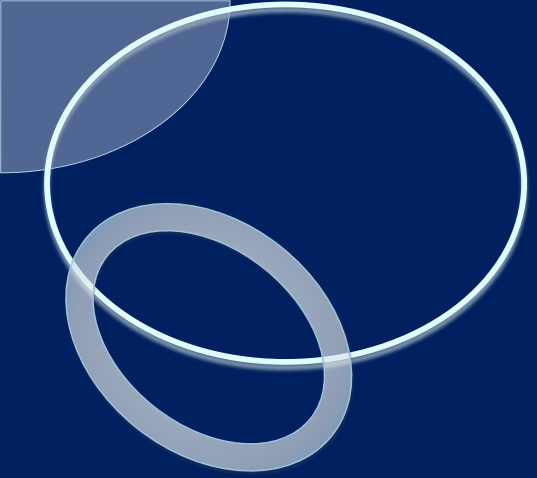

Makefile

Makefile

- **make** the program will look for a file named **makefile** in the current directory, and then execute it.
- If you have several makefiles, then you can execute them with the command:
  - make -f MyMakefile

Rules

- Compile from a command promt:
  - g++ main.cpp stack.cpp linked.cpp -o stack
- *A rule in the Makefile:*

stack: main.o stack.o linked.o

gcc main.o stack.o linked.o –o stack

Target: This is what this line of the make file will produce. It is followed by a ':'

Prerequisites. If these change, the target is out of date

s is a tab

I said it is a tab! It is not spaces. It is a tab!

This is the command that will make the target.

Multiple rules

- A make file can have many different rules.
- If you type **make <target>** that rule will be executed.
- If you simply type **make** the first target will be executed (which in turn will likely cause other targets to be executed).

Simple Makefile

```
all: queue

queue: main.o linkedList.o queue.o
	g++ main.o linkedList.o queue.o -o queue

main.o: main.cpp
	g++ -c main.cpp

linkedList.o: linkedList.cpp
	g++ -c linkedList.cpp

queue.o: queue.cpp
	g++ -c queue.cpp

clean:
	rm -rf *o queue
```

No rule? Thai is ok.
Prerequisite is queue.
Is there a rule for queue?

queue will first call the rules for main.o, linkedList.o and queue.o

If queue.cpp has been updated,
use g++ -c queue.cpp
to create queue.o

- make queue
- make

Variables and Comments

```
# Comments
#The following lines define a variable:
CC=gcc
CFLAGS=-c –Wall

#The following rule uses the variables we defined:
main.o: main.cpp
  $(CC) $(CFLAGS) main.cpp
```

Inference Rules

Inference rules generalize the build process so you don't have to give an explicit rule for each target.

Inference rules have the character “%” in the dependency line.

The “%” (**rule character**) is a wild card, matching zero or more characters.

Here is an inference rule for building .obj files from .c files:

```
%.obj : %.c
    $(CC) $(CFLAGS) –c $(.SOURCE)
```

Automatic Variables

$@ - The file name of the target of the rule.

$< The first prerequisite

http://www.gnu.org/software/make/manual/html_node/Automatic-Variables.html#Automatic-Variables

```
#Compile every .cpp file to create the
  corresponding .o file

%.o: %.cpp
        $(CC) -o $@ $<
```

Makefile using Variables

```
CC=gcc
CFLAGS=-c -Wall
LDFLAGS=
SOURCES=main.cpp linked.cpp queue.cpp
OBJECTS=$(SOURCES:.cpp=.o)
EXECUTABLE=main

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
   $(CC) $(LDFLAGS) $(OBJECTS) -o $@

.cpp.o: $(CC) $(CFLAGS)
   $< -o $@
```

Makefile

```
BIN  = c-  # name of thing to be built goes here
CC   = g++
# CFLAGS = -g
# CCFLAGS = -DCPLUSPLUS -g  # for use with C++ if file ext is .cc
#CFLAGS = -DCPLUSPLUS -g  # for use with C++ if file ext is .c

SRCS = parser.y parser.l token.h
OBJS = lex.yy.o parser.tab.o
LIBS = -lfl -lm

c-: $(OBJS)
	$(CC) $(CCFLAGS) $(OBJS) $(LIBS) -o c-

parser.tab.h parser.tab.c: parser.y
	bison -v -t -d parser.y

lex.yy.c: parser.l parser.tab.h
	flex parser.l
all:
	touch $(SRCS)
	make
clean:
	rm -f $(OBJS) c- lex.yy.c parser.tab.h parser.tab.c c-.tar parser.output core
tar:
	tar -cvf c-.tar $(SRCS) makefile
```

Put it all together and make a c- executable

Make parser.tab.h parser.tab.c from your parser.y

Make lex.yy.c from your parser.l and the parser.tab.h we just created

Clean out the generated files so we can start fresh.

How you can make a .tar file. (The submit program does this for you).