Variables

COP-3402 Systems Software
Paul Gazzillo

Variables Associate Symbols with Memory

- Inspired by algebra
  - Use symbols as placeholders for values
  - y = x + 1
- Imperative language variables are slightly different
  - y = 10 means assign 10 to y
  - Does not mean equality
    - x = 1; x = 2; is valid in SimpleC, but a contradiction in math
- Variables hold a value that can change (mutable)
  - Some languages have immutable variables

SimpleC with Variables

- SuperC has mutable variables
- Project 2's language adds
  - Variable declaration: `int x;`
  - Variable assignment: `x = 20 / 3;`
  - Variable usage: `print x + 4;`
- Complete example

```
int x;
x = 20 / 3;
print x + 4;
```

Demo: SimpleC Variable Semantics

Reading from Input

- SimpleC also uses variables for input
  - `read x;`
- A new keyword `read` takes an integer from standard in
  - template.ll now has a read_integer method
  - Prompts user, reads string from standard input and converts to an integer

Standard in, out, and err

- Unix processes always given three files
  - Running program is a process
  - System gives new process three files
  - stdin (input), stdout (output), stderr (diagnostics)
  - stdout/stderr differences are just a convention
- Which files used depends parent process
  - bash is parent of process run from command-line
  - All three default to the terminal file
    - This is why input and output is from/to terminal window
  - We can "redirect" or change these files from the default

Redirecting I/O

- In bash, stdin, stdout, stderr can be changed
- 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

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

Demo: Redirection and Piping

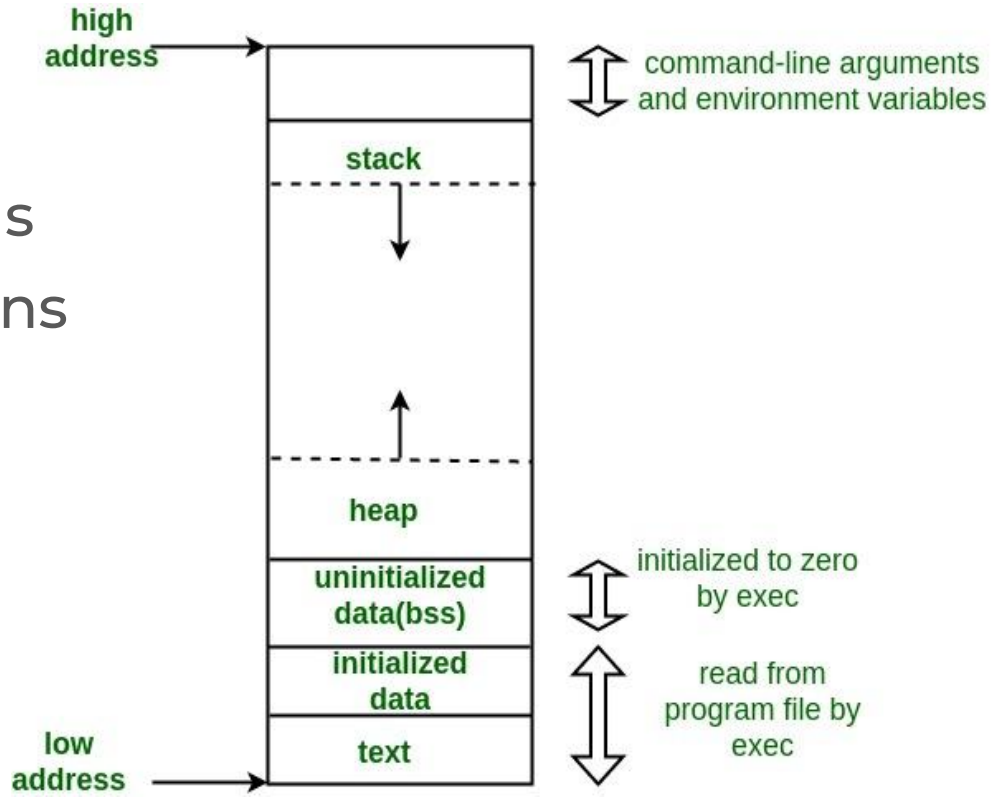
template.h

- New template now has read_integer
  - Backwards compatible with the previous template
- There is a header containing the template
  - PROLOGUE - all the stuff before your generated code
  - EPILOEG - all the stuff after
- Strings have been properly escaped for you

Demo: New Template

Memory Layout

- Heap vs. Stack allocation
  - Both are RAM
- Stack is for function locals
- Heap data across functions



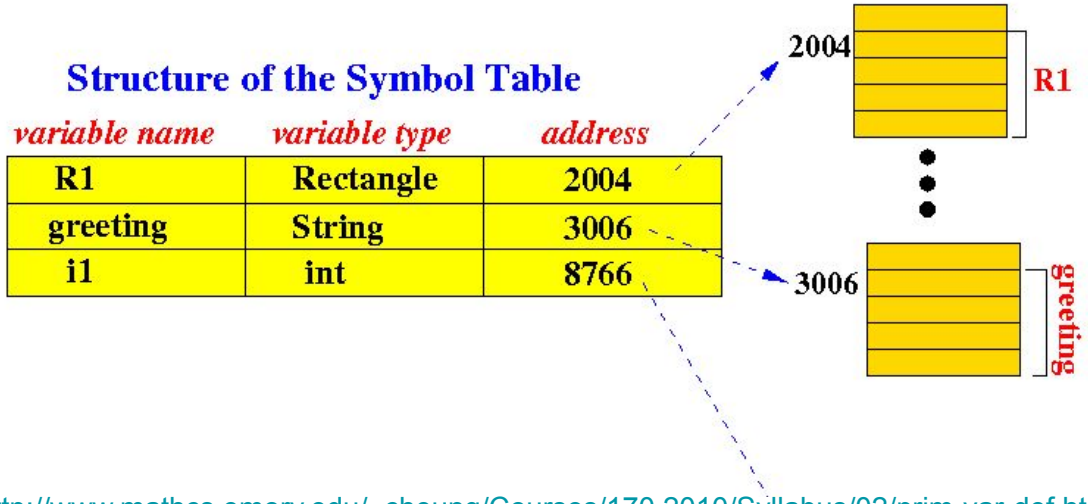
https://www.geeksforgeeks.org/memory-layout-of-c-program/

UCF

Demo: Recursion and the Stack

Symbol Table: Mapping Variables to Memory

- Compiler assigns memory to each variable
- Maintains mapping between names and locations
- Creates new mapping on declaration
- Refers to mapping when variables are used



http://www.mathcs.emory.edu/~cheung/Courses/170.2010/Syllabus/02/prim-var-def.html

Allocating Memory with LLVM IR

- Stack allocation with `alloca`
  - Creates stack entry in memory
  - Returns address (save it to a register)

```
; "int x;"
%t1 = alloca i32
```

Accessing Memory 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
```

Demo: Using the Symbol Table