C++ Programming

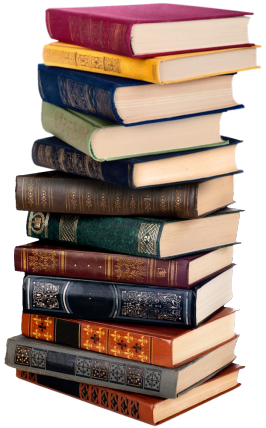
Stack and Heap

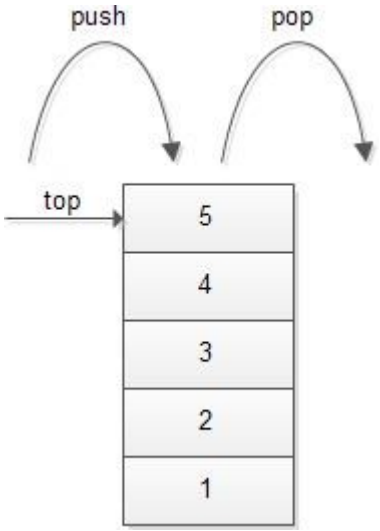
Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

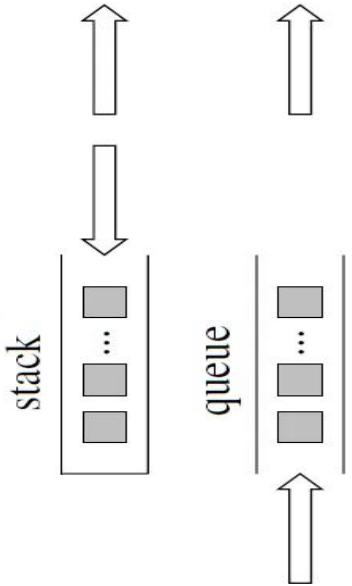
Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Recall Stack

- Imagine a box and you put book by book



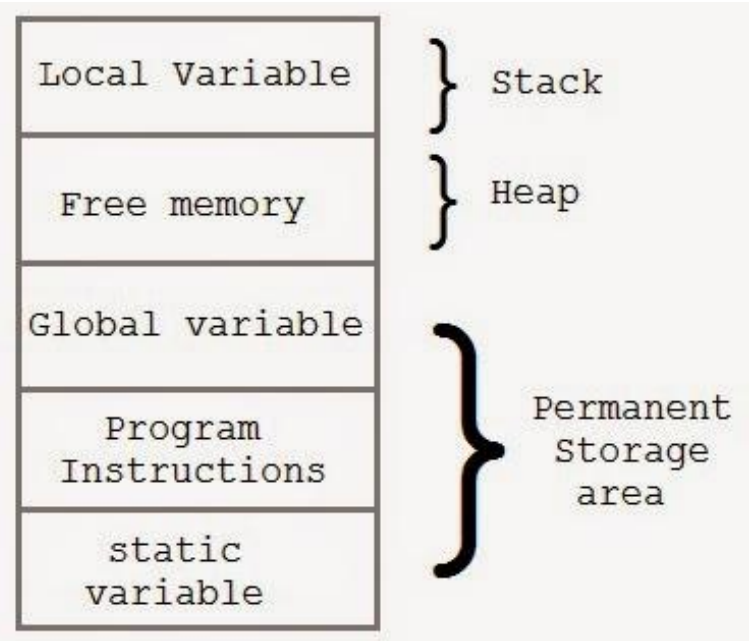




Img src src

The memory

- **Stack**: a special region of your computer's memory that stores **temporary local** variables created by each **function**
  - Automatically managed / faster
  - Local variable scope
  - Has size **limits**
  - Consecutive blocks in memory
- **Heap**: Memory managed by you (int/delete/malloc()/free())
  - No limit (up to system RAM)
  - Variables can be accessed globally
  - Maybe lead to memory fragmentation

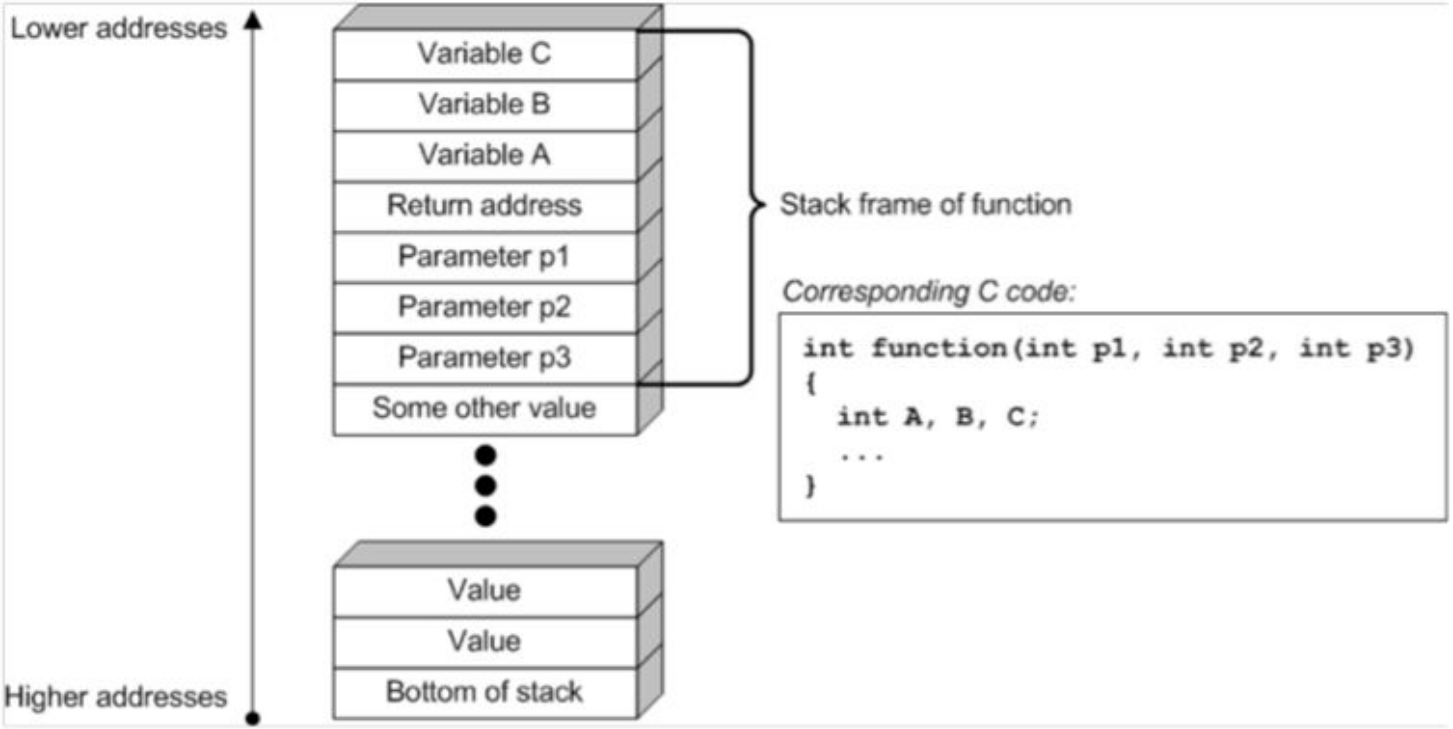


Img src

The run-time stack: memory region

- Aka: The call stack segment
- The run-time stack is stack of **function calls**
  - When a function is call, it enters stack.
  - Once finished, it is removed from the stack
- For every **active** function, following details are in the stack
  - the return value (if any)
  - the function parameter(s) (if any)
  - the return address, and
  - the function's local variable(s) (if any)
- Why we shouldn’t return a reference to a local variable from a function?
  - As all these local variables will be destroyed (removed from the stack)

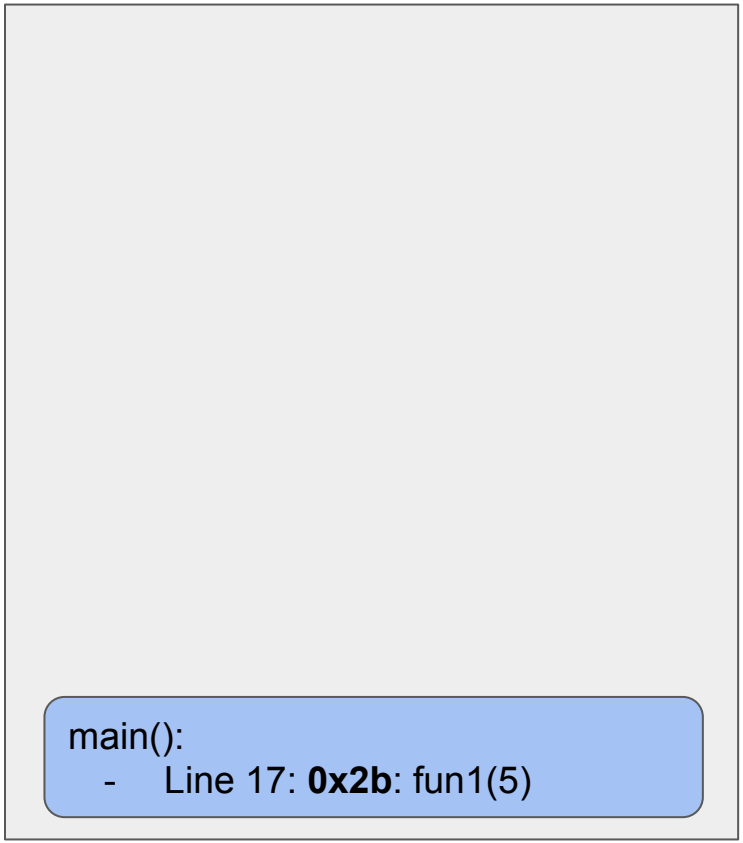
The run-time stack



Img src

The run-time stack

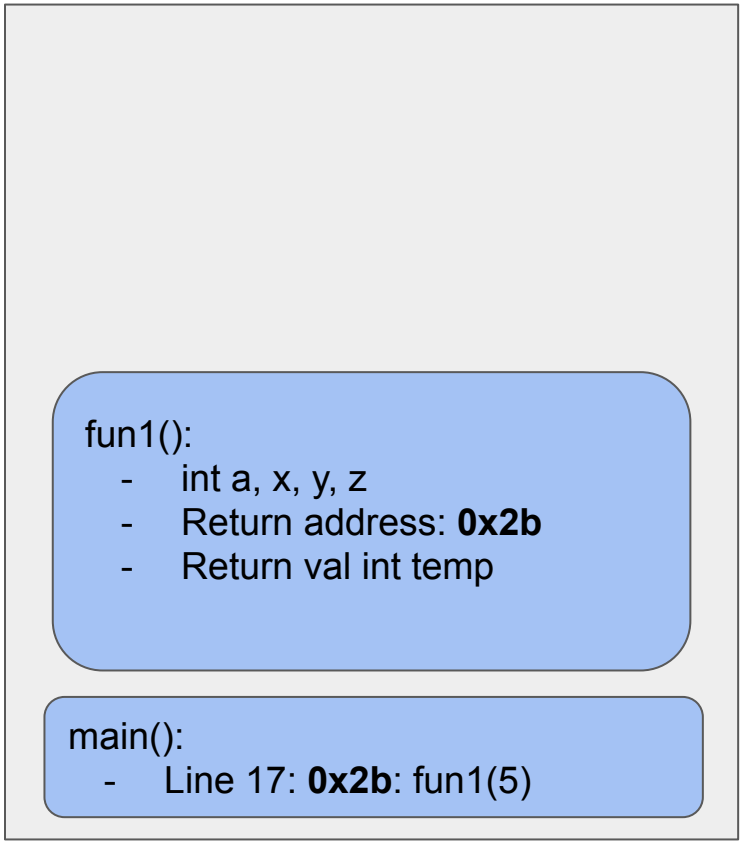
```
 4  int fun2(int m, int n) {
 5      int r = n + 2 * m;
 6      return r;
 7  }
 8  int fun1(int a) {
 9      int x = 2 * a;
10      int y = x + 1;
11
12      int z = fun2(x, y);
13
14      return z;
15  }
16  int main() {
17      fun1(5);
```



Run-time Stack

The run-time stack

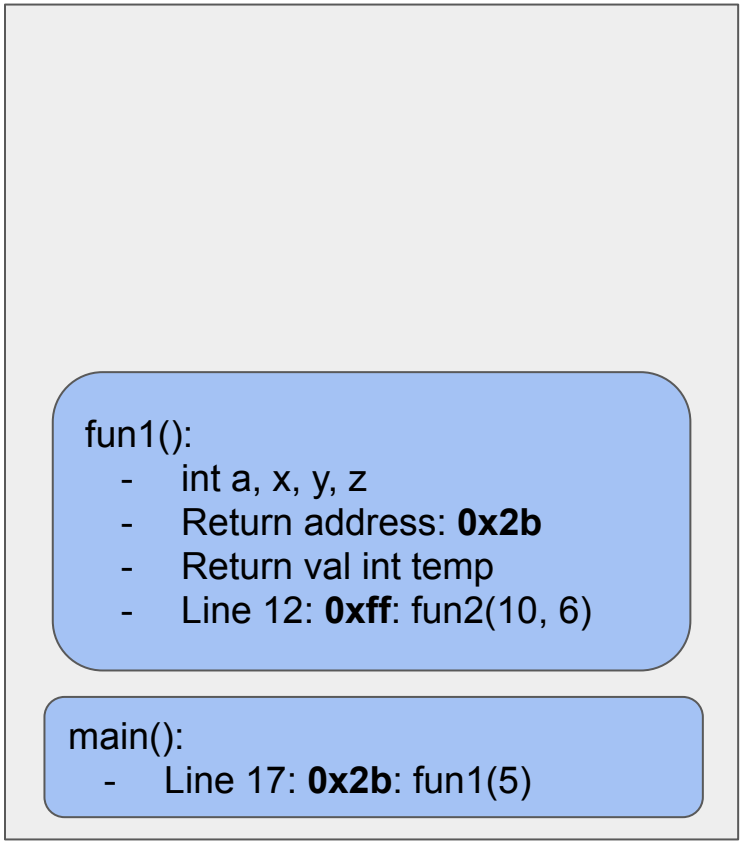
```
 4  int fun2(int m, int n) {
 5      int r = n + 2 * m;
 6      return r;
 7  }
 8  int fun1(int a) {
 9      int x = 2 * a;
10      int y = x + 1;
11
12      int z = fun2(x, y);
13
14      return z;
15  }
16  int main() {
17      fun1(5);
```



Run-time Stack

The run-time stack

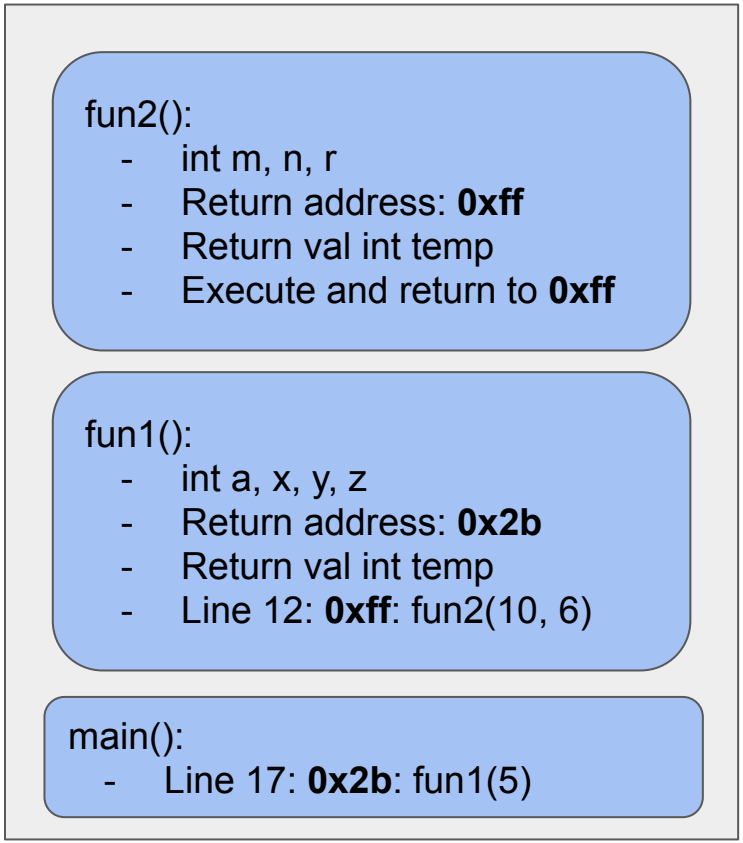
```
 4  int fun2(int m, int n) {
 5      int r = n + 2 * m;
 6      return r;
 7  }
 8  int fun1(int a) {
 9      int x = 2 * a;
10      int y = x + 1;
11
12      int z = fun2(x, y);
13
14      return z;
15  }
16  int main() {
17      fun1(5);
```



Run-time Stack

The run-time stack

```
 4  int fun2(int m, int n) {
 5      int r = n + 2 * m;
 6      return r;
 7  }
 8  int fun1(int a) {
 9      int x = 2 * a;
10      int y = x + 1;
11
12      int z = fun2(x, y);
13
14      return z;
15  }
16  int main() {
17      fun1(5);
```

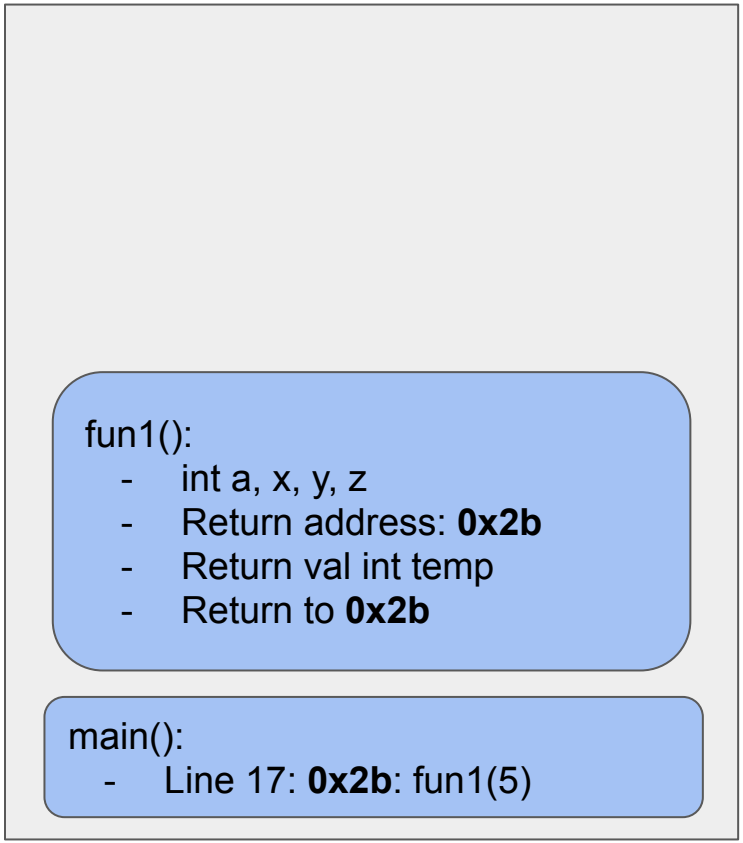


Run-time Stack

The run-time stack

```
 4  int fun2(int m, int n) {
 5      int r = n + 2 * m;
 6      return r;
 7  }
 8  int fun1(int a) {
 9      int x = 2 * a;
10      int y = x + 1;
11
12      int z = fun2(x, y);
13
14      return z;
15  }
16  int main() {
17      fun1(5);
```

Fun2 and its local vars are destroyed NOW

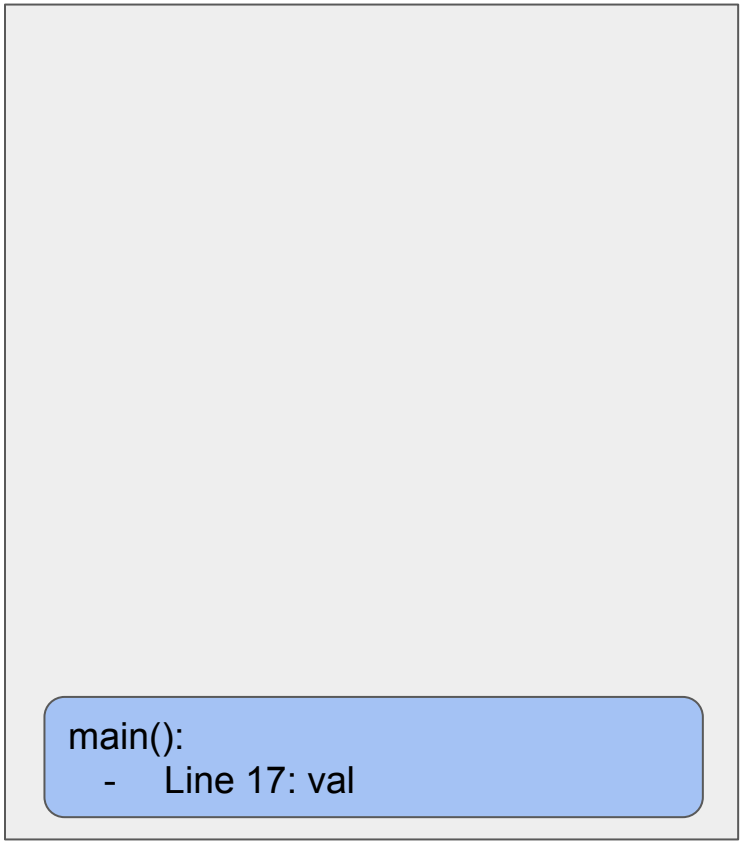


Run-time Stack

The run-time stack

```
 4  int fun2(int m, int n) {
 5      int r = n + 2 * m;
 6      return r;
 7  }
 8  int fun1(int a) {
 9      int x = 2 * a;
10      int y = x + 1;
11
12      int z = fun2(x, y);
13
14      return z;
15  }
16  int main() {
17      fun1(5);
```

Fun1 and its local vars are destroyed NOW

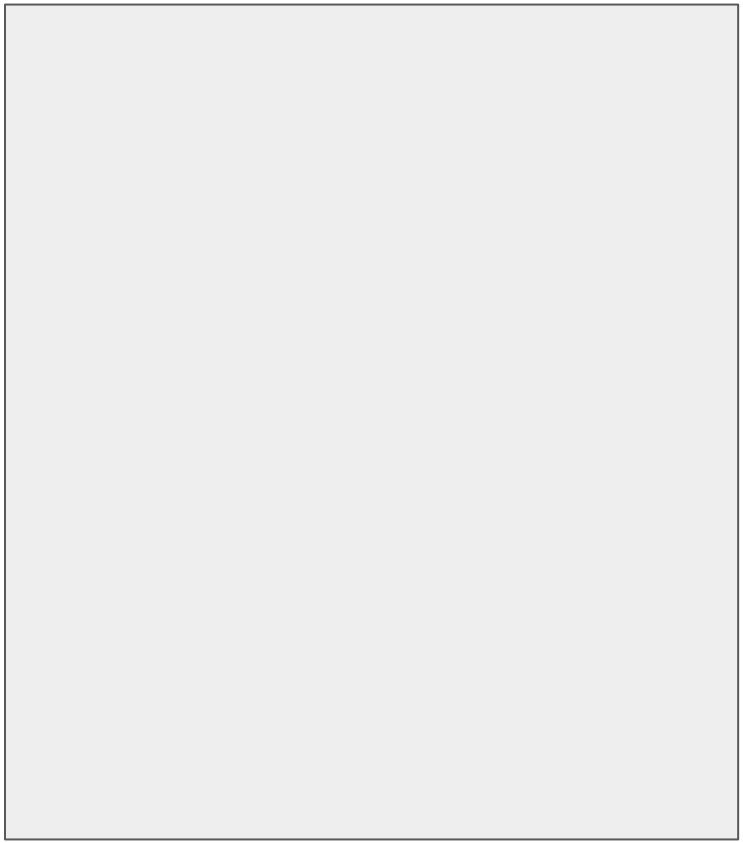


Run-time Stack

The run-time stack

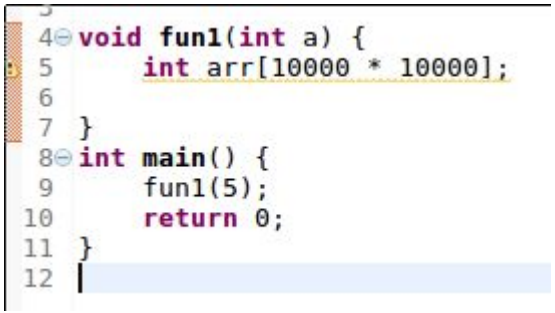
```
 4  int fun2(int m, int n) {
 5      int r = n + 2 * m;
 6      return r;
 7  }
 8  int fun1(int a) {
 9      int x = 2 * a;
10      int y = x + 1;
11
12      int z = fun2(x, y);
13
14      return z;
15  }
16  int main() {
17      fun1(5);
```

main and its local vars
are destroyed NOW



Run-time Stack

What is the potential problem?



```
4  void fun1(int a) {
5      int arr[10000 * 10000];
6
7  }
8  int main() {
9      fun1(5);
10     return 0;
11 }
12
```

- Stack overflow Problem
- We are creating a very huge local array in the stack.
  - Stack is small in size. Its memory will be consumed!
  - You should allocate in the heap
- On Windows, the default stack size is 1MB. On some unix machines, it can be as large as 8MB
  - You might be able to configure from compiler settings

What is the potential problem?

```
19
20  int frecursive1() {
21      return frecursive1();
22  }
23
24  int frecursive2(int n) {
25      int arr[100] {1, 6, 7, 8};
26      if(n < 0)
27          return 1;
28      return 1 + frecursive2(n-1);
29  }
30
31  int main() {
32      frecursive2(1000000);
33
```

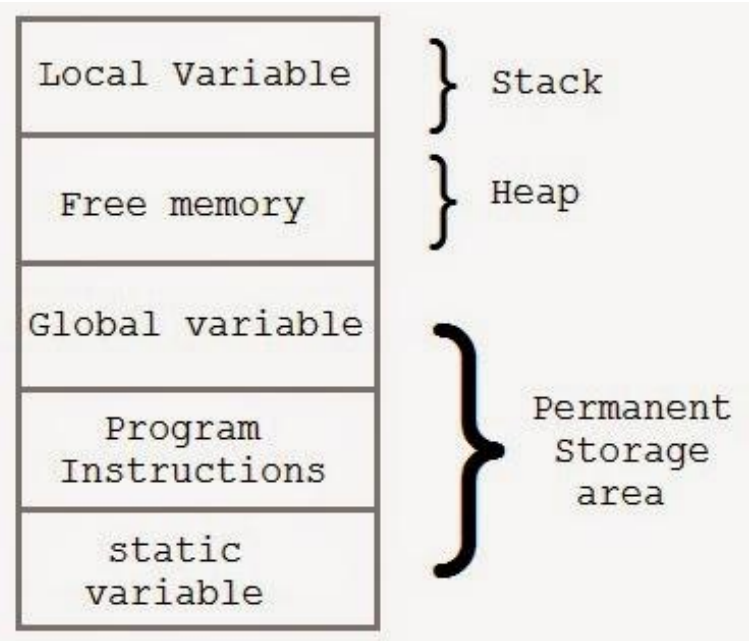
- Again Stack overflow Problem
- The first is infinite anyway
- The second, adds too many functions, with their details, to the limited stack!
- Avoid recursion in practice :)

Heap: Dynamic allocation

```
15 int* fun(int n) {
16     int* ptr1 = new int [n] {1, 3, 5, 7, 9};
17     int* ptr2 = new int [7] {1, 3, 5, 7, 9, 10, 11};
18
19     // ptr1 and ptr2: local vars in the stack
20         // Consecutive in memory
21
22     // allocated data where ptr1/ptr2 points in the HEAP
23         // ptr1 and ptr2 may not have sequential addresses
24
25     delete[] ptr2;
26     // the memory is "returned" to the heap to REUSE
27
28     return ptr1;    // ok to return. not on stack. no destroying
29 }
30 int main() {
31     int* p = fun(10);
32     delete[] p; // manage by yourself
33
```

The memory: one more time

- **Stack**: a special region of your computer's memory that stores **temporary local** variables created by each **function**
  - Automatically managed / faster
  - Local variable scope
  - Has size **limits**
  - Consecutive blocks in memory
- **Heap**: Memory managed by you (int/delete/malloc()/free())
  - No limit (up to system RAM)
  - Variables can be accessed globally
  - Maybe lead to memory fragmentation



Img src

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."