Lecture 20

Memory management

CS 241: Foundations of Sequential Programs
Winter 2018

Troy Vasiga et al
University of Waterloo

Using more than the stack

- most programming languages allow *dynamic memory allocation*
- that is, programs can request blocks of memory
- examples include
  - arrays declared dynamically
  - creating new elements to put into a linked structure
  - references/pointers to other memory locations
- in this module, we will discuss
  - where we can get these blocks of memory from
  - how to provide the basic operations to use this memory
  - how to make good decisions about using it "optimally" (almost)

Runtime memory management

Three things to consider:

- initialization
- allocation
- reclamation

Example: Stack usage

- initialization
- allocation
- reclamation

Example: Heap usage

- initialization
- allocation
- reclamation

Two words about the word “heap”

The arena

- What is it?
- Where do we put it?
- What could RAM look like?

A11P1 and A11P2 provided procedures

On A11P1, you have three primitive procedures:

- `int arena()`
- `int lw(int address)`
- `int sw(int address, int value)`

Tips to solve A11P1 and A11P2

- create the arena first (i.e., initialize), then use it
- a pointer is just an address
- an address is just an integer
- recursion is still your friend on A11P1
- carefully iterate on A11P2
- the solutions to A11P1 and A11P2 will be test input for later parts of the assignment

Four dimensions of arena allocation

1. fixed vs. variable size blocks
2. implicit vs. explicit allocation
3. implicit vs. explicit reclamation
4. language/implementation
   - Can pointers be relocated?

Fixed vs. variable size blocks

Implicit/explicit allocation and reclamation

Pointer Relocation

An easy case

Fixed size, explicit allocation, explicit reclamation, no pointer reallocation

- Initialize
- Allocate
- Reclaim

Available space list

Idea:

Initialize:

Allocate:

Reclaim:

Being slightly more clever

Idea:

Initialize:

Reclaim:

Allocate, with this change to reclaim:

A more self-contained implementation

Costs of this hybrid approach

- initialization
- allocation
- reclamation
- failure
- waste

A11P3 provided procedures

We abstract away from the `lw/sw` procedures from A11P1/A11P2 to build higher-level procedures.

- `int cons(int a, int b)`
- `int car(int p)`
- `int cdr(int p)`
- `int setcar(int p, int v)`
- `int setcdr(int p, int v)`
- `int snoc(int address)`

How to solve A11P3

- reading the input
- storing the input
- reading the stored values

Solving A11P4

`cons` and `snoc` are allocation and reclamation, respectively

The only difference is that they are two words (at least), rather than just one word.

Therefore, these are fixed blocks, but the size may not just be two.

Some sketches of A11P4 approaches

- `int cons(int a, int b)`
- `int car(int p)`
- `int cdr(int p)`
- `int setcar(int p, int v)`
- `int setcdr(int p, int v)`
- `int snoc(int address)`

Arena management with fixed size blocks

- running times of `initialize`, `allocate`, `reclaim`:
- languages which use fixed size blocks
- overall summary:

Arena management with variable size blocks

Overall summary: