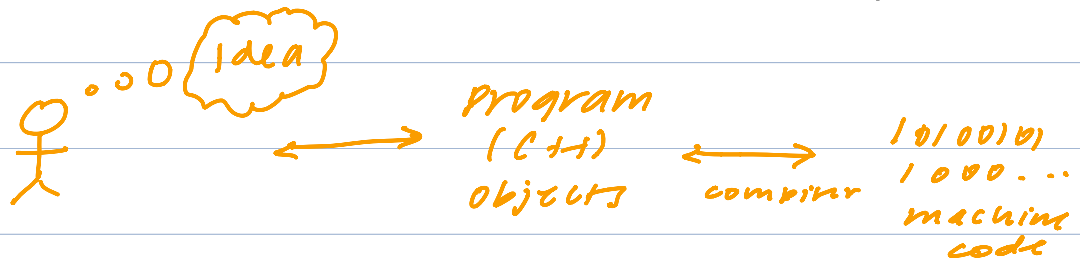
CS61: Lecture 3

September 10, 2019

Storage lifetime and data layout, Datarep 3



Storage Lifetime

```
void func() {
    { int i = 0; }
}
```

(12 hex digits)

← lifetime is only in those two brackets, automatic lifetime

```
void func() {
    int* i = new int;
}
```

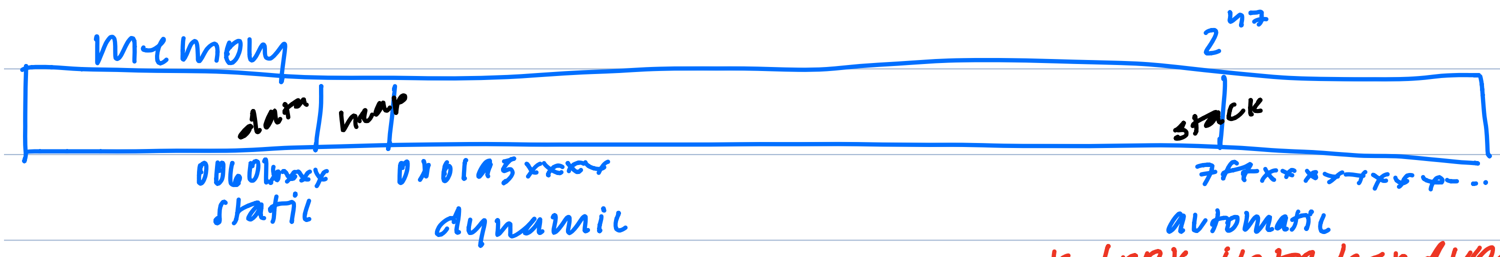
(8 hex digits)

← dynamic lifetime, ends when deleted (C++), freed (C), or program ends (memory leak)

```
#include ".h"
int i = 0;
void func() { }
```

(6 hex digits)

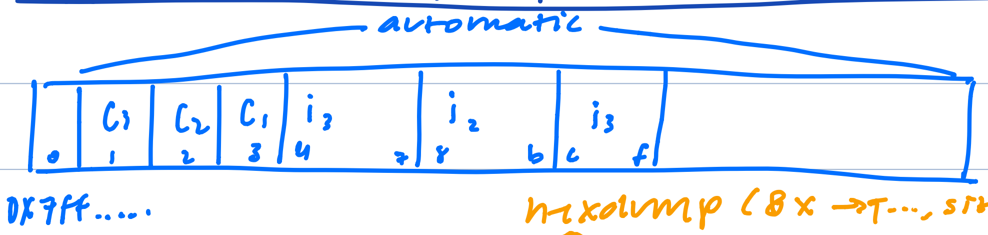
← global variable, lifetime as long as the program runs, static lifetime



* look into hexdump

lifetime	name	segment	
global	static	data	general location changes w/ compiler
function/block	automatic	stack	
malloc/free new/delete	dynamic	heap	general location changes w/ compiler

What rules do compilers use to choose locations?

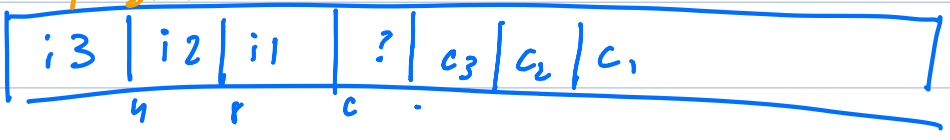


int i1;
int i2;
int i3;
char c1;
char c2;
char c3;

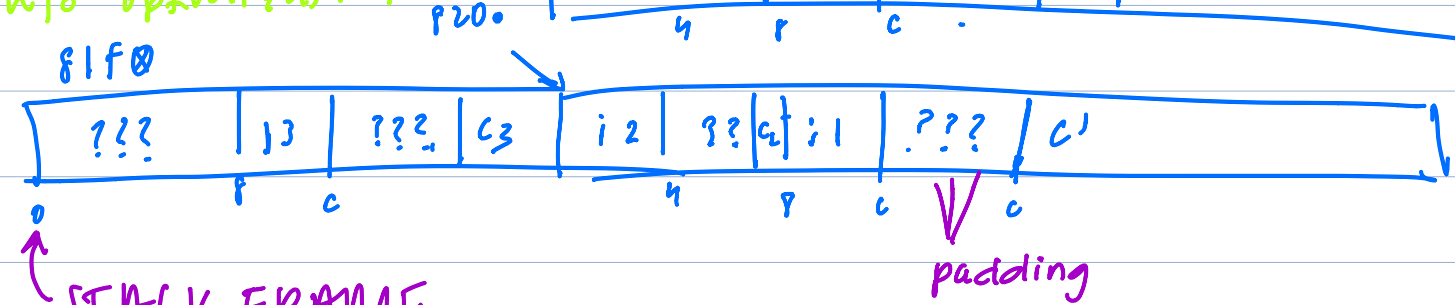
→ initialization order

hexdump (&x →T..., sizeof(T))
← program

w/ optimization



w/o optimization



STACK FRAME
(automatic storage for 1 function call)

compiler stores integers in mults. of 4

Sizes and Alignment

- same for all objects of same type
- size sizeof(T) = # of bytes required to hold value

char = 1 , int = 4

c++ requirement

not a requirement

- alignment alignof(T) evenly divides object's address
  - makes hardware better (accesses to medium-sized objects faster)

Alignment Rules

	size	alignment
char	1	1
int	4	4
long	8	8
pointers (T*)	8	8*

sizeof (T) is always a multiple of alignof (T)

* on x86 64

	size	alignment
• Array:		
T array [N]	sizeof(T) · N	alignof(T)
• Struct:		
struct {T1, T2, ...}	$\geq \sum_i$ sizeof(T_i)	max(alignof(T_i))

Laws of Alignment (mostly for collections)
↳ array, struct, class, union

1. First member law :
   - address of collection = address of first member
2. Array law:
   - elements are laid out sequentially by index w/ no gaps

```
T * array = ... ;
uintptr_t addr = (uintptr_t) array;
uintptr_t ei = addr + i x sizeof (T);
```

3. Struct rule:
   - components of simple structures are always laid out in order of declaration. There can be padding

padding → 24 bytes when alternating int and char
16 bytes when grouping ints and then chars

   - size of a structure is ≥ size of the sum of its components (due to padding)
   - Alignment = max alignment of an element