Program Development Tools

Programming Editors

Many good editors for programming exist. Two are available most anywhere.

-*vi, emacs*

vi (pronounced "vee eye")

-Being able to use *vi* ensures that you will always have an editor available

-*vi* is available on all unix systems, other editors are not.

-no need to remove your fingers from the typing area of the keyboard

-*vi* stays out of your way: no menus and short commands

-you really only need to know about a dozen commands

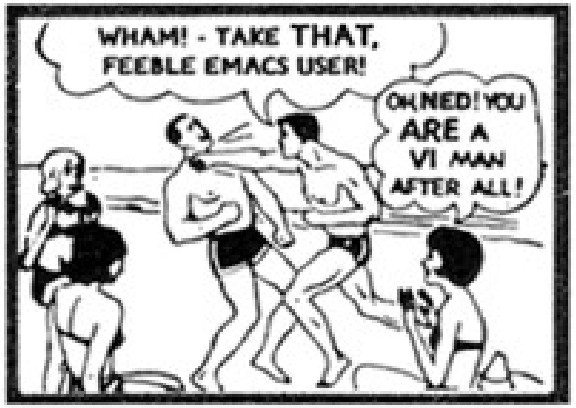
-Once those are mastered (30min), add to your repertoire

-global replacement

-split files vertically and horizontally

copy and paste between multiple open files

-"make program" and never leave the editor



-vi help sites

-http://staff.washington.edu/rells/R110/help_vi.html

-http://www.vmunix.com/~gabor/vi.html

-http://docs.freebsd.org/44doc/usd/12.vi/paper.html (Joy an Horton)

-http://thomer.com/vi/vi.html (fun way to waste time)

Program Development Tools

Vi commands

To begin editing a file:

`vi`	opens vi
`vi file_name`	opens vi on `file_name`

To close vi:

`:q`	quit (it will query you if changes were made)
`:q!`	quit without save
`:x`	save and exit
`<shift> zz`	save and exit

Writing a file:

`:w`	write out current file
`:w new_name`	write out current file as `new_name`
`:wq`	write file and quit (like “x” or “<shift>zz)

Read in a file:

`:r file_name`	Read in `file_name` into current one after the cursor

Program Development Tools

Vi commands – Modes (input, command)

Input Mode:

Insert text:

`i`	insert text just prior to cursor

Append text:

`a`	append text just after cursor

Open line:

`o`	open a new line below the current one
`O`	open a new line above the current one

Replace:

`r`	replace the character under cursor
`R`	replace continuously

Exit input mode:

`<esc>`

Program Development Tools

Vi commands – Modes (input, command)

Command mode:

In command mode we do everything else.....

-moving around in the file
-search and replace
-yank (copy) and put (paste)
-delete
-split screens

Command mode entered from input mode by <esc>

You enter vi in the command mode.

Program Development Tools

Vi commands – Modes (input, command)

Moving around in the file:

`h`	move one character left
`l`	move one character right
`k`	move one line up
`j`	move one line down
`<ctrl>u`	move up one page
`<ctrl>d`	move down one page
`w`	move forward 1 word
`b`	move back 1 word
`$`	move to end of line
`H`	move to line 1, column 1
`G`	move to last line, column 1
`:n`	go to line n

Program Development Tools

Vi commands – Modes (input, command)

Yank/paste, delete:

`yy`	yank current line
`p`	paste a line
`8yy`	yank next 8 lines (a subsequent paste will paste all 8 lines)
`x`	delete character
`cw`	change word
`dw`	delete word
`dd`	delete current line (line goes in paste buffer)
`J`	join next to line to current
`u`	undo (multi-level repeat)
`.`	repeat last command

Program Development Tools

Vi commands – Modes (input, command)

Search and Replace:

`/pattern`	search for `pattern` (forwards)
`?pattern`	search for `pattern` (backwards)
`n`	repeat last search

`:%s/old/new/g`

replace every occurrence of `old` with `new` in entire file

Program Development Tools

Vi commands – Modes (input, command)

Multiple screens:

`:sp new_file`	open split screen with `new_file` displayed
`:vsp new_file`	open vertically split screen with `new_file` displayed
`<ctrl>ww`	move between screens
`<ctrl>wn`	split existing window
`<ctrl>wv`	split existing window vertically

Program Development Tools

avr-gcc

-One of the GCC cross compilers
-Free, Open source, world class optimizing compiler
-Runs on Linux, Solaris, Mac, PC
-Available for almost any target uC/uP,
-See AVR Freaks site for avr-gcc help (www.avrfreaks.net)

uisp - Universal In System Programmer

-Allows programming of AVRs through the parallel port

avrdude - Another in System Programmer

-Allows programming of AVRs through the parallel port or USB

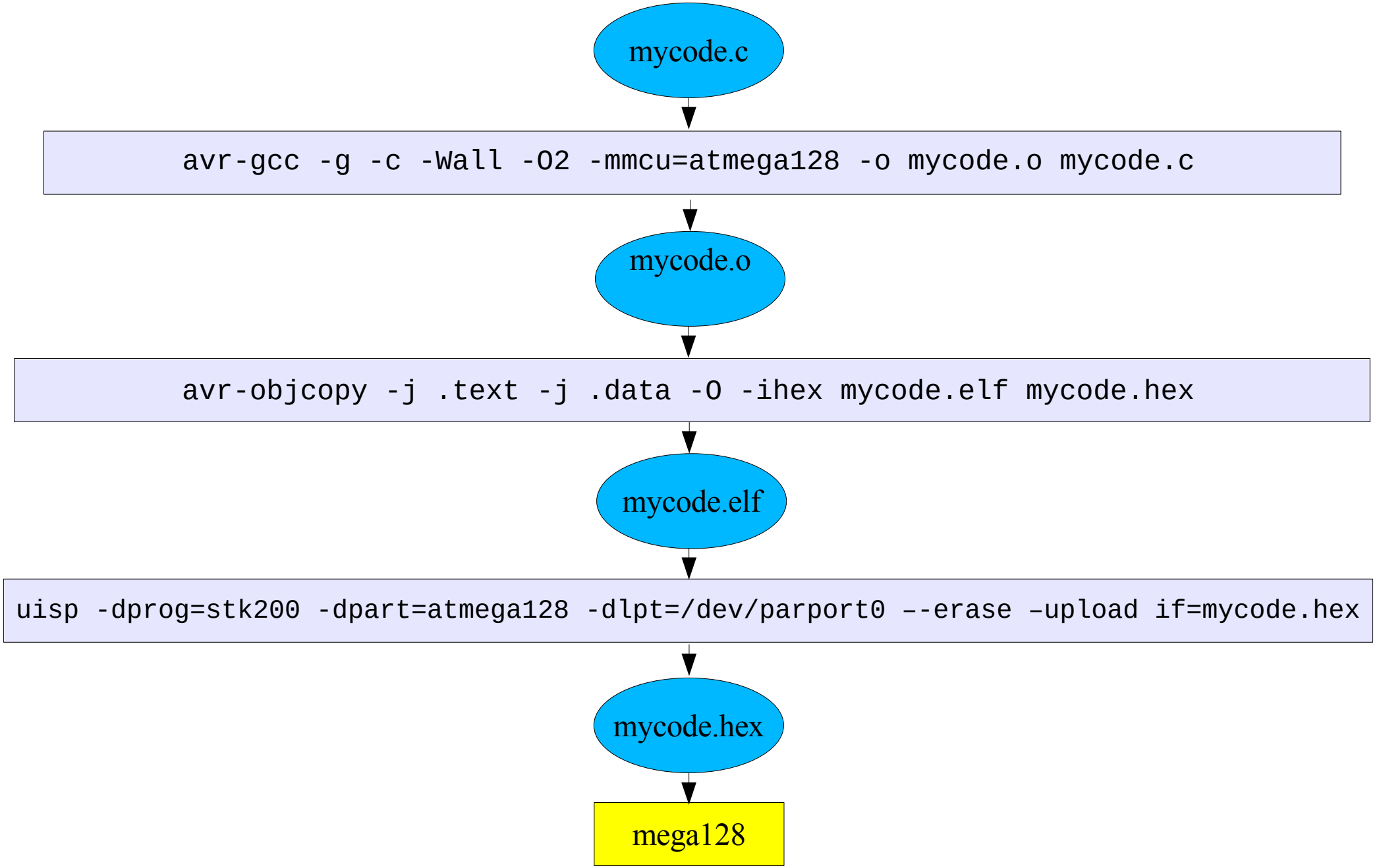
avr-objcopy

-copies and translates object files

avr-objdump

-diplays information from object files

Program Development Tools: avr-gcc/avr-objcopy/uisp



Program Development Tools

make

To automate and simplify the process of making an executable image for the microcontroller we use the `make` utility

`make` is the *puppetmaster* of your build environment. It...

-checks file dependencies and compiles only the files that require it
-executes other commands to generate other files (EEPROM images)
-is very general purpose, it works with any language and other tools
-lets us make one Makefile that can be used with many other programs

In more complex programming environments, make is an excellent way to manage the creation of an executable from 10's or 100's of files.

Program Development Tools

Makefiles

`make` reads a file called "Makefile" or "makefile" that describes the dependencies and actions to take to generate the executable image.

The syntax of a makefile *rule* is as follows:

```
target-list: dependency-list
   <hard tab> command-list
```

```
sr.o  :  sr.c
    avr-gcc -g -c -Wall -O2 -mmcu=atmega128 -o sr.o sr.c
```

This rule tells make that:

- `sr.o` is dependent on `sr.c`
- if `sr.c` changes, `sr.o` must be recreated by running the avr-gcc command

Program Development Tools - make

```
#a simple make file for sr.c
PRG  =  sr

all: $(PRG).elf

sr.o    :  sr.c
  avr-gcc -g -c -Wall -O2 -mmcu=atmega128 -o sr.o sr.c

sr.elf  :  sr.o
  avr-gcc -g -Wall -O2 -mmcu=atmega128 -Wl,-Map,$(PRG).map -o sr.elf sr.o

sr.hex  :  sr.elf
  avr-objcopy -j .text -j .data -O ihex sr.elf sr.hex

program :  $(PRG).hex
    uisp -dprog=stk200 -dpart=atmega128 -dlpt=/dev/parport0 --erase --upload\
    if=sr.hex

lst  :   $(PRG).lst

sr.lst  : sr.elf
    avr-objdump -h -S sr.elf > sr.lst

clean  :
    rm -rf *.o *.elf *.hex *.lst *.map
```

dependancy with no command

a phony target

Program Development Tools - make

```
PRG            = sr
OBJ            = $(PRG).o
MCU_TARGET     = atmega128
OPTIMIZE       = -O0    # options are 1, 2, 3, s
OBJCOPY        = avr-objcopy
OBJDUMP        = avr-objdump
CC             = avr-gcc

override CFLAGS        = -g -Wall $(OPTIMIZE) -mmcu=$(MCU_TARGET)
override LDFLAGS       = -Wl,-Map,$(PRG).map

#The automatic variables make recognizes:
#   $@        file name of target, i.e., left hand side of :
#   $<        name of first prerequisite
#   $?        name of all the prerequisites that are newer than the target
#   $^, $+    Names of all the prerequisites
#

all: $(PRG).elf lst text eeprom
##########################################################
#  The dependency for 'all" is the program.elf and three other rules.
#  The target "all" is known as a "phony target" as it is not a file
#  but a name used to have "make" do something for you.
##########################################################
```

Program Development Tools - make

```
$(PRG).elf: $(OBJ)
    $(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^
##########################################################
#  Make understands that to make an .obj you compile a .c
#  Thus we only need to say we want a .elf from an .obj
#  $@ is make shorthand for left hand side of  ":"
#  $^ is make shorthand for right hand side of ":"
##########################################################

clean:
    rm -rf *.o $(PRG).elf *.lst *.map
##########################################################
#  The target "clean" is another phony target that cleans
#  up all the extra files we make at each compile.
##########################################################

program: $(PRG).hex
    uisp -dprog=stk200 -dpart=atmega128 -dlpt=/dev/parport0 \
    –erase --upload if=$(PRG).hex
##########################################################
#  Target "program" depends on the .hex file which it
#  downloads to the target board using the parallel port.
##########################################################
```

Program Development Tools - make

```
lst:  $(PRG).lst
%.lst: %.elf
    $(OBJDUMP) -h -S $< > $@
#####################################################################
#  The target "lst" is another phony target. Its depends
#  on the .lst (list)  file.  The list file shows the
#  assembly language output of the compiler intermixed
#  with the C code so it is easier to debug and follow.
#  avr-objdump ($OBJDUMP) is the avr binutil tool we us to
#  get information from the .elf file.
#
#  $< is shorthand for source file for the single dependency
#####################################################################
```

Program Development Tools - make

```
#############################################################
# Following are rules for building the .text rom images
# This is the stuff that will be put in flash.
#############################################################
text: hex bin srec

hex:  $(PRG).hex
bin:  $(PRG).bin
srec: $(PRG).srec

%.hex: %.elf
    $(OBJCOPY) -j .text -j .data -O ihex $< $@
#############################################################
#  Take any .elf file and build the .hex file from it using
#  avr-objcopy.  avr-objcopy is used to extract the downloadable
#  portion of the .elf file to build the flash image.
#############################################################

%.srec: %.elf
    $(OBJCOPY) -j .text -j .data -O srec $< $@
#Make the Motorola S-record file

%.hex: %.elf
    $(OBJCOPY) -j .text -j .data -O binary $< $@
#Make a binary imagae also
```

Program Development Tools - make

```
#############################################################
# Following are rules for building the .eeprom images
# This is the stuff that will be put in EEPROM.
#############################################################
eeprom: ehex ebin esrec

ehex:  $(PRG)_eeprom.hex
ebin:  $(PRG)_eeprom.bin
esrec: $(PRG)_eeprom.srec

%_eeprom.hex: %.elf
    $(OBJCOPY) -j .eeprom --change-section-lma .eeprom=0 -O ihex $< $@
#############################################################
#This builds the EEPROM image from the .elf file for downloading.
#############################################################

%_eeprom.srec: %.elf
    $(OBJCOPY) -j .eeprom --change-section-lma .eeprom=0 -O srec $< $@
#############################################################
#This builds the S-record image from the .elf file if necessary.
#############################################################

%_eeprom.bin: %.elf
    $(OBJCOPY) -j .eeprom --change-section-lma .eeprom=0 -O binary $< $@
#############################################################
#This builds a binary image from the .elf file if necessary.
#############################################################
```