CSCI 334: Principles of Programming Languages

Instructor: Dan Barowy

Lecture 11: Control Structures II

Instructor: Dan Barowy

**Williams**

---

**Computer Architecture**
(a really really fast introduction)

**Memory**

- main memory (typically GB worth)
  
<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>
  slow

- 1 byte

- registers (typically 32 to 512 bytes)
  
<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>
  fast

- 1 byte

---

**Instructions**

- add x y: add x to y, store in x
- sub x y: subtract y from x, store in x
- jmp x: jump to location x
- and x y: logical and, store in x
- or x y: logical or, store in x
- not x: logical not, store in x
- mov x y: copy memory from y into x

...
Sample x86 Assembly Program

```
L1:
.asciz "/bin/sh"
push ebp
mov ebp, esp
sub esp, 8
mov ebx, OFFSET FLAT:L1
mov DWORD PTR [ebp-8], ebx
mov DWORD PTR [ebp-4], 0
mov eax, 11
lea ecx, DWORD PTR [ebp-8]
mov edx, 0
int 0x80
leave
ret
```

BASIC

(Beginner’s All-purpose Symbolic Instruction Code)

- Invented in 1964 at Dartmouth College
- Implemented by undergrads!
- An “unstructured” programming language
- Inspired by FORTRAN (and similar in spirit)
- Intentionally simplified in order to appeal to beginners.
- As powerful as any other language (Turing complete).
- Wildly popular

Activity

- Write a Java/Python/pseudocode program that...
- Asks user for their name.
- Greets them with “Hello <name>”
- Asks them how many stars (‘*’) to print.
- Prints \( n \) stars
- Asks the user if they want more stars
- If yes, asks them for \( m \) more prints \( n \cdot m \), and asks again.
- Otherwise, quits.

Activity

```
10 INPUT "What is your name: "; U$
20 PRINT "Hello "; U$
30 INPUT "How many stars do you want: "; N
40 S$ = ""
50 FOR I = 1 TO N
60 S$ = S$ + "*"
70 NEXT I
80 PRINT S$
90 INPUT "Do you want more stars? "; A$
100 IF LEN(A$) = 0 THEN GOTO 90
110 A$ = LEFT$(A$, 1)
120 IF A$ <> "Y" AND A$ <> "y" THEN GOTO 160
130 INPUT "How many more stars? "; M
140 N = N + M
150 GOTO 40
160 PRINT "Goodbye "; U$
170 QUIT
```
Structured Programming

• Coined by Edsger Dijkstra
• “GOTO Statement Considered Harmful”
• Argued that GOTO made programming much harder to understand.
• “the quality of programmers is a decreasing function of the density of GOTO statements in the programs they produce.”

Dijkstra's argument

You have to debug this program. How much information do you need to keep in your head?

- Basically just a pointer.

Dijkstra's argument

Add a loop.

A little harder.

Need to remember a loop counter. (The rest you can determine by induction.)

Dijkstra's argument

http://www.malevole.com/mv/misc/killerquiz/
Dijkstra's argument

10  A = 1
20  B = 2
30  C = 3
40  D = 4
50  FOR I = 1 TO D
60  E = D
70  NEXT I
80  IF N < 10 THEN
90    D = 10
100 ELSE
110  D = 20
120 END IF
130 QUIT

Add a conditional.
Don't need to remember anything new.

Dijkstra's argument

10  A = 1
20  B = 2
30  C = 3
40  D = 4
50  FOR I = 1 TO D
60  E = D
70  NEXT I
80  IF N < 10 THEN
90    D = 10
100   GOTO 50
100 ELSE
110  D = 20
120 END IF
130 QUIT

Add a GOTO.
Need to remember all program values that might be updated.

Block Structured Programming

Only 3 building blocks for programs.

sequence conditional loop
statement true false true false statement

No GOTOs.

Block Structured Programming

Only 3 building blocks for programs.

sequence conditional loop
statement true false true false statement

Structured Program Theorem: Blocks are Turing-complete
Structured programs can be evaluated using a call stack

Stacks are made out of *activation records*

Stacks are used to track...
1. *which function* is being executed *now*,
2. *the parameters* to that function,
3. *the local variables* used in that function,
4. *temporary results* needed along the way,
5. *where to return* when done,
6. *where to put the result* when done,
7. *where to find non-local variables* (optional)

Those parts are named...
1. *which function*: top of the stack
2. *parameters*: actual parameters
3. *local variables*: local variables
4. *temporary results*: temporary storage
5. *where to return*: control link
6. *where to ret. result*: return result address
7. *non-local variables*: access link
Translation for C hackers...

activation record - stack frame

top of the stack - top of the stack

actual parameters - function parameters

local variables - local variables

temporary storage - local variables

control link - frame pointer

return result address - EAX

access link - does not exist in C!

What can a function return?

Stack frame layout
Activity

fun f x y = g x + g y
fun g x = x + 1
f 1 1