Lecture 8: Data Structures in Assembly

CS 105 Fall 2020

From Last Time…

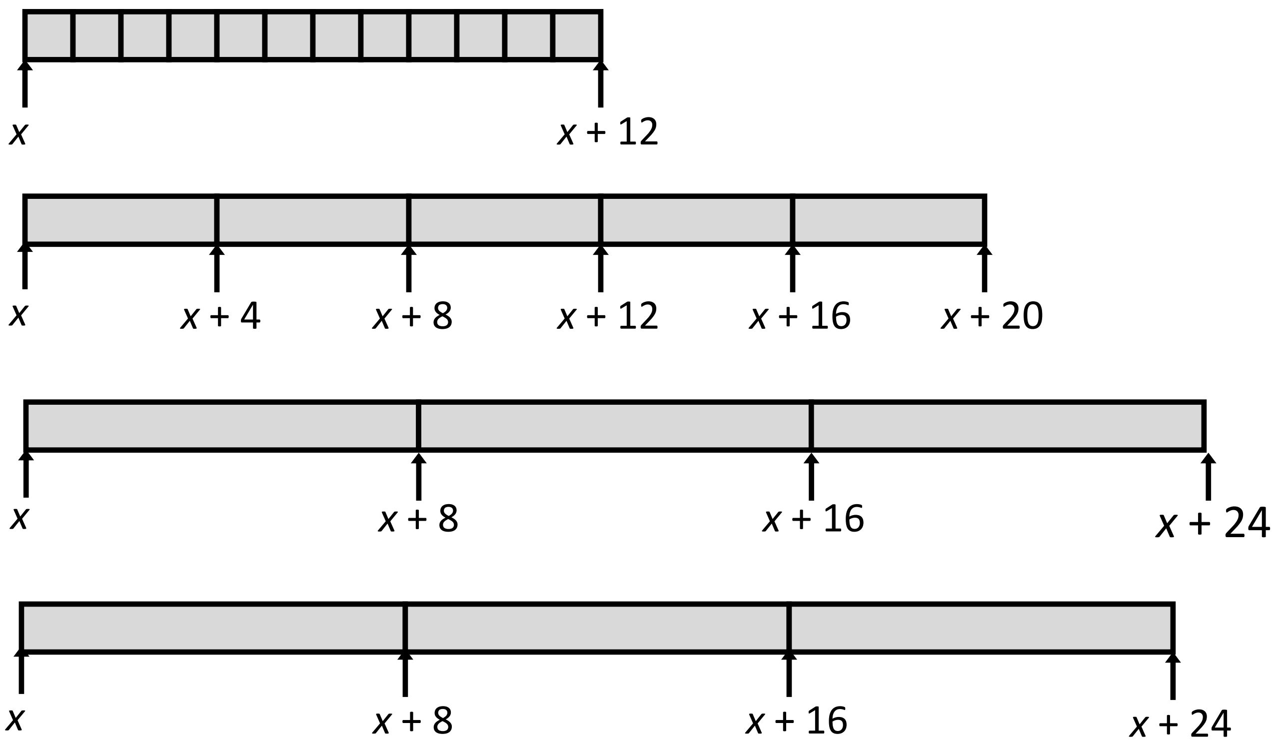
```
int proc(int *p);

int example1(int x) {
  int a[4];
  a[3] = 10;
  return proc(a);
}
```

```
example1:
  subq   $16, %rsp
  movl   $10, 12(%rsp)
  movq   %rsp, %rdi
  call   0x400546 <proc>
  addq   $16, %rsp
  ret
```

Array Allocation

- Basic Principle *T* **A[*L*]**;
  - Array of data type *T* and length *L*
  - Contiguously allocated region of *L* * **sizeof**(*T*) bytes in memory
  - Identifier **A** can be used as a pointer to array element 0: Type *T**

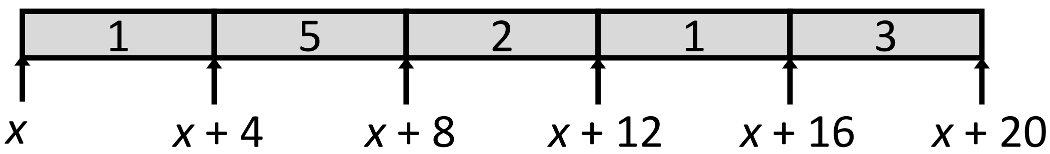
```
char string[12];
int val[5];
double a[3];
char *p[3];
```



Exercise 1: Array Access

- Basic Principle *T* **A[*L*]**;
  - Array of data type *T* and length *L*
  - Contiguously allocated region of *L* * **sizeof**(*T*) bytes in memory
  - Identifier **A** can be used as a pointer to array element 0: Type *T**

`int val[5];`

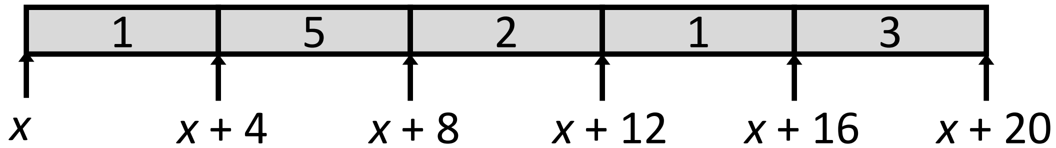


Reference	Type	Value
val[4]		
val		
val+1		
&val[2]		
val[5]		
***(val+1)**		
val + *i*		

Exercise 1: Array Access

- Basic Principle *T* **A[*L*]**;
  - Array of data type *T* and length *L*
  - Contiguously allocated region of *L* * **sizeof**(*T*) bytes in memory
  - Identifier **A** can be used as a pointer to array element 0: Type *T**

`int val[5];`



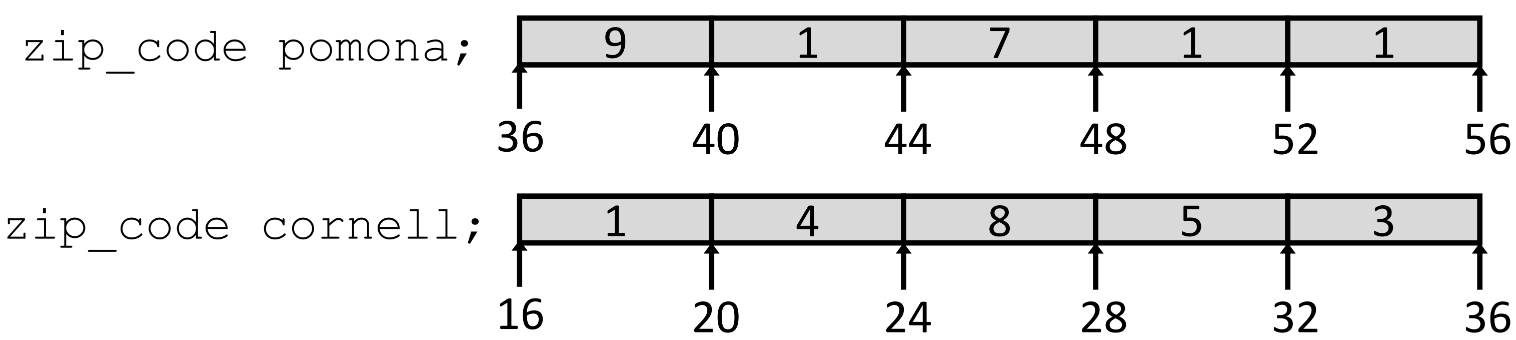
- Reference / Type / Value

Reference	Type	Value
`val[4]`	`int`	`3`
`val`	`int *`	`x`
`val+1`	`int *`	`x+4`
`&val[2]`	`int *`	`x+8`
`val[5]`	`int`	`??`
`*(val+1)`	`int`	`5`
`val + i`	`int *`	`x+4i`

Array Example

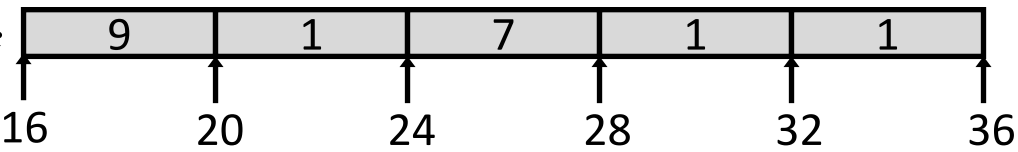
```
#define ZLEN 5
typedef int zip_code[ZLEN];

zip_code pomona  = { 9, 1, 7, 1, 1 };
zip_code cornell = { 1, 4, 8, 5, 3 };
```

- Declaration "`zip_code pomona`" equivalent to "`int pomona[5]`"



Array Accessing Example

```
zip_code pomona;
```




```
int get_digit(zip_code z, int digit){
  return z[digit];
}
```

```
# %rdi = z, %rsi = digit
movl (%rdi,%rsi,4), %eax  # z[digit]
```

- Register `%rdi` contains starting address of array
- Register `%rsi` contains array index
- Desired digit at `%rdi + 4*%rsi`
- Use memory reference `(%rdi,%rsi,4)`

Exercise 2: Array Loop

```
array_loop:
  movl    $0, %esi
  xorl    %eax, %eax
  jmp     L2
L1:
  addl    (%rdi,%rsi,4), %eax
  incq    %rsi
L2:
  cmpq    $5, %rsi
  jl      L1
  retq
```

```
int array_loop(zip_code z) {
  int sum = ______;
  int i;
  for(i = ___; i < ___; ___ )
    sum = _______;
  }
  return _______;
}
```

Variable	Register
z	
sum	
i	

Exercise 2: Array Loop

```
array_loop:
  movl     $0, %esi
  xorl     %eax, %eax
  jmp     L2
L1:
  addl     (%rdi,%rsi,4), %eax
  incq     %rsi
L2:
  cmpq     $5, %rsi
  jl     L1
  retq
```

Variable	Register
z	%rdi
sum	%rax
i	%rsi

```
int array_loop(zip_code z) {
  int sum = __0___;
  int i;
  for(i = _0_; i < _5_; i++ )
    sum = sum+z[i];
  }
  return __sum___;
}
```

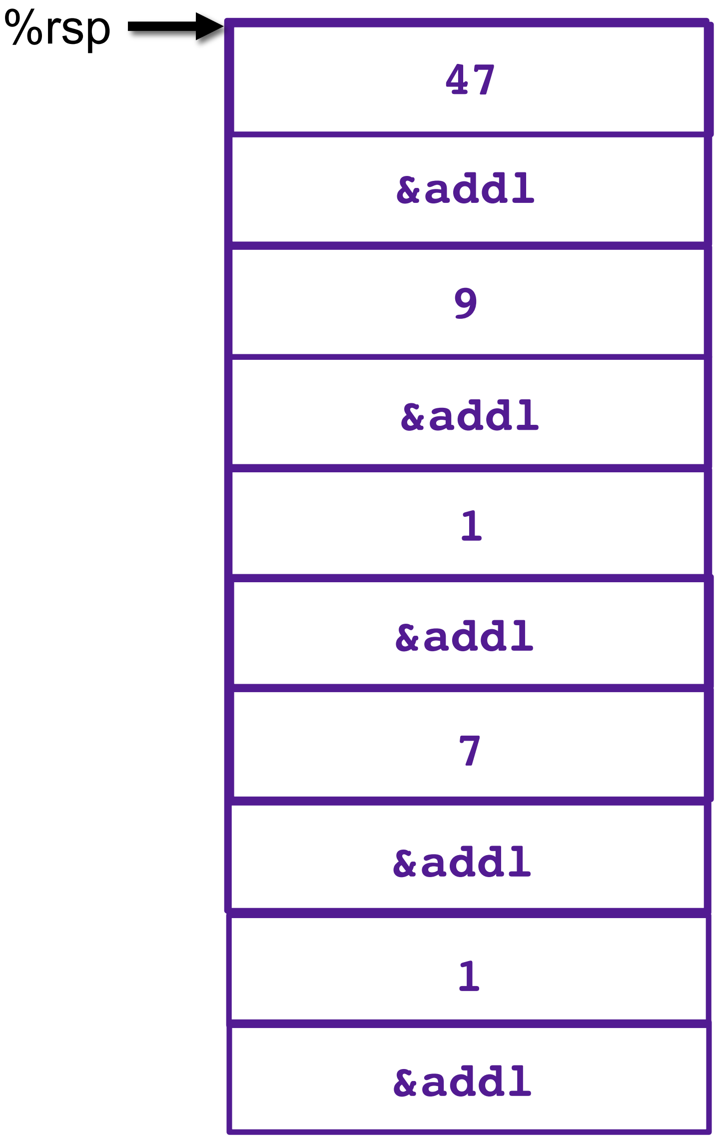
```
int array_loop(zip_code z){
  int sum = __0___;
  int i;
  for(i = _0_; i < 5__; i++ )
    sum = sum+*(z+i);
  }
  return __sum___;
}
```

Array Recursion

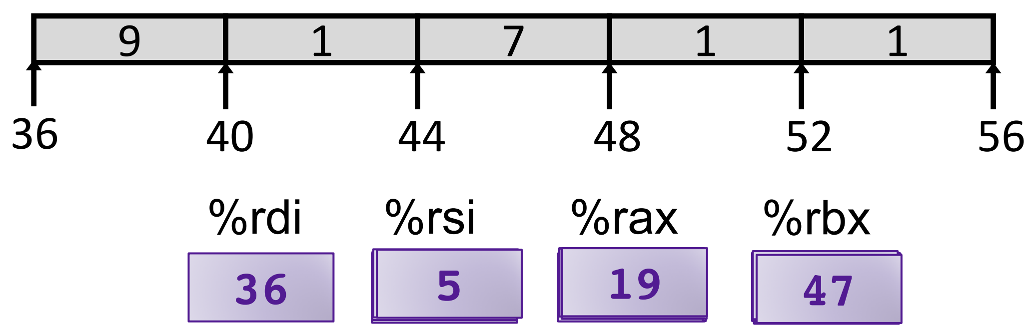
```
array_loop:
  movl    $0, %esi
  xorl    %eax, %eax
  jmp     L2
L1:
  addl    (%rdi,%rsi,4), %eax
  incq    %rsi
L2:
  cmpq    $5, %rsi
  jl      L1
  retq
```

```
array_r:
  xorl    %eax, %eax
  cmpq    $5, %rsi
  jge      L2
  pushq   %rbx
  movl    (%rdi,%rsi,4), %ebx
  incq    %rsi
  callq   array_r
  addl    %ebx, %eax
  popq    %rbx
L2:
  retq
```

Example: Array Recursion

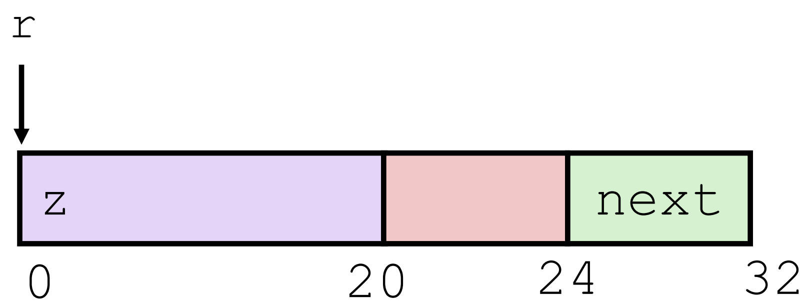



```
array_r:
   xorl     %eax, %eax
   cmpq     $5, %rsi
   jge       L2
   pushq    %rbx
   movl     (%rdi,%rsi,4), %ebx
   incq     %rsi
   callq    array_r
   addl     %ebx, %eax
   popq     %rbx
L2:
   retq
```



Structure Representation

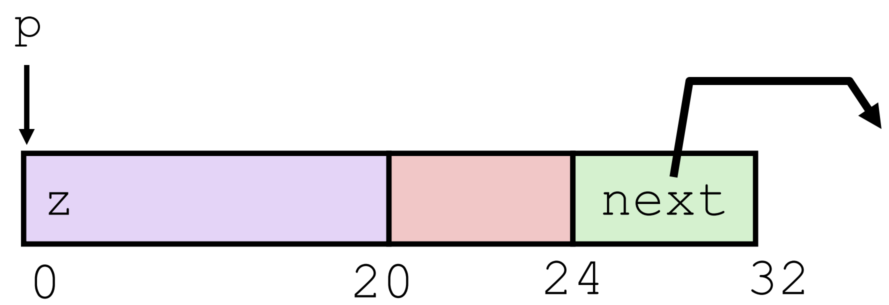
```
struct rec {
  zip_code z;
  struct rec *next;
};
```



- Structure represented as block of memory
  - **Big enough to hold all of the fields**
- Fields ordered according to declaration
  - **Even if another ordering could yield a more compact representation**
- Compiler determines overall size + positions of fields
  - **Machine-level program has no understanding of the structures in the source code**

Following Linked List

```
typedef struct rec {
  zip_code z;
  struct rec *next;
} zip_node;
```



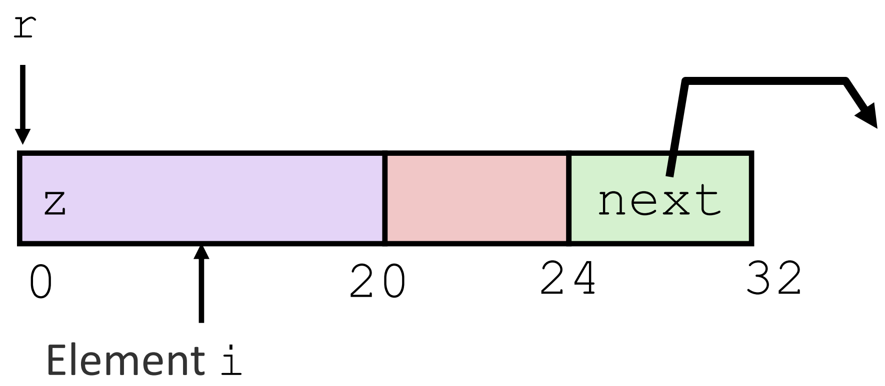

```
zip_node*get_tail_ptr(zip_node *p){
  if(p == NULL){
    return NULL;
  }

  while(p->next != NULL){
    p = p->next;
  }

  return p;
}
```

```
get_tail_ptr:
  testq    %rdi, %rdi
  jne       L1
  xorl     %eax, %eax
  retq
L1:
  movq     %rdi, %rax
  movq     24(%rax), %rdi
  testq    %rdi, %rdi
  jne      L1
  retq
```

Register	Value
%rdi	**p**

Exercise 3: Structs

```
typedef struct rec {
  zip_code z;
  struct rec *next;
} zip_node;
```



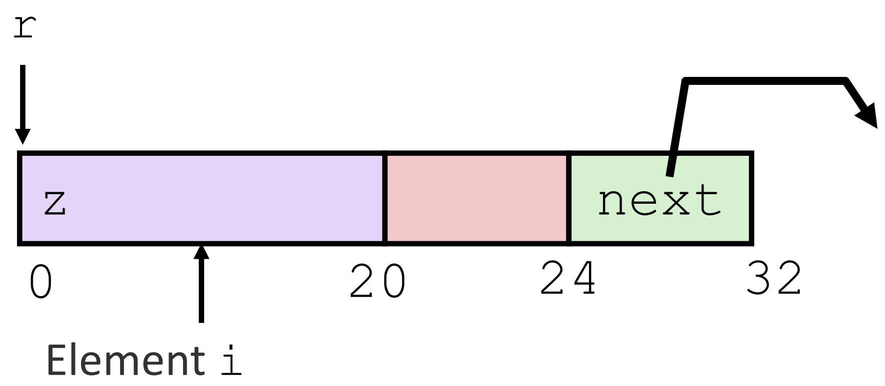

```
exercise:
  movq    %rdi, %rax
  testq   %rax, %rax
  je      L1
  movq    24(%rax), %rdi
  testq   %rdi, %rdi
  je      L2
  pushq   %rax
  callq   exercise
  addq    $8, %rsp
L2:
  retq
L1:
  xorl    %eax, %eax
  retq
```

```
zip_node *exercise(zip_node *p){



}
```

Register	Value
%rdi	**p**

Exercise 3: Structs

```
typedef struct rec {
  zip_code z;
  struct rec *next;
} zip_node;
```




```
exercise:
  movq    %rdi, %rax
  testq   %rax, %rax
  je      L1
  movq    24(%rax), %rdi
  testq   %rdi, %rdi
  je      L2
  pushq   %rax
  callq   exercise
  addq    $8, %rsp
L2:
  retq
L1:
  xorl    %eax, %eax
  retq
```

```
zip_node *exercise(zip_node *p){
    zip_node * ret = p;
    if(ret == NULL){
        return NULL;
    }
    p = p->next;
    if(p == NULL){
        return ret;
    }
    return exercise(p);
}
```

Register	Value
%rdi	p

Exercise 4: Feedback

1. Rate how well you think this recorded lecture worked
   1. Better than an in-person class
   2. About as well as an in-person class
   3. Less well than an in-person class, but you still learned something
   4. Total waste of time, you didn't learn anything
2. How much time did you spend on this video lecture (including time spent on exercises)?
3. Do you have any questions that you would like me to address in this week's problem session?
4. Do you have any other comments or feedback?