Compiler Process

- **Compiler Stage**: All C++ language code in the .cpp file is converted into a lower-level language called Assembly language; making .s files.

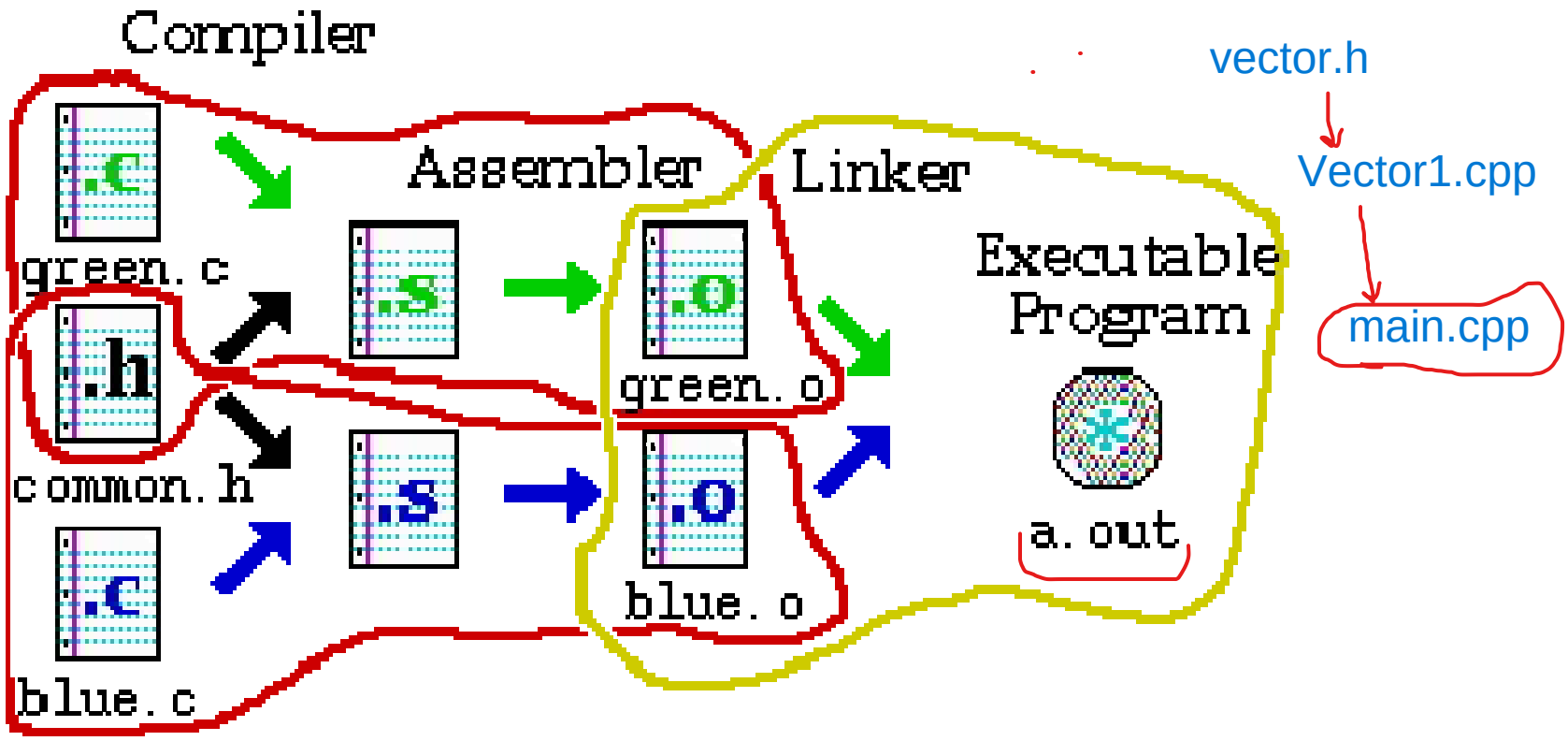
- **Assembler Stage**: The assembly language code made by the previous stage is then converted into object code which are fragments of code which the computer understands directly. An object code file ends with .o.

- **Linker Stage:** The final stage in compiling a program involves linking the object code to code libraries which contain certain "built-in" functions, such as cout. This stage produces an executable program, which is named a.out by default.

Makefiles

- Provide a way for separate compilation.
- Describe the dependencies among the project files.
- The `make` utility.

g++ file.cpp -o a.out
g++ file1.cpp file2.cpp file3.cpp -o a.out



Using makefiles

Naming:

- *makefile* or *Makefile* are standard
- other name can be also used

Running `make`

`make`

`make -f` *`filename`* – if the name of your file is not "makefile" or "Makefile"

Sample makefile

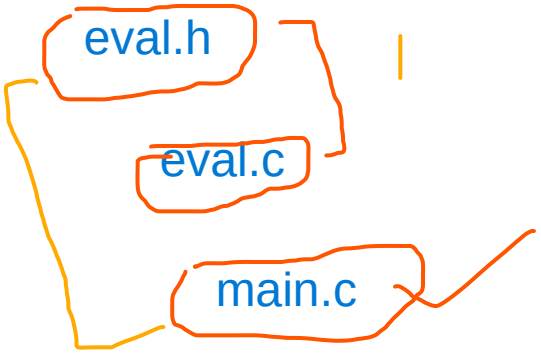
- Makefiles main element is called a *rule*:

```
target : dependencies
TAB  commands                    #shell commands
```

Example:

```
my_prog : eval.o main.o
   g++ -o my_prog eval.o main.o

eval.o : eval.c eval.h
   g++ -c eval.c
main.o : main.c eval.h
   g++ -c main.c
```



```
# -o to specify executable file name
# -c to compile only (no linking)
```

Variables

The old way (no variables)

```
my_prog : eval.o main.o
g++ -o my_prog eval.o main.o
eval.o : eval.c eval.h
g++ -c –g eval.c
main.o : main.c eval.h
g++ -c –g main.c
```

A new way (using variables)

```
C = g++
OBJS = eval.o main.o
HDRS = eval.h

my_prog : eval.o main.o
$(C) -o my_prog $(OBJS)
eval.o : eval.c
$(C) –c –g eval.c
main.o : main.c
$(C) –c –g main.c
$(OBJS) : $(HDRS)
```

Automatic variables

Automatic variables are used to refer to specific part of rule components.

```
target : dependencies
TAB   commands            #shell commands
```

```
eval.o : eval.c eval.h
    g++ -c eval.c
```

`$@` - The name of the target of the rule (`eval.o`).

`$<` - The name of the first dependency (`eval.c`).

`$^` - The names of all the dependencies (`eval.c eval.h`).

`$?` - The names of all dependencies that are newer than the target

`make` options

`make` options:

`-f` *filename* - when the makefile name is not standard

`-t` - (touch) mark the targets as up to date

`-q` - (question) are the targets up to date, exits with 0 if true

`-n` - print the commands to execute but do not execute them

/ `-t, -q,` and `-n,` cannot be used together /

`-s` - silent mode

`-k` - keep going – compile all the prerequisites even if not able to link them **!!**

Conditionals (directives)

Possible conditionals are:

`if  ifeq  ifneq  ifdef  ifndef`

All of them should be closed with `endif.`

Complex conditionals may use `elif` and `else.`

Example:

```
libs_for_gcc = -lgnu
normal_libs =
ifeq ($(CC),gcc)
  libs=$(libs_for_gcc)          #no tabs at the beginning
else
  libs=$(normal_libs)           #no tabs at the beginning
endif
```