ECE 220

Lecture x0013
Linked data structures & C to LC3

Lesson objectives

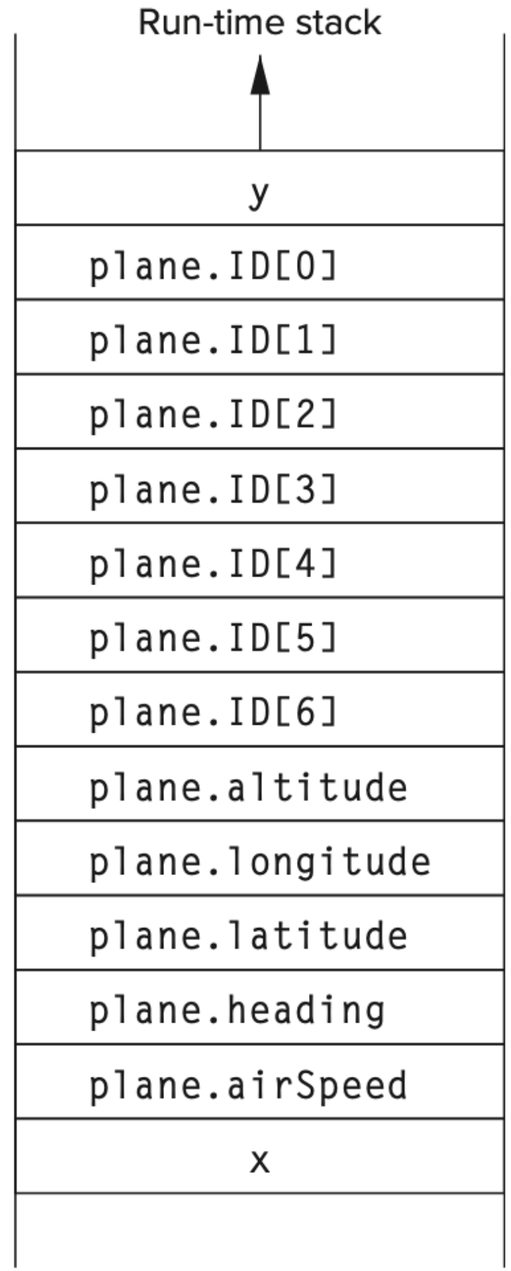
- Understand how structs are laid out in memory, specifally LC-3.
- Be able to translate C code using structs and pointers to equivalent LC-3 code in conjunction with LC-3 calling convention and Run-Time stack protocol.
- Be able to understand and reason about common operations involving linked lists and extend ideas to other kinds of linked lists (doubly linked list, circularly linked list, etc.).

Structs in LC3

Memory allocation

```
struct flightType {
    char ID[7];              // Max 6 characters
    int altitude;            // in meters
    int longitude;           // in tenths of degrees
    int latitude;            // in tenths of degrees
    int heading;             // in tenths of degrees
    double airSpeed;         // in kilometers/hour
};


int x;
struct flightType plane;
int y;
```



Memory representation

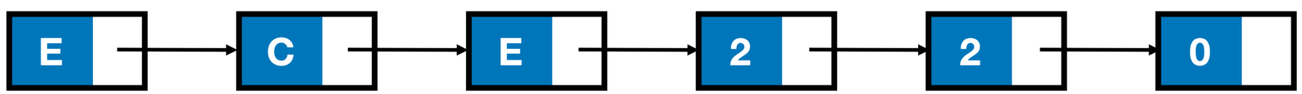
```
typedef struct Node{
    char symbol;
    struct Node *next;
}node;

int main(){
  node N3, N2, N1;
  N1.symbol = 'E'
  N2.symbol = 'C'
  N3.symbol = 'E'

  N1.next = &N2;
  N2.next = &N3;
  N3.next = NULL;
}
```

	Address	Value	Symbol
N1	x6000	E	N1.symbol
	x6001	x6002	N1.next
N2	x6002	C	N2.symbol
	x6003	x6004	N2.next
N3	x6004	E	N3.symbol
	x6005	NULL	N3.next

Printing a linked list – setup



What is at memory location x4004?

```
typedef struct Node{
    char symbol;
    struct Node *next;
}node;

void print_list(node *cursor){
  if (cursor==NULL)
    /* List empty; do nothing */
    return;
  else{
    /* Print and recurse */
    printf("%c", cursor->symbol);
    print_list(cursor->next);
  }
}
```

```
.ORIG x3000

MAIN
  LD R5, RSTACK
  LD R6, RSTACK
  LD R0, HEAD
  STR R0, R6, #0 ; push argument onto stack
  ADD R6, R6, #-1
  JSR PRINT_LIST
  ADD R6, R6, #2 ; caller teardown (pop retval & args)
  HALT

HEAD .FILL x4004
RSTACK .FILL x7000
```

```c
typedef struct Node{
    char symbol;
    struct Node *next;
}node;

void print_list(node *cursor){
  if (cursor==NULL)
    /* List empty; do nothing */
    return;
  else{
    /* Print and recurse */
    printf("%c", cursor->symbol);
    print_list(cursor->next);
  }
}
```

Printing a linked list

Head pointer →

x4004	69	x0045
x4005	16390	x4006
x4006	67	x0043
x4007	16392	x4008
x4008	69	x0045
x4009	16394	x400A
x400A	50	x0032
x400B	16396	x400C
x400C	50	x0032
x400D	16398	x400E
x400E	48	x0030

What is at memory location x7000?

Runtime stack

…	
x6FF5	
x6FF6	
x6FF7	
x6FF8	
x6FF9	
x6FFA	
x6FFB	
x6FFC	
x6FFD	
x6FFE	
x6FFF	
x7000	x4004

R5 → **R6** →

```
.ORIG x3000

MAIN
  LD R5, RSTACK
  LD R6, RSTACK
  LD R0, HEAD
  STR R0, R6, #0 ; push argument onto stack
  ADD R6, R6, #-1
  JSR PRINT_LIST
  ADD R6, R6, #2 ; caller teardown (pop retval & args)
  HALT

HEAD .FILL x4004
RSTACK .FILL x7000
```

```c
typedef struct Node{
    char symbol;
    struct Node *next;
}node;

void print_list(node *cursor){
  if (cursor==NULL)
    /* List empty; do nothing */
    return;
  else{
    /* Print and recurse */
    printf("%c", cursor->symbol);
    print_list(cursor->next);
  }
}
```

Printing LL: Buildup

R0	R1
69	x4004

```
PRINT_LIST
;;Part 1 - callee build up
  ADD R6, R6, #-1     ;make space for return value
  STR R7, R6, #0      ;push return addr to stack
  ADD R6, R6, #-1
  STR R5, R6, #0      ;push caller frame pointer
  ADD R5, R6, #-1     ;set new frame pointer

;;Part 2 - implement function logic

;if(cursor == NULL) skip to the end;
  LDR R1, R5, #4     ;load head to R1
  BRz TEAR_DOWN      ;if head null, nothing to do
;
;else printf("%c", cursor->symbol);
  LDR R0, R1, #0     ;load cursor->symbol to R0
  OUT
```

x4004	69	x0045	← Head
x4005	16390	x4006	
x4006	67	x0043	
x4007	16392	x4008	
x4008	69	x0045	
x4009	16394	x400A	
x400A	50	x0032	
x400B	16396	x400C	
x400C	50	x0032	
x400D	16398	x400E	

Runtime stack

…	
x6FF5	
x6FF6	
x6FF7	
x6FF8	
x6FF9	
x6FFA	
x6FFB	
x6FFC	
x6FFD	CFP
x6FFE	Ret Addr
x6FFF	Ret Val
x7000	x4004

R5 → Rx6 →

```c
typedef struct Node{
    char symbol;
    struct Node *next;
}node;

void print_list(node *cursor){
  if (cursor==NULL)
    /* List empty; do nothing */
    return;
  else{
    /* Print and recurse */
    printf("%c", cursor->symbol);
    print_list(cursor->next);
  }
}
```

Printing: Teardown

R0	R1
69	x4006

R5	R7
CFP	Ret Addr

```
  ;print_list(head->next);
    LDR R1, R1, #1    ;load head->next to R1
    ADD R6, R6, #-1
    STR R1, R6, #0    ;push head->next to the stack
    ADD R6, R6, #-1
    JSR PRINT_LIST
    ADD R6, R6, #2    ; caller stack tear down

;skip here if head is null
;;Part 3 - callee tear down (preparing to return)
TEAR_DOWN
    LDR R5, R6, #0    ;Restore old frame pointer
    ADD R6, R6, #1
    LDR R7, R6, #0    ;Restore return address
    ADD R6, R6, #1

RET

.END
```

x4004	69	x0045	← **Head**
x4005	16390	x4006	**R5** →
x4006	67	x0043	
x4007	16392	x4008	
x4008	69	x0045	
x4009	16394	x400A	
x400A	50	x0032	
x400B	16396	x400C	
x400C	50	x0032	

Runtime stack

…		
x6FF5	…	
x6FF6	Ret Addr	
x6FF7	Ret Val	
x6FF8	x4008	
x6FF9	CFP	
x6FFA	Ret Addr	
x6FFB	Ret Val	
x6FFC	x4006	
x6FFD	CFP	← **R6**
x6FFE	Ret Addr	
x6FFF	Ret Val	
x7000	x4004	

Linked Lists Practice

More linked lists

- Exercise(s)
  - Given two already sorted linked lists, merge them.
    - Result should still be sorted!
  - Reverse a singly linked list.
  - Implement a doubly linked list.

Sorted merging

- Given two lists of commanders-in-chiefs, merge the list while maintaining their sorted property
  - `(Reagan, 1911)->(Nixon, 1913)->(Trump, 1946)->(Bush, 1946)->NULL`
  - `(Carter, 1924)->(Biden, 1942)->(Clinton, 1946)->(Obama, 1961)->NULL`
- What should the merged list look like?

```
typedef struct person{
    char *name;
    unsigned int byear;
    struct person *next;
}node;
```

Reverse a singly-linked list

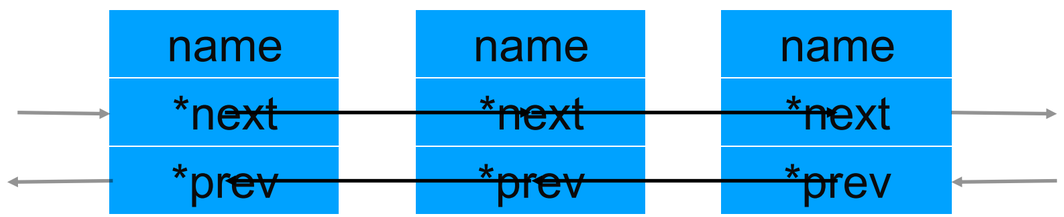
- Given the list, generate a ***new*** (not just print) linked list where the nodes are linked in ***reverse*** order.

- (Reagan, 1911)->(Nixon, 1913)->(Carter, 1924)->(Biden, 1942)->(Trump, 1946)->(Bush, 1946)->(Clinton, 1946)->(Obama, 1961)->NULL

(Obama, 1961)->(Clinton, 1946)->(Bush, 1946)->(Trump, 1946)->(Biden, 1942)->(Carter, 1924)->(Nixon, 1913)->(Reagan, 1911)->NULL

Doubly linked lists

- How many pointers to maintain?
  - Head pointer? Tail pointer?
- How should insertion/deletion work?
  - At head, middle, tail.
- We will just do an autosorted one using strcmp



Left as exercise!

```
typedef struct person{
    char *name;
    struct person *next;
    struct person *prev;
}node;
```

More practice? Okey doke!

- Practicing on linked lists:
  - Stanford's Linked List Question Bank
  - Also good practice for technical interviews!