6.S096: Introduction to C/C++

Frank Li, Tom Lieber, Kyle Murray

Lecture 3: C Memory Management

January 15, 2013

Today…

- Computer Memory
- Pointers/Addresses
- Arrays
- Memory Allocation

Today…

- **Computer Memory**
- Pointers/Addresses
- Arrays
- Memory Allocation

Heap

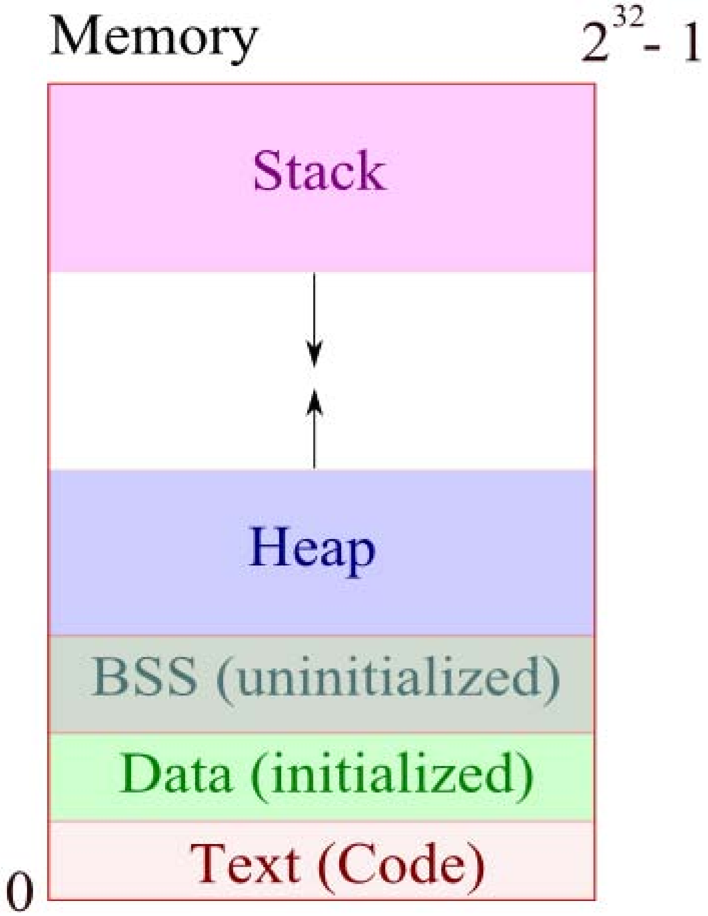
- Heap is a chunk of memory that users can use to dynamically allocated memory.

- Lasts until freed, or program exits.

Stack

- Stack contains local variables from functions and related book-keeping data. LIFO structure.
  - Function variables are pushed onto stack when called.
  - Functions variables are popped off stack when return.

Memory Layout



Memory Layout diagram courtesy of bogotobogo.com, and used with permission.

Call Stack

- Example: DrawSquare called from main()

```
void DrawSquare(int i){
    int start, end, …. //other local variables
    DrawLine(start, end);
}
void DrawLine(int start, int end){
    //local variables
    ...
}
```

Call Stack

- Example:

```
void DrawSquare(int i){
    int start, end, …
    DrawLine(start, end);
}
void DrawLine(int start,
int end){
    //local variables
    …
}
```

Lower address

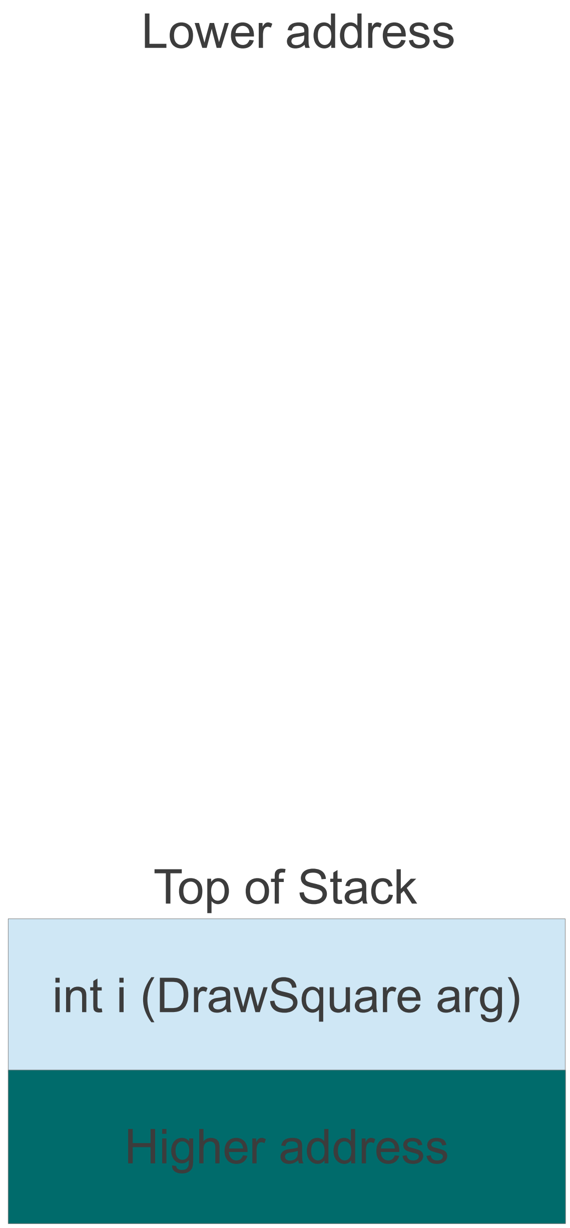
Top of Stack

Higher address

Call Stack

- Example:

```
void DrawSquare(int i){
    int start, end, …
    DrawLine(start, end);
}
void DrawLine(int start,
int end){
    //local variables
    …
}
```



Lower address

Top of Stack

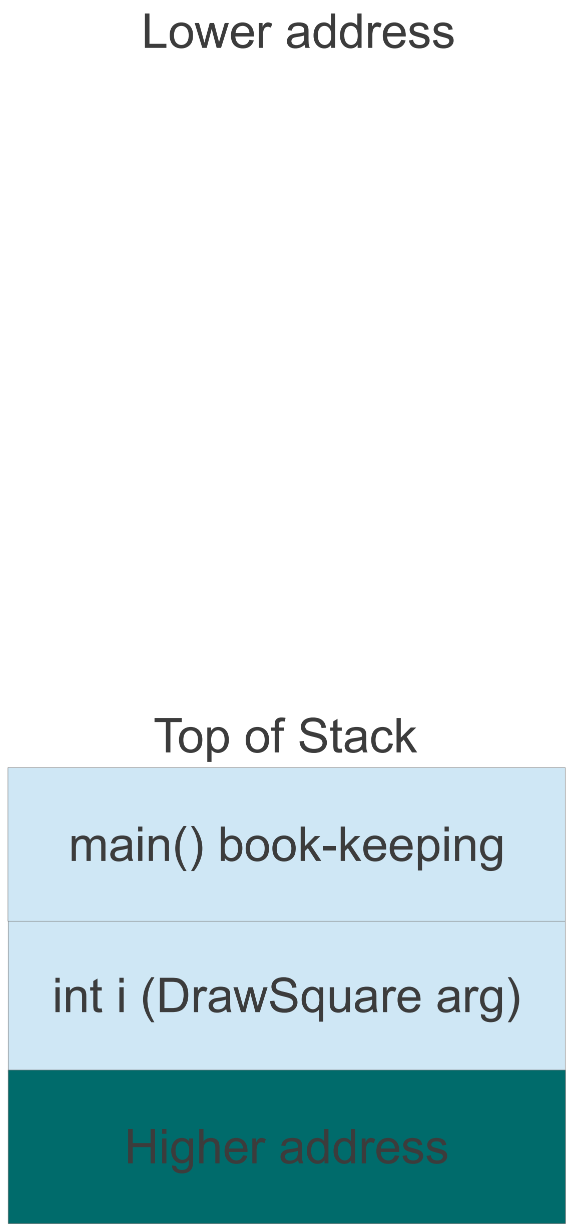
int i (DrawSquare arg)

Higher address

Call Stack

- Example:

```
void DrawSquare(int i){
    int start, end, ...
    DrawLine(start, end);
}
void DrawLine(int start,
int end){
    //local variables
    ...
}
```



Lower address

Top of Stack

main() book-keeping

int i (DrawSquare arg)

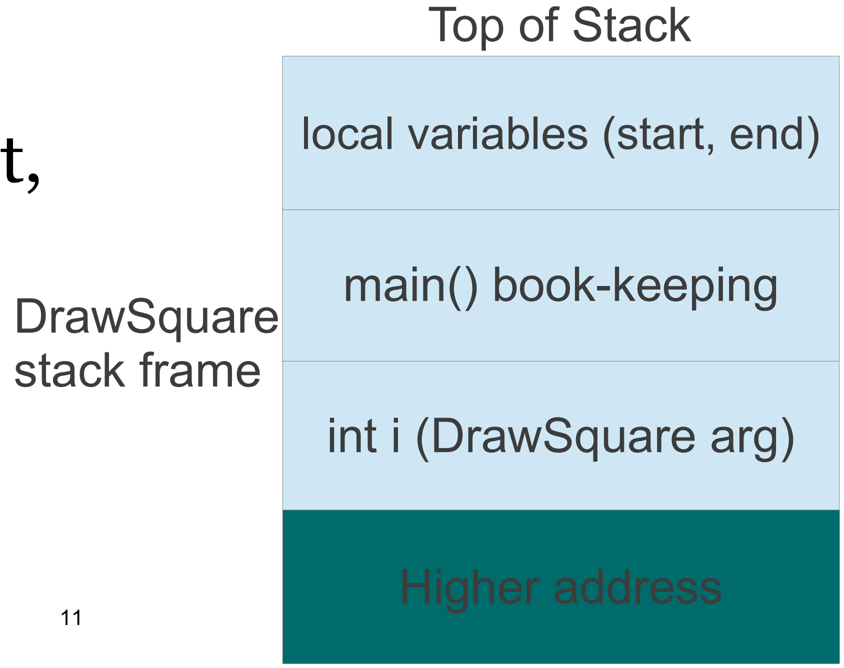
Higher address

Call Stack

Lower address

- Example:

```
void DrawSquare(int i){
    int start, end, …
    DrawLine(start, end);
}
void DrawLine(int start,
int end){
    //local variables
    …
}
```



Top of Stack

local variables (start, end)

main() book-keeping

int i (DrawSquare arg)

Higher address

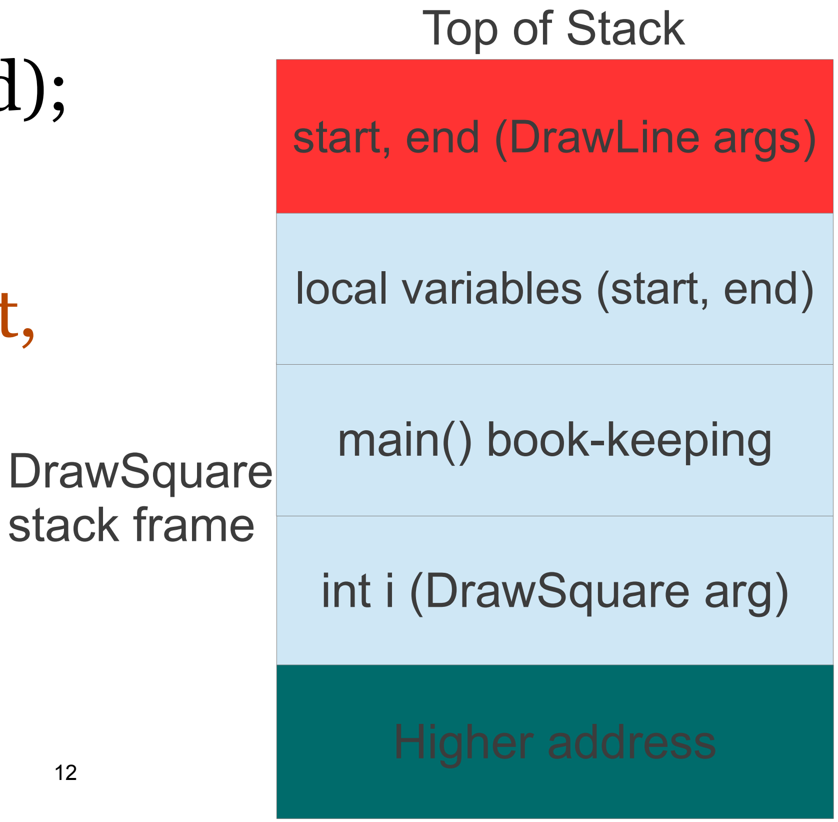
DrawSquare stack frame

Call Stack

- Example:

```
void DrawSquare(int i){
    int start, end, ...
    DrawLine(start, end);
}
void DrawLine(int start,
int end){
    //local variables
    ...
}
```

Lower address



Top of Stack

start, end (DrawLine args)

local variables (start, end)

main() book-keeping

int i (DrawSquare arg)

DrawSquare stack frame

Higher address

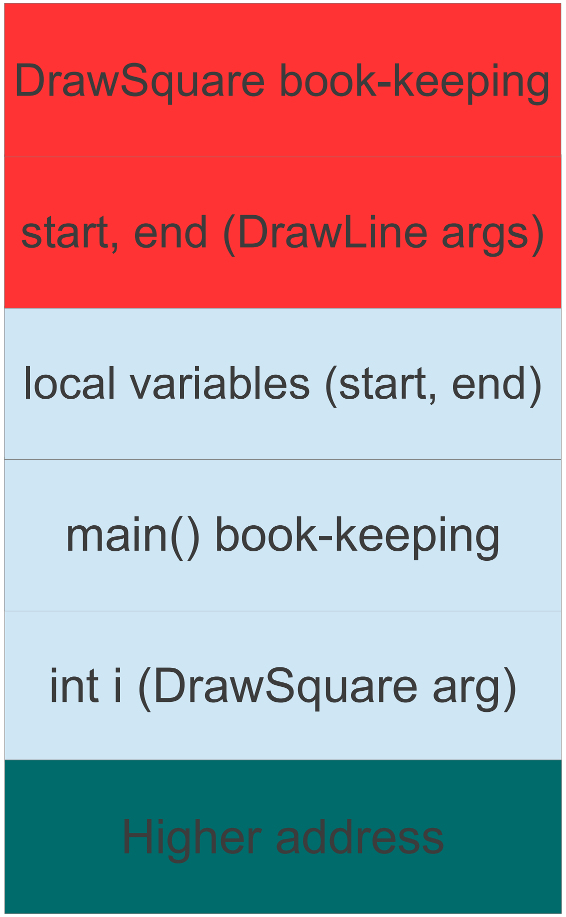
Call Stack

- Example:

```
void DrawSquare(int i){
    int start, end, ...
    DrawLine(start, end);
}
void DrawLine(int start,
int end){
    //local variables
    ...
}
```

Lower address

Top of Stack



DrawSquare book-keeping

start, end (DrawLine args)

local variables (start, end)

main() book-keeping

int i (DrawSquare arg)

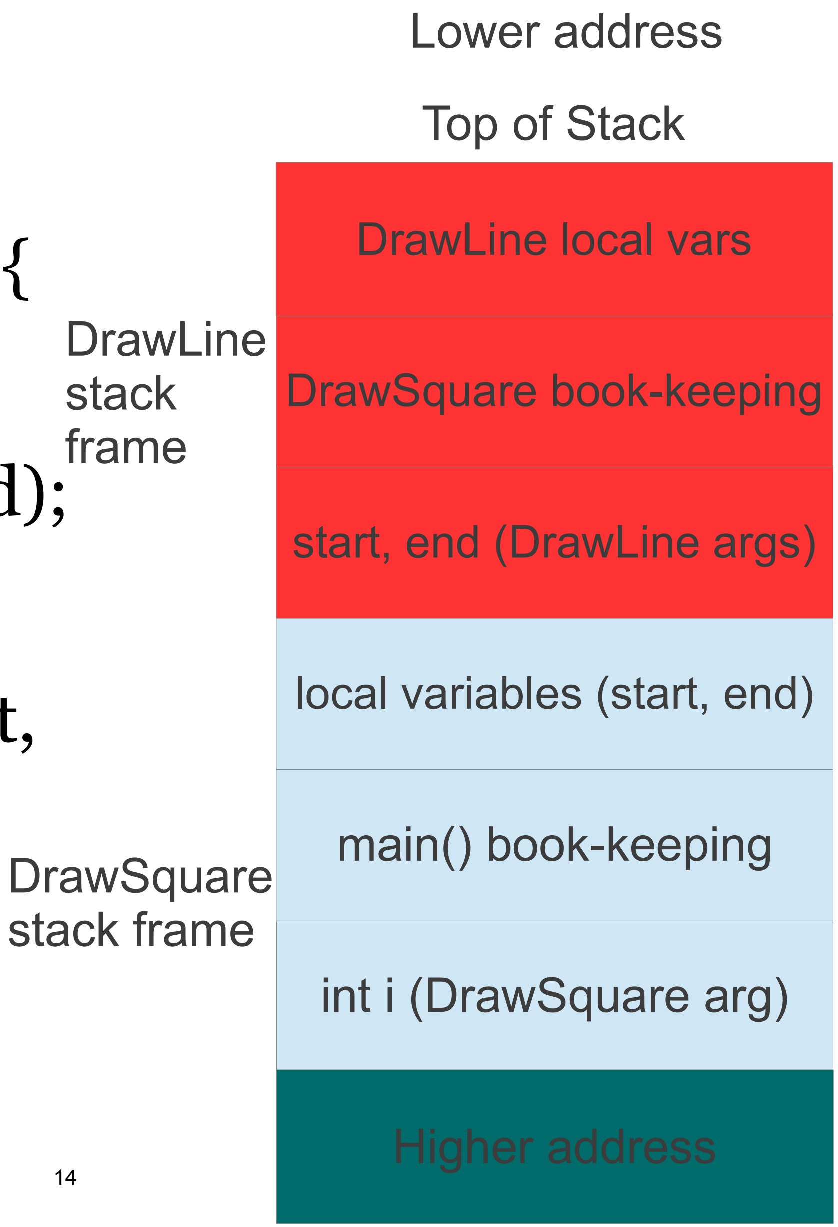
Higher address

DrawSquare stack frame

Call Stack

- Example:

```
void DrawSquare(int i){
      int start, end, ...
      DrawLine(start, end);
}
void DrawLine(int start,
int end){
      //local variables
      ...
}
```



Lower address

Top of Stack

DrawLine local vars

DrawSquare book-keeping

start, end (DrawLine args)

local variables (start, end)

main() book-keeping

int i (DrawSquare arg)

Higher address

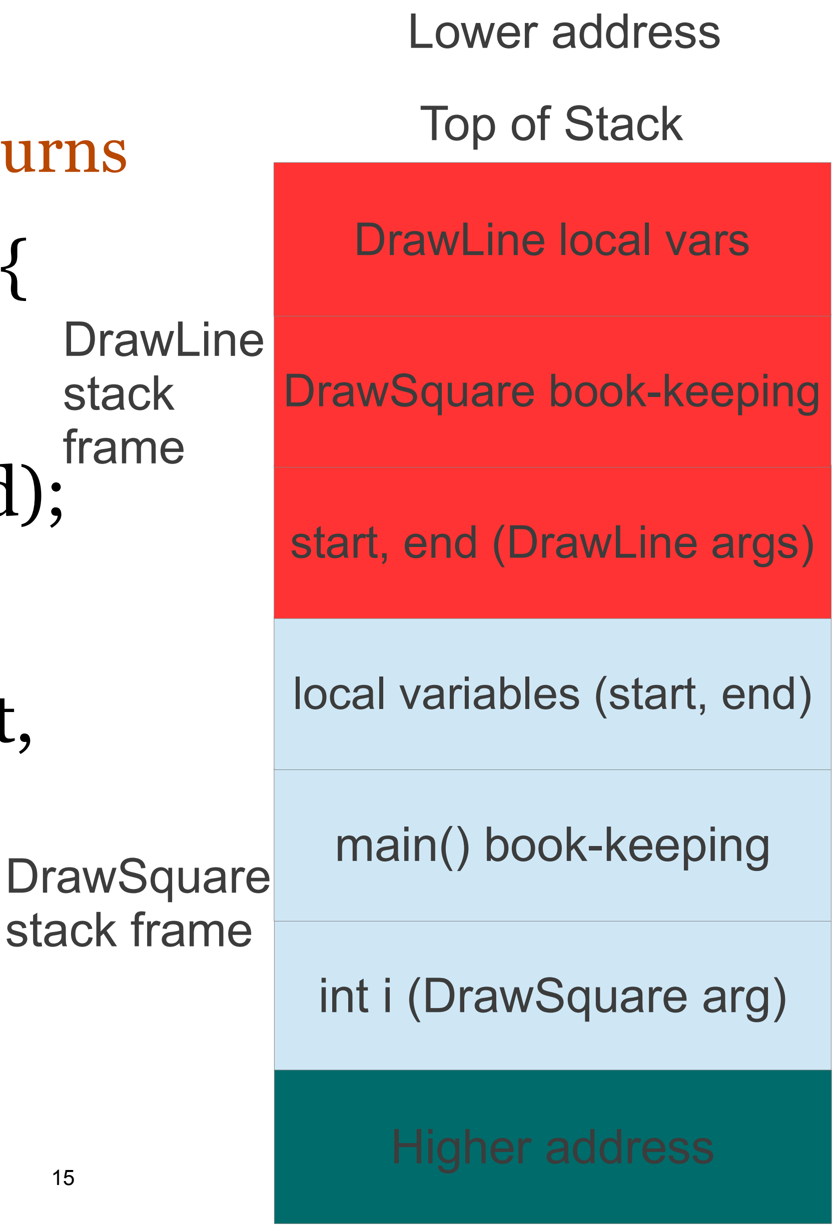
DrawLine stack frame

DrawSquare stack frame

Call Stack

- Example: DrawLine returns

```
void DrawSquare(int i){
    int start, end, ...
    DrawLine(start, end);
}
void DrawLine(int start,
int end){
    //local variables
    ...
}
```



Lower address

Top of Stack

DrawLine stack frame

DrawLine local vars

DrawSquare book-keeping

start, end (DrawLine args)

DrawSquare stack frame

local variables (start, end)

main() book-keeping

int i (DrawSquare arg)

Higher address

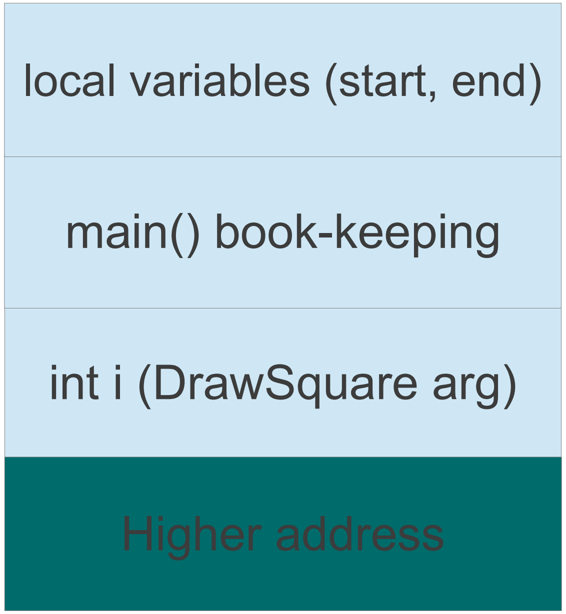
Call Stack

Lower address

- Example: DrawLine returns

```
void DrawSquare(int i){
    int start, end, ...
    DrawLine(start, end);
}
void DrawLine(int start,
int end){
    //local variables
    ...
}
```

Top of Stack



local variables (start, end)

main() book-keeping

int i (DrawSquare arg)

Higher address

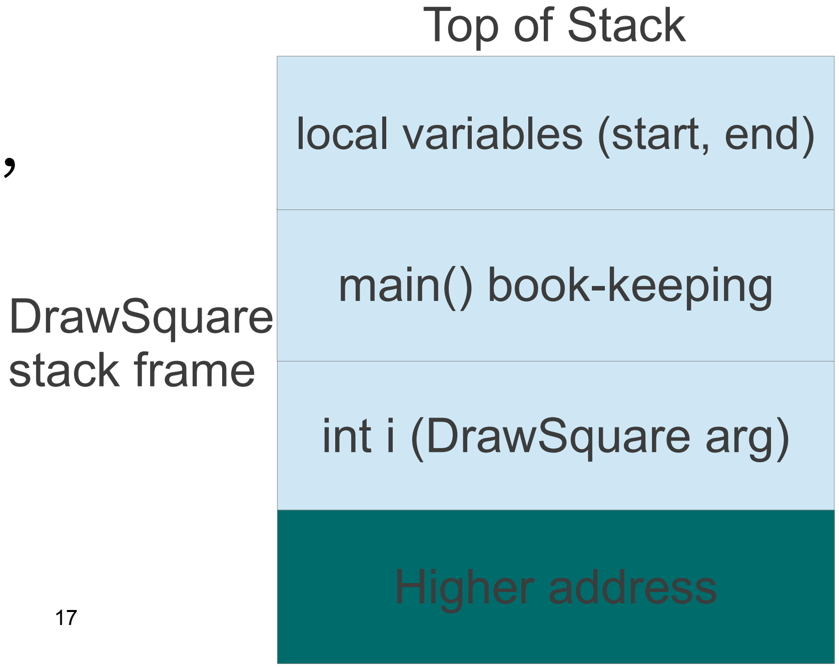
DrawSquare stack frame

Call Stack

Lower address

- Example: DrawSquare returns

```
void DrawSquare(int i){
      int start, end, …
      DrawLine(start, end);
}
void DrawLine(int start,
int end){
      //local variables
      ...
}
```



Call Stack

- Example: DrawSquare returns

```
void DrawSquare(int i){
    int start, end, ...
    DrawLine(start, end);
}
void DrawLine(int start,
int end){
    //local variables
    ...
}
```

Lower address

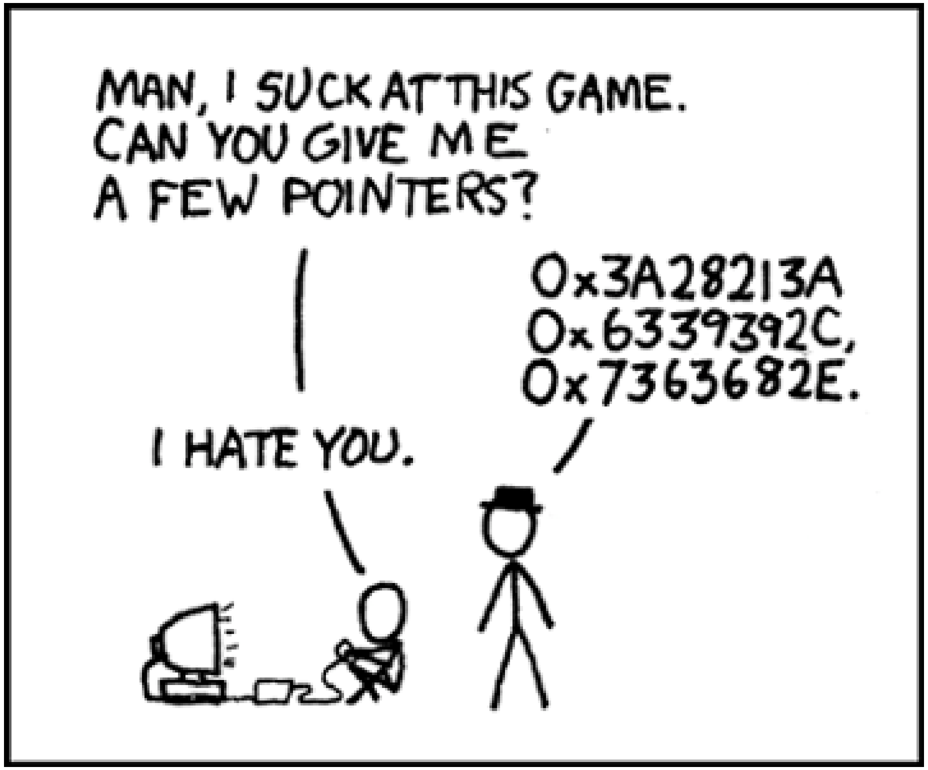
Top of Stack

Higher address

Today…

- Computer Memory
- **Pointers/Addresses**
- Arrays
- Memory Allocation

Pointers and Addresses


MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A 0x6339392C, 0x7363682E.
I HATE YOU.

Courtesy of xkcd at http://xkcd.com/138/, available under a CC by-nc license

Addresses

- Each variable represents an address in memory and a value.

- Address: &*variable* = address of variable

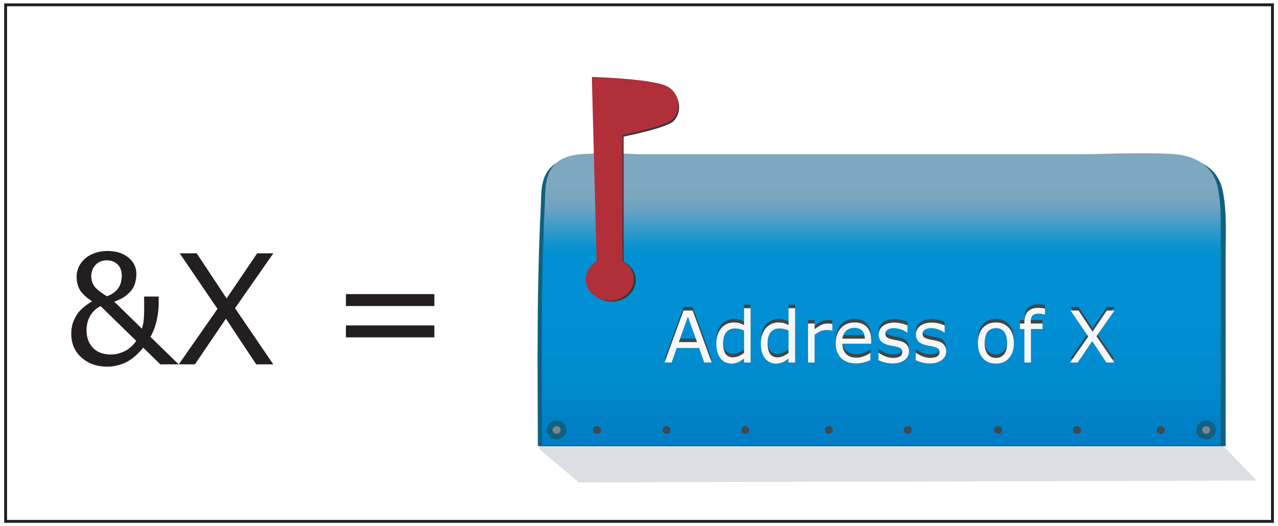


Image by MIT OpenCourseWare.

Pointers

A pointer is a variable that “points” to the block of memory that a variable represent.

- Declaration: data_type *pointer_name;
- Example:

```
char x = 'a';
char *ptr = &x; // ptr points to a char x
```

Pointers

A pointer is a variable that “points” to the block of memory that a variable represent.

- Declaration: data_type *pointer_name;
- Example:

  char x = 'a';

  char *ptr = &x; // ptr points to a char x
- Pointers are integer variables themselves, so can have pointer to pointers: char **ptr;

Data type sizes

Name	Description	Size*	Range*
char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1byte	true or false
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)

Dereferencing = Using Addresses

- Also uses * symbol with a pointer. Confusing? I know!!!

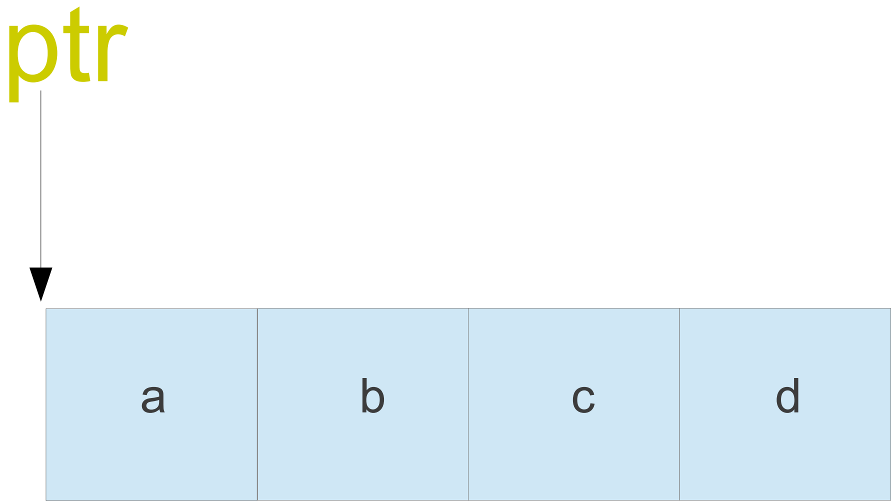
- Given pointer ptr, to get value at that address, do: *ptr

  -
    ```
    int x = 5;
    int *ptr = &x;
    *ptr = 6; // Access x via ptr, and changes it to 6
    printf(“%d”, x); // Will print 6 now
    ```

- Can use void pointers, just cannot dereference without casting

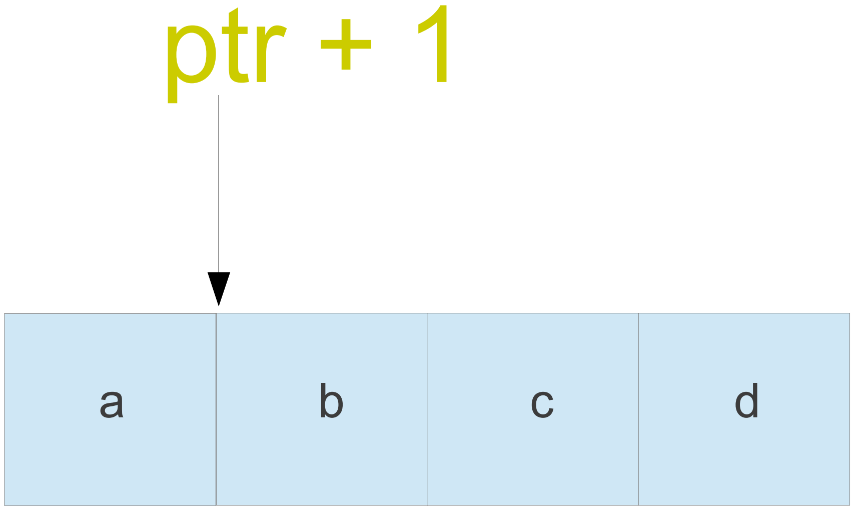
Pointer Arithmetic

- Can do math on pointers
  - Ex: char* ptr



Pointer Arithmetic

- Can do math on pointers
  - Ex: char* ptr

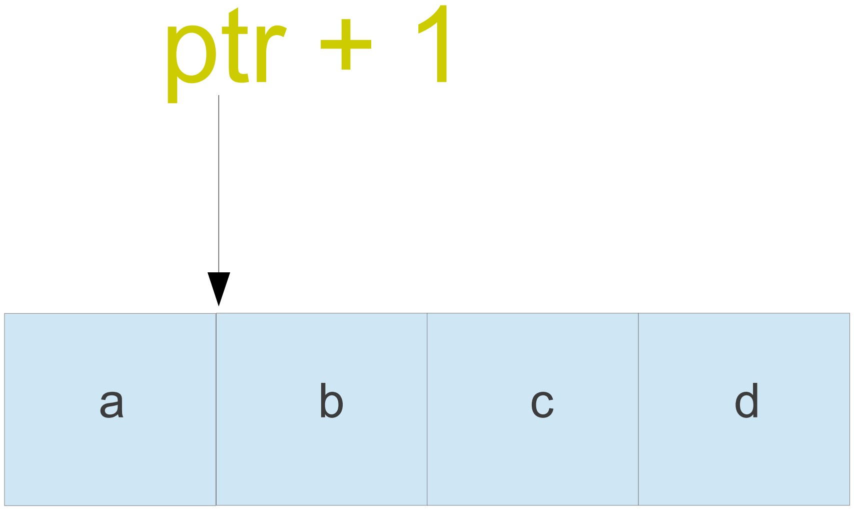


ptr + 1

a b c d

Pointer Arithmetic

- Can do math on pointers
  - Ex: char* ptr



ptr+i has value: ptr + i * sizeof(data_type of ptr)

Pointer Arithmetic

- Can do math on pointers
  - p1 = p2: sets p1 to the same address as p2
  - Addition/subtraction:
    - p1 + c , p1 - c
  - Increment/decrement:
    - p1++, p1--

Why use pointers? They so confuzin…

- Pass-by-reference rather than value.

```
void sample_func( char* str_input);
```

- Manipulate memory effectively.

- Useful for arrays (next topic).

Today…

- Computer Memory
- Pointers/Addresses
- **Arrays**
- Memory Allocation

C Arrays (Statically Allocated)

- Arrays are really chunks of memory!

- Declaration:
    - Data_type array_name[num_elements];

- Declare array size, cannot change.

C Arrays (Statically Allocated)

- Can initialize like:
  - int data[] = {0, 1, 2}; //Compiler figures out size
  - int data[3] = {0, 1, 2};
  - int data[3] = {1}; // data[0] = 1, rest are set to 0
  - int data[3]; //Here, values in data are still junk

    data[0] = 0;

    data[1] = 1;

    data[2] = 2;

Array and Pointers

- Array variables are pointers to the array start!
  - char *ptr;

    char str[10];

    ptr = str; //ptr now points to array start

    ptr = &str[0]; //Same as above line

- Array indexing is same as dereferencing after pointer addition.
  - str[1] = ‘a’ is same as *(str+1) = ‘a’

C-Style Strings

- No string data type in C. Instead, a string is interpreted as a null-terminated char array.
- Null-terminated = last char is null char ‘\0’, not explicitly written

```
char str[] = “abc”;
```

a	b	c	\0

- String literals use “ ”. Compiler converts literals to char array.

C-Style Strings

- Char array can be larger than contained string

char str[5] = “abc”;

a	b	c	\0	

- Special chars start with ‘\’:
  - \n, \t, \b, \r: newline, tab, backspace, carriage return
  - \\, \’, \”: backslash, apostrophe, quotation mark

String functionalities

- #include <string.h>

- char pointer arguments: char str1[14]
  - char* strcpy(char* dest, const char* source);

    strcpy(str1, "hakuna ");
  - char* strcat(char* dest, const char* source);

    strcat(str1, "matata"); //str1 now has "hakuna matata"

- More in documentation…

Today…

- Computer Memory
- Pointers/Addresses
- Arrays and Strings
- **Memory Allocation**

Dynamic Allocation

- #include <stdlib.h>
- **sizeof** (a C language keyword) returns number of bytes of a data type.
- **malloc/realloc** finds a specified amount of free memory and returns a void pointer to it.

  -
    ```
    char * str = (char *) malloc( 3 * sizeof(char) );
    strcpy(str, “hi”);
    ```

  -
    ```
    str = (char *) realloc( str , 6 * sizeof(char) );
    strcpy(str, “hello”);
    ```

Dynamic Deallocation

- #include <stdlib.h>

- **free** declares the memory pointed to by a pointer variable as free for future use:

```
char * str = (char *) malloc( 3 * sizeof(char) );
strcpy(str, “hi”);
… use str …
free(str);
```

Dynamically Allocated Arrays

- Allows you to avoid declaring array size at declaration.

- Use malloc to allocate memory for array when needed:

```
int *dynamic_array;
dynamic_array = malloc( sizeof( int ) * 10 );
dynamic_array[0]=1; // now points to an array
```

Summary

- Memory has stack and heap.
- Pointers and addresses access memory.
- Arrays are really chunks of memory. Strings are null-terminated char arrays.
- C allows user memory allocation. Use malloc, realloc and free.