CSE216 Programming Abstractions

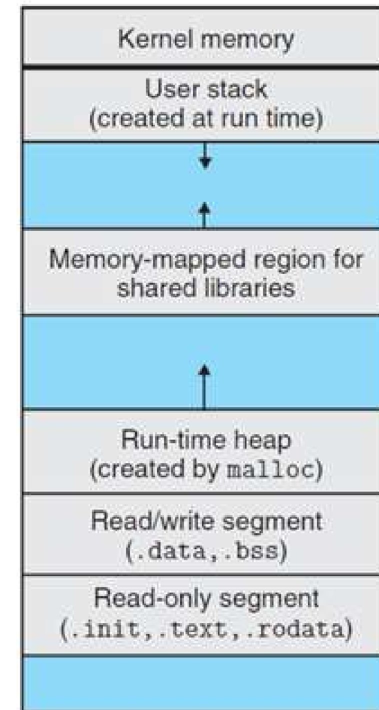
Dynamic Memory Allocation

YoungMin Kwon

SUNY Korea
The State University of New York

Dynamic Memory Allocation

- malloc
  - Allocates memory in heap
  - Allocated memory is secure until it is freed
- free
  - Frees memory allocated by malloc



SUNY Korea

Dynamic Memory Allocation

- In stdlib.h

```
//allocate size bytes of memory in heap
//returns the beginning of the allocated memory
void *malloc(size_t size);

//ptr is the address of the memory allocated by malloc
//frees the memory block starting from ptr
void free(void *ptr);

//allocate nmemb members each of size byte
//initialize the allocated memory block with 0
void *calloc(size_t nmemb, size_t size);
```

SUNY Korea

Dynamic Memory Allocation

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//incorrect
char* make_hello_1() {
    char s[] = "Hello 1"; //array is allocated in stack
    return s;
}
//correct but not dynamic
char* make_hello_2() {
    char *s = "Hello 2";  //s is the address of the string in .rodata
    return s;
}
//dynamic memory allocation
char* make_hello_3() {
    char *s = malloc(8);  //s is allocated in heap
    strcpy(s, "Hello 3"); //copy string to s
    return s;
}
```

SUNY Korea

Dynamic Memory Allocation

```
void test() {
    char *str;
    //str is the address of the array in stack (invalid now)
    str = make_hello_1();
    printf("%s\n", str);

    //str is the address of the string in .rodata
    str = make_hello_2();
    printf("%s\n", str);

    //str is the address of the string in heap
    str = make_hello_3();
    printf("%s\n", str);
    //dynamically allocated memory needs to be freed
    free(str);
}

int main() {
    test();
}
```

SUNY Korea

Dynamic Memory Allocation

- Compile and execution results

```
> gcc malloc.c
malloc.c: In function 'make_hello_1':
malloc.c:8:12: warning: function returns address of local variable…
     return s;

> a.exe
Hello 2
Hello 3
```

It was lucky that the program didn't crash

SUNY Korea
The State University of New York

Linked List Example

- Make a linked list of students
  - Circularly doubly linked list with a sentinel head
- Each student has
  - A name of string type
  - An id of integer type
- Take arbitrary number of inputs from a user until the user stops

SUNY Korea

```c
//
// studentlist.c
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct node {
    char *name;
    int id;

    //link to the next and the previous node
    struct node *next;
    struct node *prev;
};
```

```c
struct node *make_node(char *name, int id) {
    //allocate memory of sizeof(struct node) bytes for n
    struct node *n = malloc(sizeof(struct node));

    //allocate memory of strlen(name)+1 bytes for n->name
    n->name = malloc(strlen(name) + 1);
    //strcpy to n->name from name
    strcpy(n->name, name);

    //initialize other fields
    n->id = id;
    n->next = n->prev = NULL;
    return n;
}

void destroy_node(struct node *n) {
    printf("destroying node: %d, %s\n", n->id, n->name);
    //deallocate n->name
    free(n->name);
    //deallocate n
    free(n);
}
```

```c
void add_node_between(struct node *pred,
                      struct node *succ,
                      struct node *n) {
    //insert n in between pred and succ
    n->prev = pred;
    n->next = succ;
    pred->next = n;
    succ->prev = n;
}

void remove_node(struct node *n) {
    //remove n from the linked list
    struct node *pred = n->prev;
    struct node *succ = n->next;
    pred->next = succ;
    succ->prev = pred;
}
```

SUNY Korea
The State University of New York

```c
struct node *make_list() {
    char name[256];
    int id;
    //make a head node (sentinel)
    struct node *head = make_node("head", -1);
    //circularly doubly linked list
    head->next = head->prev = head;

    while(1) {
        struct node *n;
        printf("enter name or quit to stop: ");
        scanf("%255s", name);
        if(strcmp(name, "quit") == 0)
            break;
        printf("enter id: ");
        scanf("%d", &id);
        //make a node with name and id
        n = make_node(name, id);
        //add n to the last position of the list
        add_node_between(head->prev, head, n);
    }
    //return head
    return head;
}
```

SUNY Korea
The State University of New York

```c
void print_list(struct node *head) {
    struct node *n;
    //from head->next until n reaches head, print n->id and n->name
    for(n = head->next; n != head; n = n->next)
        printf("%3d: %s\n", n->id, n->name);
}

void destroy_list(struct node *head) {
    while(head->next != head) {
        struct node *n = head->next;
        //remove n from the list
        remove_node(n);
        //destroy n
        destroy_node(n);
    }
    //destroy head
    destroy_node(head);
}

int main() {
    struct node *head;
    head = make_list();
    print_list(head);
    destroy_list(head);
}
```

SUNY Korea

Execution result

```
> a.exe
enter name or quit to stop: abc
enter id: 1
enter name or quit to stop: bcd
enter id: 2
enter name or quit to stop: cde
enter id: 3
enter name or quit to stop: quit
  1: abc
  2: bcd
  3: cde
destroying node: 1, abc
destroying node: 2, bcd
destroying node: 3, cde
destroying node: -1, head
```

SUNY Korea
The State University of New York

Modular Linked List in C

- Problems with the previous example
  - Linked list is not modular
  - List algorithms are mixed with other parts of the code
  - List algorithms cannot be reused the for other structures
- Approach
  - Embed a linked list structure in a container

SUNY Korea

Pointer Arithmetic

- A pointer variable is associated with a type
- `+`, `-` operators on pointers add or subtract the size of the associated type
- Examples
  - `p++`, `p--`, `p += 2`, …
  - `p – q` is the number of elements between `p` and `q`
  - `p + q`, `p * 2`, `p / 2`,… are invalid operations

SUNY Korea

Pointer Arithmetic

```c
#include <stdio.h>
void getbuf(int** p, int n) {
    static int next = 0;
    static int buf[100] = {
        0, 1, 2, 3, 4, 5, 6, };

    // TODO: boundary check

    *p = buf + next;
    next += n;
}
```

```c
int main() {
    int *a, *b;

    getbuf(&a, 4);
    getbuf(&b, 4);

    printf("a + 1: %d, %d, %d\n",
           a, a + 1, (int)a + 1);
    printf("b - a: %d, %d\n",
           b - a, (int)b - (int)a);
}
```

```
$ a.out
a + 1: 4202560, 4202564, 4202561
b - a: 4, 16
```

SUNY Korea

Offset of

- offsetof: offset of a field in a structure

```
typedef struct pair {
    char i;
    int  j;
} pair_t;

void offsetof_test() {
    pair_t pr;
    printf("&pr: %p, &pr.i: %p, &pr.j: %p\n", &pr, &pr.i, &pr.j);

    //compile error if types do not match for pointer arithmetic
    //printf("&pr.i - &pr: %d\n", &pr.i - &pr);
    //error: invalid operands to binary
    // - (have 'char *' and 'pair_t *' {aka 'struct pair *'})

    printf("&pr.i - &pr: %d\n", (char*)&pr.i - (char*)&pr);
    printf("&pr.j - &pr: %d\n", (char*)&pr.j - (char*)&pr);
}
```

```
> a.exe
&pr: 0061FEB8, &pr.i: 0061FEB8, &pr.j: 0061FEBC
&pr.i - &pr: 0
&pr.j - &pr: 4
```

Offset of

- offsetof macro:
  - How far a field of a structure is from the beginning of the structure

```
//if an st type data is at address 0,
//the address of its field m is the offset of m
#define offsetof(st, m) ( (size_t) &(((st *)0)->m) )

typedef struct pair {
    char i;
    int  j;
} pair_t;

void offsetof_test() {
    printf("offsetof(pair_t, i): %d\n", offsetof(pair_t, i));
    printf("offsetof(pair_t, j): %d\n", offsetof(pair_t, j));
}
```

```
> a.exe
offsetof(pair_t, i): 0
offsetof(pair_t, j): 4
```

SUNY Korea

Container of

- containerof: container of a field
  - Find the address of the container when we know the address of its field
  - Container adrs = field adrs - offset of the field

```
void containerof_test() {
    pair_t pr;

    int offset_i = (char*)&pr.i - (char*)&pr;
    printf("&pr: %p, &pr.i: %p, offset_i: %d\n", &pr, &pr.i, offset_i);
    printf("&pr.i - offset_i: %p\n",             (char*)&pr.i - offset_i);

    int offset_j = (char*)&pr.j - (char*)&pr;
    printf("&pr: %p, &pr.j: %p, offset_j: %d\n", &pr, &pr.j, offset_j);
    printf("&pr.j - offset_j: %p\n",             (char*)&pr.j - offset_j);
}
```

SUNY Korea

Container of

- containerof macro

```
//container address = field address - offset of the field
#define containerof(ptr, st, m) \
        ((st *) (((char*)(ptr)) - offsetof(st, m)))

void containerof_test() {
    pair_t pr;

    printf("containerof(&pr.i, pair_t, i): %p\n",
            containerof(&pr.i, pair_t, i));
    printf("containerof(&pr.j, pair_t, j): %p\n",
            containerof(&pr.j, pair_t, j));
}
```

SUNY Korea

Container of

- containerof macro

```
> a.exe
containerof test

&pr: 0061FEB0, &pr.i: 0061FEB0, offset_i: 0
&pr.i - offset_i: 0061FEB0
containerof(&pr.i, pair_t, i): 0061FEB0

&pr: 0061FEB0, &pr.j: 0061FEB4, offset_j: 4
&pr.j - offset_j: 0061FEB0
containerof(&pr.j, pair_t, j): 0061FEB0
```

Modular Linked List in C

- Modular linked list
  - Linked list structure has only prev and next fields without data
  - Linked list will be embedded in a container

```
typedef struct list {
    struct list *next;
    struct list *prev;
} list_t;

struct node {
    char *name;
    int id;

    //embedded linked list: algorithms for list_t can be
    //developed separately from its containers
    list_t lst;
};
```

Modular Linked List in C

```
//
// list.h
//
#ifndef __LIST__
#define __LIST__

//offset of m in st
#define offsetof(st, m) ((size_t) &(((st *)0)->m))

//container address when the address of m in st is ptr
#define containerof(ptr, st, m) \
        ((st *) (((char*)(ptr)) - offsetof(st, m)))

//list_t will be embedded in a container
typedef struct list {
    struct list *next;
    struct list *prev;
} list_t;
```

SUNY Korea

Modular Linked List in C

…

```
extern void    list_init_head   (list_t *head);
extern int     list_size        (list_t *head);
extern int     list_is_empty    (list_t *head);
extern void    list_add_to_first(list_t *head, list_t *node);
extern void    list_add_to_last (list_t *head, list_t *node);
extern void    list_add_after   (list_t *pos, list_t *node);
extern void    list_add_before  (list_t *pos, list_t *node);
extern list_t *list_remove      (list_t *node);
extern list_t *list_remove_first(list_t *head);
extern list_t *list_remove_last (list_t *head);
extern list_t *list_find        (list_t *head, void *data,
                                  int (*comp)(list_t*, void*));

#endif
```

SUNY Korea
The State University of New York

```c
//
// list.c
//
#include "list.h"
#include <stdio.h>

//init head
void list_init_head(list_t *head) {
    head->next = head;
    head->prev = head;
}

//size
int list_size(list_t *head) {
    int size = 0;
    list_t *pos;
    for(pos = head->next; pos != head; pos = pos->next)
        size++;
    return size;
}

//is empty
int list_is_empty(list_t *head) {
    return head->next == head;
}
```

SUNY Korea

```c
//add methods
static void list_add_between(list_t *prev, list_t *succ, list_t *node) {
    node->prev = prev;
    node->next = succ;
    prev->next = node;
    succ->prev = node;
}

void list_add_to_first(list_t *head, list_t *node) {
    list_add_after(head, node);
}
void list_add_to_last(list_t *head, list_t *node) {
    list_add_before(head, node);
}
void list_add_after(list_t *pos, list_t *node) {
    list_add_between(pos, pos->next, node);
}
void list_add_before(list_t *pos, list_t *node) {
    list_add_between(pos->prev, pos, node);
}
```

```c
//remvoe methods
list_t *list_remove(list_t *node) {
    list_t *pred = node->prev;
    list_t *succ = node->next;
    pred->next = succ;
    succ->prev = pred;
    return node;
}

list_t *list_remove_first(list_t *head) {
    return list_remove(head->next);
}
list_t *list_remove_last(list_t *head) {
    return list_remove(head->prev);
}

//find method
list_t *list_find(list_t *head, void *data, int (*comp)(list_t*, void*)) {
    list_t *pos;
    for(pos = head->next; pos != head; pos = pos->next)
        if(comp(pos, data))
            return pos;
    return NULL;
}
```

SUNY Korea
The State University of New York

Linked List Example (2nd round)

- Make a linked list of students
  - Using an embedded linked list
- Each student has
  - A string of name
  - In integer of id
- Take arbitrary number of inputs from a user until the user stops

SUNY Korea

```c
//
// studentlist2.c
//
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "list.h"

struct node {
    char *name;
    int id;

    //embedded linked list
    list_t lst;
};
```

SUNY Korea
The State University of New York

```c
struct node *make_node(char *name, int id) {
    //allocate memory of sizeof(struct node) bytes for n
    struct node *n = malloc(sizeof(struct node));
    //allocate memory of strlen(name)+1 bytes for n->name
    n->name = malloc(strlen(name) + 1);
    //strcpy to n->name from name
    strcpy(n->name, name);
    //initialize other fields
    n->id = id;

    n->lst.next = n->lst.prev = NULL;

    return n;
}

void destroy_node(struct node *n) {
    printf("destroying node: %d, %s\n", n->id, n->name);
    //deallocate n->name
    free(n->name);
    //deallocate n
    free(n);
}
```

SUNY Korea
The State University of New York

```c
void make_list(list_t *head) {
    char name[256];
    int id;

    //make a list head
    list_init_head(head);

    while(1) {
        struct node *n;
        printf("enter name or quit to stop: ");
        scanf("%255s", name);
        if(strcmp(name, "quit") == 0)
            break;
        printf("enter id: ");
        scanf("%d", &id);
        //make a node with name and id
        n = make_node(name, id);

        //add the embedded lst to the last position of the list
        list_add_to_last(head, &n->lst);
    }
}
```

SUNY Korea

```c
void print_list(list_t *head) {
    list_t *pos;
    //from head->next until pos reaches head, print n->id and n->name
    for(pos = head->next; pos != head; pos = pos->next) {
        //get the node pointer from pos using containerof
        struct node *n = containerof(pos, struct node, lst);
        printf("%3d: %s\n", n->id, n->name);
    }
}

void destroy_list(list_t *head) {
    while(! list_is_empty(head)) {
        //remove node from the linked list
        list_t *pos = list_remove_first(head);

        //get the node pointer from pos using containerof
        struct node *n = containerof(pos, struct node, lst);

        //destroy node
        destroy_node(n);
    }
}
```

SUNY Korea

```c
int main() {
    list_t head;    //head of the list
    make_list(&head);
    print_list(&head);
    destroy_list(&head);
}
```

```
> a.exe
enter name or quit to stop: abc
enter id: 1
enter name or quit to stop: bcd
enter id: 2
enter name or quit to stop: cde
enter id: 3
enter name or quit to stop: quit
  1: abc
  2: bcd
  3: cde
destroying node: 1, abc
destroying node: 2, bcd
destroying node: 3, cde
```