15-122 Bootcamp: Memory & Pointers

Summer 2023

materials!

Today's Agenda

01
C0 Memory Recap

02
Code to Pointer Diagrams

03
C0 Visualizer

04
Pointer Diagrams to Code

https://tinyurl.com/n23pointers

materials!

01

C0 Memory Recap

How local vs. allocated memory work in C0 (REVIEW)

https://tinyurl.com/n23pointers

Local vs. Allocated Memory

- Variables in **local memory**...
  - Are primitive types only (`int, char, string, bool`)
  - Are created in functions, & are out-of-scope once they return

- Cells in **allocated memory**...
  - Need to be created with call to `alloc` function
  - Have unique **memory addresses**

Pointers in C0

- Pointers store memory addresses to cells in allocated memory
- Cells themselves could contain memory addresses: `int**` and `int****` and so on…
- Read & write to memory cells through pointers by **dereferencing** (`*p`)

Aliasing & Garbage Collection

- Remember the 122 mantra?

If we set two pointers equal to each other, they point to the same cell in allocated memory

- When there's no way to access a memory cell, they are **garbage collected**

Passing & Returning Pointers

- Pointers as parameters to functions are **aliases** of the original
    - The pointers reset to original address when function ends

```
void example_helper (int* a) {
    a = alloc(int);
    *a = 7;
}
void example () {
    int* b = alloc(int);
    *b = 5;
    example_helper(b);
    printf(“%d”, *b);  // prints 5 not 7
}
```

Passing & Returning Pointers

- Pointers as parameters to functions are **aliases** of the original
  - The pointers reset to original address when function ends

```
int* example_helper (int* a) {
     a = alloc(int);
     *a = 7;
     return a;
}
void example () {
     int* b = alloc(int);
     *b = 5;
     b = example_helper(b);
     printf(“%d”, *b);  // prints 7 this time
}
```

Structs in C0

```
struct goose_header {
    int height;
    string name;
}
```

- In C0, we can only have structs in allocated memory (no variables of type `struct goose_header`)

Quick Notes on typedef

- `typedef struct goose_header goose;`
- `goose* honk = alloc(goose);`
- `// typedef ___* chicken_t;`

02

Code to Pointer Diagrams

How we draw pointer diagrams from C0 code

TA Example

```c
typedef struct abc abc_t;
struct abc{
    int* a;
    int** c;
};

int main()
{
    abc_t* T = alloc(struct abc);           // OR alloc(abc_t); why?
    T->a = alloc(int);                      // what type is T->c?
    *(T->a) = 4;                            // what type is *(T->a)?
    T->c = alloc(int*);                     // what type is T->c?
    *(T->c) = T->a;
    // draw the memory diagram as it is on this line
    // make sure to mark any garbage collected memory with a pac-man
    return 1;
}
```

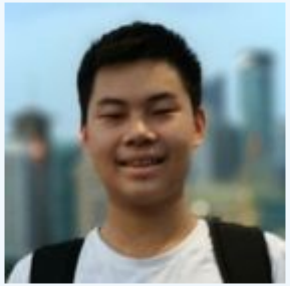
03

C0 Visualizer

A brief intermission on an awesome 15-122 tool

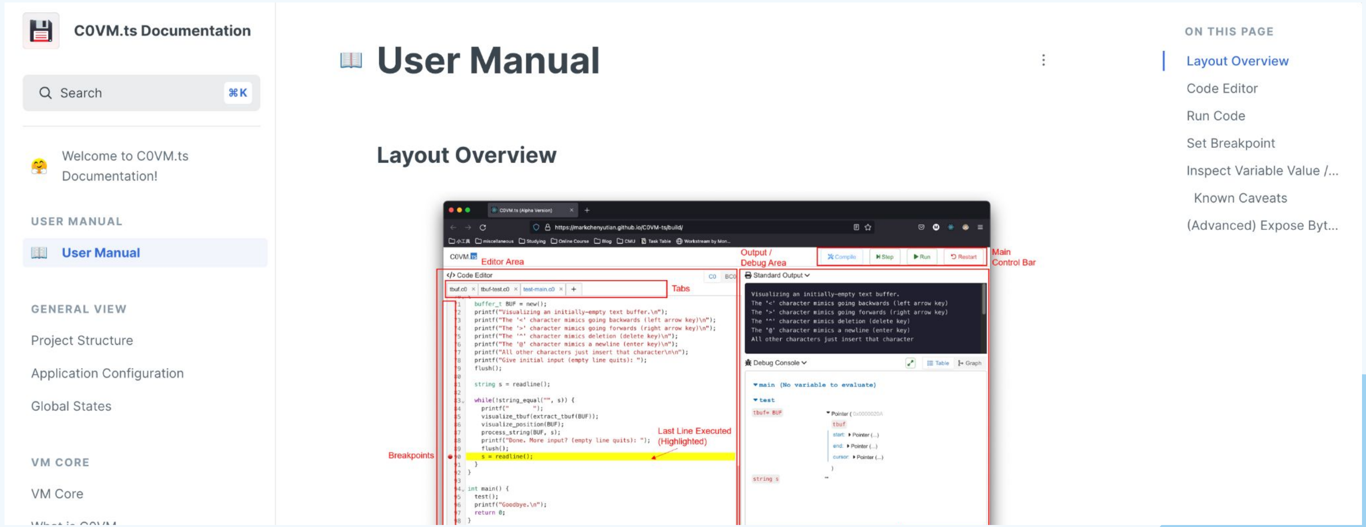
What's the C0 Visualizer?

- The C0 Visualizer is a tool built by 15-122 TAs that can help you debug and visualize your code in C0 & C1
- Found at https://cs122.andrew.cmu.edu/visualc0/
- Features include:
  - Built in code editor
  - Compiling & running your code
  - Setting breakpoints
  - Debug Console to inspect variable value/memory diagram


Yutian!

C0 Visualizer User Manual

- Access the full user manual at https://yutian-chen.gitbook.io/c0vm.ts-dev-documentation/user-manual/user-manual



C0VM.ts Documentation

Search ⌘K

Welcome to C0VM.ts Documentation!

USER MANUAL

User Manual

GENERAL VIEW

Project Structure

Application Configuration

Global States

VM CORE

VM Core

User Manual

Layout Overview

https://markchenyutian.github.io/C0VM-ts/build/

```
buffer_t BUF = new();
printf("Visualizing an initially-empty text buffer.\n");
printf("The '<' character mimics going backwards (left arrow key)\n");
printf("The '>' character mimics going forwards (right arrow key)\n");
printf("The '^' character mimics deletion (delete key)\n");
printf("The '@' character mimics a newline (enter key)\n");
printf("All other characters just insert that character\n\n");
printf("Give initial input (empty line quits): ");
flush();

string s = readline();

while(!string_equal("", s)) {
  printf("       ");
  visualize_tbuf(extract_tbuf(BUF));
  visualize_position(BUF);
  process_string(BUF, s);
  printf("Done. More input? (empty line quits): ");
  flush();
  s = readline();
}
}

int main() {
  test();
  printf("Goodbye.\n");
  return 0;
}
```

```
Visualizing an initially-empty text buffer.
The '<' character mimics going backwards (left arrow key)
The '>' character mimics going forwards (right arrow key)
The '^' character mimics deletion (delete key)
The '@' character mimics a newline (enter key)
All other characters just insert that character
```

```
▾main (No variable to evaluate)
▾test
tbuf* BUF        ▾Pointer ( 0x0000020A
                   tbuf
                   start:  ▸Pointer (...)
                   end:    ▸Pointer (...)
                   cursor: ▸Pointer (...)
                 )
string s         ""
```

ON THIS PAGE

C0 Visualizer Demo

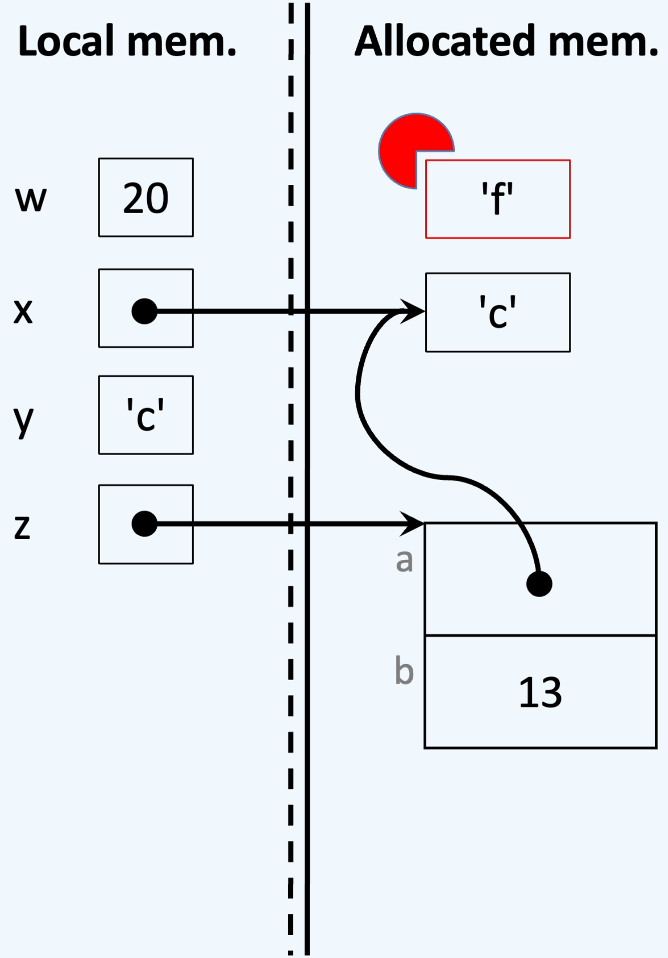
```
typedef struct abc abc_t;
struct abc{
    int* a;
    int** c;
};

int main()
{
    abc_t* T = alloc(struct abc);
    T->a = alloc(int);
    *(T->a) = 4;
    T->c = alloc(int*);
    *(T->c) = T->a;
    return 1;
}
```

(same code as earlier)

04

Pointer Diagrams to Code

How we deduce the code from a given pointer diagram



TA Example

Thanks