link all objects
   a. **$g++ -o p09 p09.oRandom09.o Card09.o Hand09.o Deck09.o -lm**

Function prototypes
1. Inform the compiler how to call a function.
2. Inform the compiler functions may be defined elsewhere
3. Validate function prototypes against actual function definitions

.h File organization.
1. **#ifndef** …
2. **#define** …
3. File description comment
4. Author identification comment
5. Standard C++ include files
6. Application include files
7. Namespaces
8. Class definition
9. **#endif**

.cpp File organization.
1. File description comment
2. Author identification comment
3. Standard C++ include files
4. Application include files
5. Namespaces
6. Member function implementations
7. C++ functions

Include Files
1. The include file contains a class that defines the abstract data type.
2. The **.cpp** file contains the implementation of member functions in the class.
3. Directs compiler how to call functions, number and type of parameters and return type

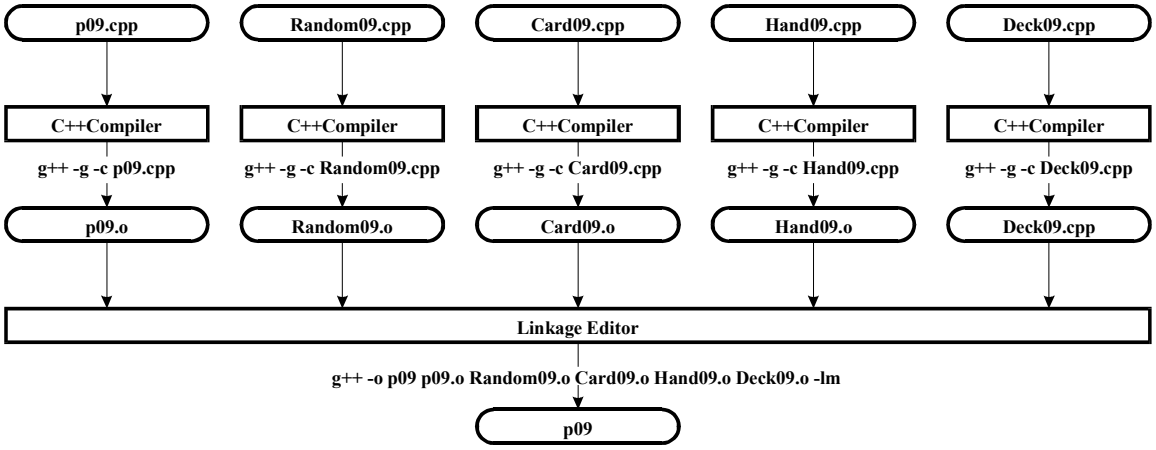

Figure 2. Program translation for a program containing multiple source files

Makefiles

1. Form
   target file: source files
   instructions
2. File **p09make** contains

```
obj =       p09.o Random09.o Card09.o Hand09.o Deck09.o
p09:        ${obj}
            g++ -o p09 ${obj} –lm
p09.o:      p09.cpp Deck09.h Hand09.h
            g++ -c –g p09.cpp
Random09.o: Random09.cpp Random09.h
            g++ -c –g Random09.cpp
Card09.o:   Card09.cpp Card09.h
            g++ -c –g Card09.cpp
Hand09.o:   Hand09.cpp Hand09.h Card09.h
            g++ -c –g Hand09.cpp
Deck09.o:   Deck09.cpp Deck09.h Card09.h Random09.h
            g++ -c –g Deck09.cpp
```

3. Invoking makefiles
   $ **make -f p09make**
4. File **p09make** notes:
   4.1. Lines beginning with a # sign are comments
   4.2. One or more UNIX tabs begin lines that are indented. There is no substitute for an UNIX tab. Code a UNIX tab using an editor on the Department of Computer Science computer. A tab in a PC or Windows editor does not translate reliably to an UNIX tab.

```
#---------------------------------------------------------------------
# File p09make creates executable file p09.  File p09 deals,
# sorts, and prints two poker hands.
#---------------------------------------------------------------------
# Author: Thomas R. Turner
# E-Mail: trturner@ucok.edu
# Date:   April, 2003
#---------------------------------------------------------------------
# Object files
#---------------------------------------------------------------------
obj     =       p09.o \
                Random09.o \
                Card09.o \
                Hand09.o \
                Deck09.o
#---------------------------------------------------------------------
#Bind the objects of executable file p09 together
#---------------------------------------------------------------------
p09:            ${obj}
                g++ -o p09 ${obj} -lm
#---------------------------------------------------------------------
# Compile file p09.cpp that processes command line arguments
#---------------------------------------------------------------------
p09.o:          p09.cpp Deck09.h Hand09.h
                g++ -c -g p09.cpp
#---------------------------------------------------------------------
# Compile file Random.cpp that implements class Random
#---------------------------------------------------------------------
Random09.o:     Random09.cpp Random09.h
                g++ -c -g Random09.cpp
#---------------------------------------------------------------------
# Compile file Card09.cpp that implements class Card
#---------------------------------------------------------------------
Card09.o:       Card09.cpp Card09.h
                g++ -c -g Card09.cpp
#---------------------------------------------------------------------
# Compile file Hand.cpp that implements class Hand
#---------------------------------------------------------------------
Hand09.o:       Hand09.cpp Hand09.h Card09.h
                g++ -c -g Hand09.cpp
#---------------------------------------------------------------------
# Compile file Deck09.cpp that implements class Deck
#---------------------------------------------------------------------
Deck09.o:       Deck09.cpp Deck09.h Card09.h Random09.h
                g++ -c -g Deck09.cpp
```

Figure 3. File **p09make**