Richard's C Programming Course

Lecture 2: Memory, arrays and pointers

Tips welcome in XRP:
richard43NZXStHcjJi2UB8LGDQGFLKNs

Variable scope

- Variables defined inside functions can't be used by other functions

```
int main() {
  int x = 2;
  /* can only use x here */
}
```

- Variables defined outside functions can be used by any function

```
int x = 2;
/* x can be used in any function */
int main() {
}
```

Arrays

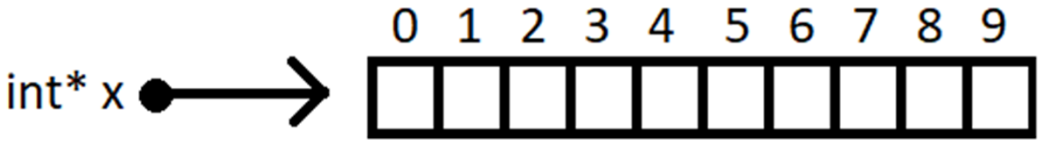
- Recall that arrays are a sequence of variables next to each other in memory.
- We can define an array of 10 integers like this:

```
int x[10];
```

- What is the variable type of `x` in this case?
  1. Is it an `int`?
  2. Is it an array?
  3. Something else?

Arrays continued

- It's actually an integer pointer. `int*`
- Pointers are variables that point to memory locations



- We can follow a pointer using the dereference operator `*`
  - `*x is the same as x[0]`
  - `*(x+5) is the same as x[5]`

- Code Example

Returning pointers

- We might want to make a function that returns an array

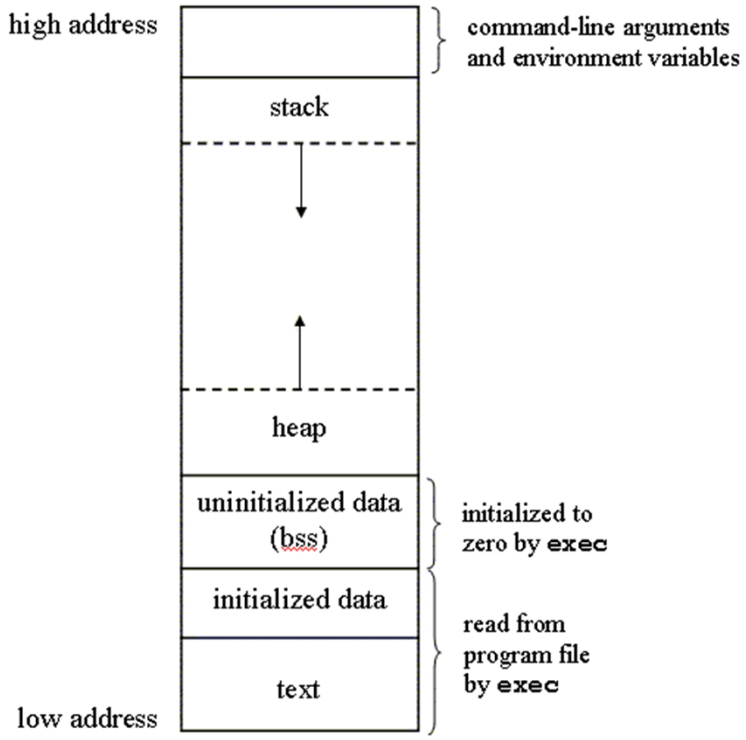
- Will this work?

```
long* myfunc() {
    long x[5];
    return x;
}
```

- No!

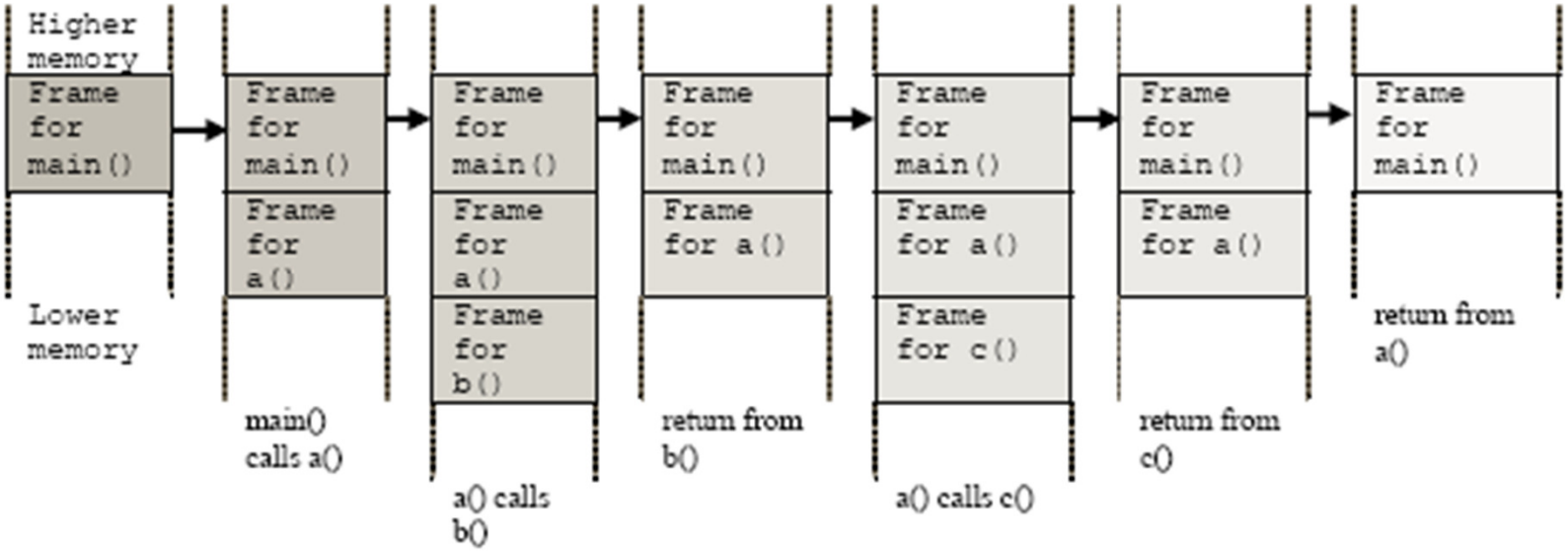
More about memory

- Stack and heap are both allocated in memory but grow from different ends



Stack frames (function calls)

- You can pass a stack allocated array to another function as a pointer
- But you cannot return a stack allocated array from a function because the stack frame is destroyed on return



Returning pointers cont.

- If we want to return a pointer it must not point to invalid memory such as a deleted stack frame.
- When a function returns it deletes all its local variables, including arrays.
- There are two modes of memory allocation in C:
  - Stack
  - Heap
- Allocation on the heap allows a function to share variables and arrays even after that function returns.

Passing pointers correctly

- Will this work?

```
void myfunc2(long* y) {
    y[2] = 0;
    return;
}
int main() {
    long x[5];
    myfunc2(x);
    printf("%d", x[2]);
    return 0;
}
```

Allocation on the heap

- Use `malloc(int)` to allocate memory on the heap
- `malloc` returns a `void*` pointer (no type information, just a block of memory)
- Up to you to set a type for the array and use it
- Always `free(void*)` after you are finished with the memory or you will run out!
- Code Example

More on Strings

- C has a lot of string manipulation functions, e.g.
  - `strcpy(char* dest, char* src)`
  - `strcat(char* dest, char* src)`
  - `strcmp(char* str1, char* str2)`
  - `strlen(char* s)`
- You can also work with strings using pointer arithmetic

- Code Example

Working with files

- We'll discuss files at length when we get to processes
- FILE* is a file stream pointer, just think of it as a handle to a file
- `fopen(char* filename, char* mode)`
- `fgets(char* buffer, int maxread, FILE* file)`
- `fprintf(FILE* file, char* formatstr, …)`
- `fclose(FILE* file)`

- Code Example