COP-3402 Systems Software
10.08 Thu

Reminder: **Project2 - Variables**

Grammar:

```
program
  = declaration* statement*

declaration
  = INT IDENTIFIER SEMI

statement
  = PRINT expression SEMI
  | READ IDENTIFIER SEMI
  | IDENTIFIER ASSIGN expression SEMI

expression
  = expression PLUS expression
  | expression MINUS expression
  | expression TIMES expression
  | expression DIVIDE expression
  | expression MOD expression
  | LPAREN expression RPAREN
  | NUMBER
  | IDENTIFIER
```

For Project0, our language can print arithmetic operations out
For Project1, we can print out arbitrary arithmetic operations, everything was still constant.
As for Project2, we add some support for variables into our language.We write a compiler that can implement variables.

At the machine level, everything is just memory addresses, there are no symbolic names for the data you want to work with. Our compiler will take this symbolic names and map them into memory locations.

- Use symbols as placeholders for values; y = x + 1
- y = 10 means assign 10 to y

Variable declaration: int x
Variable assignment: x = 20 / 3
Variable usage: print x + 4;

Reading from input:
SimpleC uses variables for input

read takes an integer from standard in
template.ll has *read_integer* method: reads string from standard input, converts to an integer and passes to us.
stdin(input), stdout(output), stderr(diagnostics)

Redirecting:

- Redirecting the output
  - ls > file_list.txt
- Redirecting the input
  - sort < file_list.txt
- "Pipes" chain multiple programs together
  - ls | sort
  - output of ls becomes the input of sort
- Redirecting standard err
  - find / 2> results.err

7

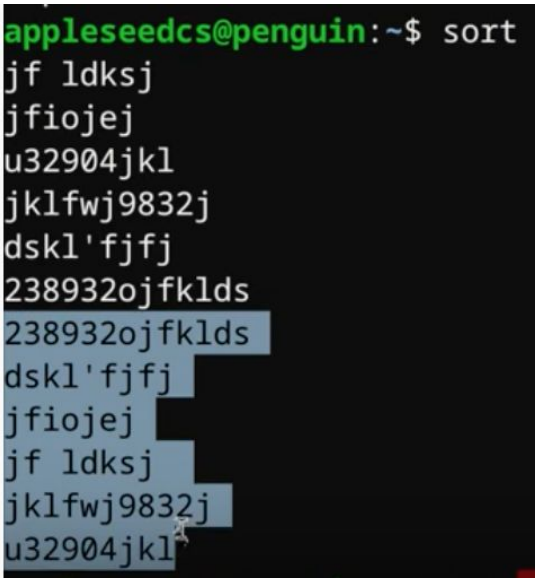
Redirecting ex.:

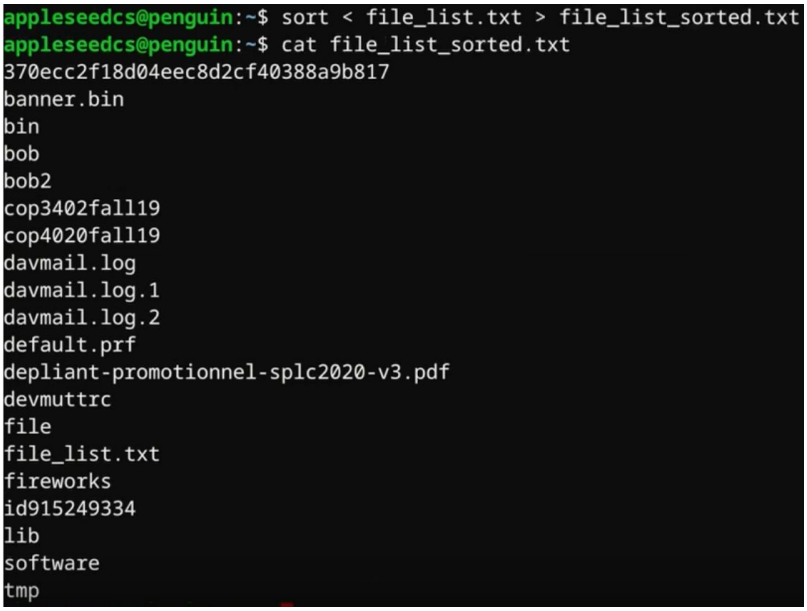
```
appleseedcs@penguin:~$ echo "JLFKDSjflkfjkldsjflkdsjfdks" > /dev/pts/1
JLFKDSjflkfjkldsjflkdsjfdks
```

/dev/pts/1 represents the file to the terminal window. When we write something to that file, it appears on the terminal window.

```
appleseedcs@penguin:~$ ls > file_list.txt
appleseedcs@penguin:~$ cat file_list.txt
370ecc2f18d04eec8d2cf40388a9b817
banner.bin
bin
bob
bob2
cop3402fall19
cop4020fall19
davmail.log
davmail.log.1
davmail.log.2
default.prf
depliant-promotionnel-splc2020-v3.pdf
devmuttrc
file
file_list.txt
fireworks
id915249334
lib
software
tmp
```

sort sorts out the inputs(6 in this example) you have entered in order: (Ctrl D after entering the inputs)





ls | *sort* takes the output of ls and turns it into the input of sort.

Now your compiler will now have to report errors. We have to declare variables, we will also have undeclared variable errors.

There are 3 variables given to you. *stdin, *stdout, *stderr. These are special files that represents standard i/o process. When we want to report an error, we do fprintf and the file std err.

Some Redirection Tricks:

- Redirect to nowhere
  - find / >/dev/null
- Redirect stderr to stdout
  - find > results.txt 2>&1
- Piping both stderr and stdout
  - find / |& sort
- Piping from cat instead of redirection stdin
  - "cat file.txt | sort" is equivalent to "sort < file.txt"
- Redirecting all three
  - ./prog < prog.in > prog.out 2> prog.err

For ex:
cd tests/cases/proj2
cat example3.simplec
int x;
x = 1 + y;

Another ex:

```
paul@dev:proj2$ cat example3.groundtrutherr
error: use of undeclared variable y
```

Go to syllabus/projects/template.h for two macros PROLOGUE/EPILOGUE and formatting to turn it into a string.

To print this; go to syllabus/projects/template_usage.c

```
paul@dev:projects$ clang -o template_usage template_usage.c
paul@dev:projects$ ./template_usage
```

```
paul@dev:projects$ ./template_usage > test.ll
paul@dev:projects$ clang -o test test.ll
paul@dev:projects$ ./test
please enter an integer
```

```
paul@dev:projects$ ./test > stdout.txt
please enter an integer
5432
paul@dev:projects$ cat stdout.txt
5432
```

An ex. on how to run:

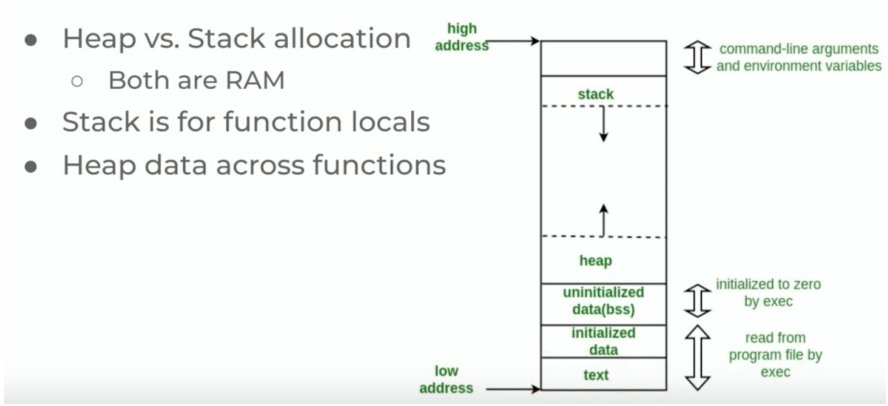
```
paul@dev:examples$ ./phony_run.sh ../tests/proj2/example2.ll < ../tests/proj2/e
xample2.in > ../tests/proj2/example2.out
paul@dev:examples$ cat ../tests/proj2/example2.out
5
```

If there is no difference between the ground truth and your output, it will not print anything:

```
paul@dev:examples$ diff ../tests/proj2/example2.groundtruth ../tests/proj2/exam
ple2.out
```

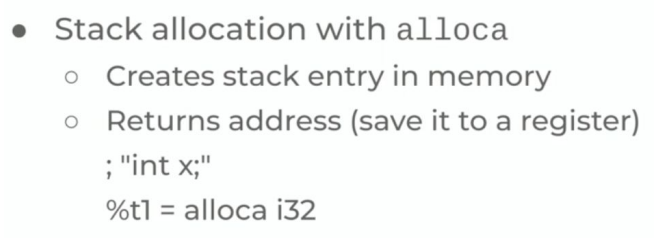
=> if you hit the tab in the command window, it will automatically complete the name of the file.
pwd returns the path you are in.

Memory layout:



Compiler assigns memory to each variable
Maintains mapping between names and locations
Creates new mapping on declaration
Compiler keeps track of this mapping
Compiler will look up the symbol table, find the address that we are using to store the value of that variable and put it into the memory location.

Allocating with LLVM IR:



- Load from memory with `load`
  - Loads value from memory at given location

```
; "print x;"
%t2 = load i32, i32* %t1
```

- Store from memory with `store`
  - Stores a value to memory at given location

```
; "x = 7;"
store i32 7, i32* %t1
```

Expecting to see the error on undeclared y:

```
paul@dev:projects$ cd tests/proj2/
paul@dev:proj2$ cat example3.simplec
int x;
x = 1 + y;
paul@dev:proj2$ cat example3.groundtrutherr
error: use of undeclared variable y
```