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++ 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, *gdb*
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 p01 p01.o List01.o**
   directs the linker to name the executable file **p01**.
5. The linker (linkage editor) is invoked when all the input files have a .o suffice - when all the input files are relocatable objects.

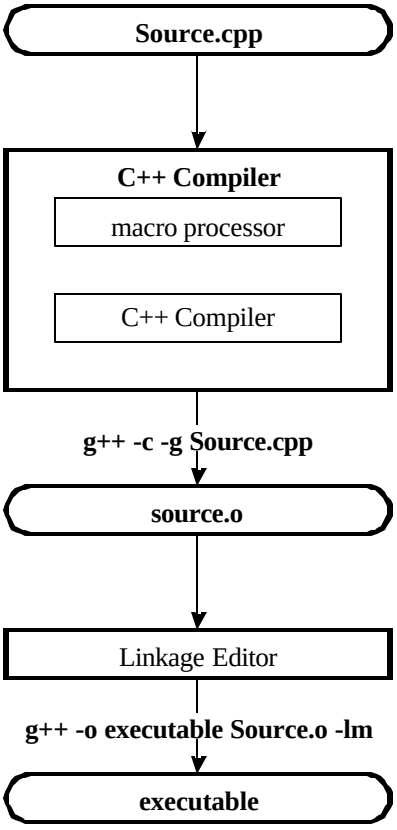

Figure 1. Translation Process

Programs consisting of multiple Source files

1. Compile all Source files.
   a. **$g++ -c -g p01.cpp**
   b. **$g++ -c -g Radix01.cpp**
   c. **$g++ -c -g List01.cpp**
2. Link all objects
   a. **$g++ -o p01 p01.o Radix01.o List01.o -lm**

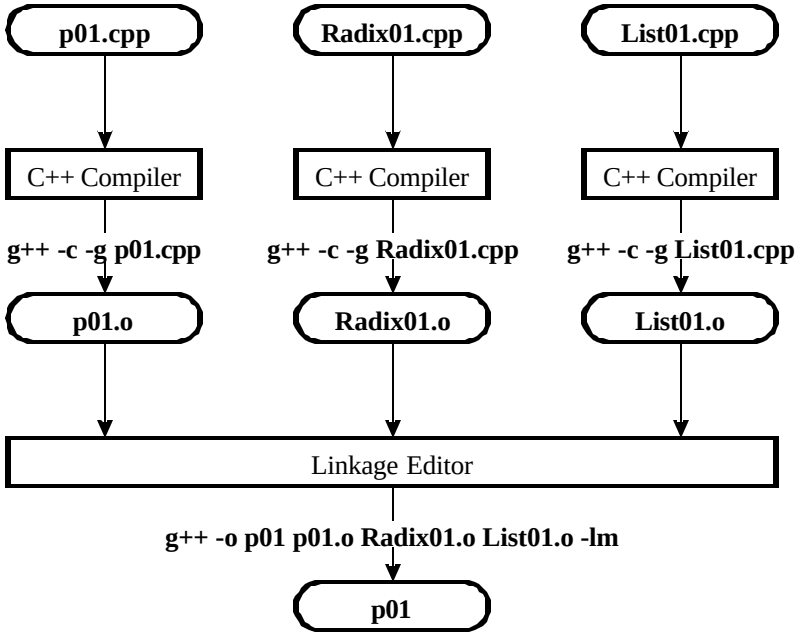

Figure 2. Translation Diagram

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

File organization.
1. File description comment
2. Author identification comment
3. Standard C++ Libraries
4. Application includes
5. Macro definitons
6. File global data
7. File functions

Include Files
1. The include file defines the interface to the abstract data type.
2. The **.cpp** file contains the implementation of the ADT
3. Included in file that exploits abstract data type.
4. Informs compiler that ADT functions are not defined in this file.
5. Directs compiler how to call functions, number and type of parameters and return type
6. Included in file that defines abstract data type. Validates functions in interface (.h file) against definition in **.cpp** file.

Makefiles

1. Form

```
target file:  Source files
              instructions
```

2. File p01make contains

```
p01:          p01.o Radix01.o List01.o
              g++ -o p01 p01.o Radix01.o List01.o -lm
p01.o:        p01.cpp Radix01.h
              g++ -c -g p01.cpp
Radix01.o:    Radix01.cpp Radix01.h List01.h
              g++ -c -g Radix01.cpp
List01.o:     List01.cpp List01.h
              g++ -c -g List01.cpp
```

3. Invoking makefiles

```
$ make -f p01make
```