Software Construction

Makefile.simple

Simple makefile

```
game : main.o graphics.o world.o
	gcc -o game main.o graphics.o world.o

main.o : main.c graphics.h world.h
	gcc -c main.c

graphics.o : graphics.c world.h
	gcc -c graphics.c

world.o : world.c world.h
	gcc -c world.c

clean:
	rm -f game main.o graphics.o world.o
```

make0.pl

Simple Perl implementation of "make".

It parses makefile rules and stores them in 2 hashes.

Building is done with a recursive function.

```
$makefile_name = "Makefile";
if (@ARGV >= 2 && $ARGV[0] eq "-f") {
    shift @ARGV;
    $makefile_name = shift @ARGV;
}

parse_makefile($makefile_name);
push @ARGV, $first_target if !@ARGV;
build($_) foreach @ARGV;
exit 0;

sub parse_makefile {
    my ($file) = @_;
    open MAKEFILE, $file or die "Can not open $file: $!";
    while (<MAKEFILE>) {
        my ($target, $dependencies) = /(\S+)\s*:\s*(.*)/ or next;
        $first_target ||= $target;
        $dependencies{$target} = $dependencies;
        while (<MAKEFILE>) {
            last if !/^\t/;
            $build_command{$target} .= $_;
        }
    }
}

sub build {
    my ($target) = @_;
    my $build_command = $build_command{$target};
    die "*** No rule to make target $target\n" if !$build_command && !-e $target;
    return if !$build_command;
    my $target_build_needed = ! -e $target;
    foreach $dependency (split /\s+/, $dependencies{$target}) {
        build($dependency);
        $target_build_needed ||= -M _ $target > -M $dependency;
    }
    return if !$target_build_needed;
    print $build_command;
    system $build_command;
}
```

Makefile.variables

Simple makefile with variables & a comment

```
CC=gcc-4.9
CFLAGS=-O3 -Wall

game : main.o graphics.o world.o
    $(CC) $(CFLAGS) -o game main.o graphics.o world.o

main.o : main.c graphics.h world.h
    $(CC) $(CFLAGS) -c main.c

graphics.o : graphics.c world.h
    $(CC) $(CFLAGS) -c graphics.c

world.o : world.c world.h
    $(CC) $(CFLAGS) -c world.c

clean:
    rm -f game main.o graphics.o world.o
```

make1.pl

Add a few lines of code to make0.pl and we can handle variables and comments.

A good example of how easy some tasks are in Perl.

```
$makefile_name = "Makefile";
if (@ARGV >= 2 && $ARGV[0] eq "-f") {
    shift @ARGV;
    $makefile_name = shift @ARGV;
}

parse_makefile($makefile_name);
push @ARGV, $first_target if !@ARGV;
build($_) foreach @ARGV;
exit 0;

sub parse_makefile {
    my ($file) = @_;
    open MAKEFILE, $file or die "Can not open $file: $!\n";
    while (<MAKEFILE>) {
        s/#.*//;
        s/\$\((\w+)\)/$variable{$1}||''/eg;
        if (/^\s*(\w+)\s*=\s*(.*)$/) {
            $variable{$1} = $2;
            next;
        }
        my ($target, $dependencies) = /(\S+)\s*:\s*(.*)/ or next;
        $first_target ||= $target;
        $dependencies{$target} = $dependencies;
        while (<MAKEFILE>) {
            s/#.*//;
            s/\$\((\w+)\)/$variable{$1}||''/eg;
            last if !/^\t/;
            $build_command{$target} .= $_;
        }
    }
}

sub build {
    my ($target) = @_;
    my $build_command = $build_command{$target};
    die "*** No rule to make target $target\n" if !$build_command && !-e $target;
    return if !$build_command;
    my $target_build_needed = ! -e $target;
    foreach $dependency (split /\s+/, $dependencies{$target}) {
        build($dependency);
        $target_build_needed ||= -M  $target > -M $dependency;
    }
    return if !$target_build_needed;
    print $build_command;
    system $build_command;
}
```

Makefile.builtin_variables

Simple makefile with builtin variables

```
game : main.o graphics.o world.o
    $(CC) $(CFLAGS) -o $@ main.o graphics.o world.o

main.o : main.c graphics.h world.h
    $(CC) $(CFLAGS) -c $<

graphics.o : graphics.c world.h
    $(CC) $(CFLAGS) -c $*.c

world.o : world.c world.h
    $(CC) $(CFLAGS) -c $< -o $@

clean:
    rm -f game main.o graphics.o world.o
```

Makefile.implicit

Simple makefile with builtin variables relying on implict rules

```
game : main.o graphics.o world.o
	$(CC) $(CFLAGS) -o $@ $^

main.o : main.c graphics.h world.h

graphics.o : graphics.c world.h

world.o : world.c world.h

clean:
	rm -f game main.o graphics.o world.o
```

make2.pl

Add a few lines of code to make1.pl and we can handle some builtin variables and an implicit rule.

Another good example of how easy some tasks are in Perl.

```
$makefile_name = "Makefile";
if (@ARGV >= 2 && $ARGV[0] eq "-f") {
    shift @ARGV;
    $makefile_name = shift @ARGV;
}
%variable = (CC => 'cc', CFLAGS => '');
parse_makefile($makefile_name);
push @ARGV, $first_target if !@ARGV;
build($_) foreach @ARGV;
exit 0;

sub parse_makefile {
    my ($file) = @_;
    open MAKEFILE, $file or die "Can not open $file: $!";
    while (<MAKEFILE>) {
        s/#.*//;
        s/\$\((\w+)\)/$variable{$1}||''/eg;
        if (/^\s*(\w+)\s*=\s*(.*)$/) {
            $variable{$1} = $2;
            next;
        }
        my ($target, $dependencies) = /(\S+)\s*:\s*(.*)/ or next;
        $first_target = $target if !defined $first_target;
        $dependencies{$target} = $dependencies;
        while (<MAKEFILE>) {
            s/#.*//;
            s/\$\((\w+)\)/$variable{$1}||''/eg;
            last if !/^\t/;
            $build_command{$target} .= $_;
        }
    }
}

sub build {
    my ($target) = @_;
    my $build_command = $build_command{$target};
    if (!$build_command && $target =~ /(.*)\.o/) {
        $build_command = "$variable{CC} $variable{CFLAGS} -c \$< -o \$@\n";
    }
    die "*** No rule to make target $target\n" if !$build_command && !-e $target;
    return if !$build_command;
    my $target_build_needed = ! -e $target;
    foreach $dependency (split /\s+/, $dependencies{$target}) {
        build($dependency);
        $target_build_needed ||= -M _ $target > -M $dependency;
    }
    return if !$target_build_needed;
    my %builtin_variables;
    $builtin_variables{'@'} = $target;
    ($builtin_variables{'*'} = $target) =~ s/\.[^\.]*$//;
    $builtin_variables{'^'} = $dependencies{$target};
    ($builtin_variables{'<'} = $dependencies{$target}) =~ s/\s.*//;
    $build_command =~ s/\$(.)/$builtin_variables{$1}||''/eg;
    print $build_command;
    system $build_command;
}
```