Lecture 08

Void 함수 및 해시테이블

리뷰

- 포인터
  - 어떤 변수의 메모리 번지를 갖고 있는 것임

```
int x; int *px = &x;
```

  - 포인터의 포인터

```
int x; int *px = &x; int **ppx = &px;
```

  - 포인터의 배열

```
char *names[] = {"Alex", "Dave", "Anne"};
```

  - 다차원 배열

```
int x[10][10];
```

리뷰

- 스택(stack)
  - 후입선출(Last In First Out) 자료구조
  - 스택에 실행 할 수 있는 연산
    - Push : 스택에 데이터를 추가함
    - Pop : 스택에서 데이터를 삭제함
    - Peek : 스택에서 데이터를 삭제하지 않고, 들여다보기만 함
  - 배열이나 연결리스트를 기반으로 구현할 수 있음

```
typedef struct {
    int data[100];
    int top;
} stack;
```

```
typedef struct Node {
    int data;
    struct Node *next;
} node;
node *stack = NULL;
```

리뷰

- 큐(queue)
  - 선입선출(First In First Out) 자료구조
  - 큐에 실행 할 수 있는 연산
    - Enqueue : 큐 끝에 데이터를 추가함
    - Dequeue : 큐 앞에서 데이터를 삭제함
  - 배열이나 연결리스트를 기반으로 구현할 수 있음

```
typedef struct {
    int data[100];
    int front;
    int rear;
} queue;
```

```
typedef struct Node {
    int data;
    struct Node *next;
} node;
typedef struct {
    node *front;
    node *rear;
} queue;
```

리뷰

- 수식 표기법
  - Infix(중위) : * + A B - C D
  - Prefix(전위) : (A + B) * (C - D)
  - Postfix(후위) : A B + C D - *

Infix – Postfix 변환: A + B * C - D

토큰	출력 큐	스택
A	A	
+	A	+
B	A B	+
*	A B	+ *
C	A B C	+ *
-	A B C * +	-
D	A B C * + D	-
(끝)	A B C * + D -	

Postfix 수식 실행: 3 4 + 5 1 - *

토큰	스택
3	3
4	3 4
+	7
5	7 5
1	7 5 1
-	7 4
*	28 (최종 결과)
(끝)	

Void 포인터

- C에서 **void** 변수를 선언할 수 없지만, **void**는 함수의 리턴형이나 함수의 매개변수로 사용될 수 있음
  - `void x;` // 에러
  - `void func(void);` // 매개변수를 받지 않고, 리턴하는 값이 없음
- **void** 포인터는 선언될 수 있으며, 어떤 형의 포인터로 변환될 수 있음

```
int x=10;
float y=5.87;
void *p;
p = &x; printf("%d\n", *(int*)p); // 10 출력
p = &y; printf("%f\n", *(float*)p); // 5.87 출력
```

Void 포인터

- **void** 포인터에 * 연산자를 바로 할 수 없으며, * 연산자를 하기 전에 다른 데이터형의 포인터로 변환해줘 야 함

```
int x=10;
void *p = &x;
printf("%d\n", *p); // 에러
printf("%d\n", *(int*)p); // 정상
```

- 원래 C에서 **void** 포인터에 연산을 할 수 없지만, GNU C에서는 **void** 포인터에 연산을 할 수 있음

```
int a[2] = {1, 2};
void *p = a;
p = p+sizeof(int); // 원래 C에서는 이러한 연산을 허용하지 않음
printf("%d\n", *(int*)p); // 2 출력
```

함수의 포인터

- 함수는 변수가 아니지만, 함수의 포인터를 선언할 수 있음

```
int (*fp)(int);
int (*fp)(void*, void*);
```

- 함수의 포인터는 다른 함수로 전달될 수 있음

```
int func(int, int, int (*fp)(int));
```

- 함수의 포인터로 이루어지는 배열을 선언할 수 있음

```
int (*farr[])(int, int) = {func1, func2, func3};
```

함수의 포인터

- 콜백(callback)
  - 다른 함수로 전달될 수 있는 실행이 가능한 코드임
  - C에서 함수의 포인터를 통해 콜백을 구현할 수 있음
- 예로, 표준 라이브러리의 `qsort()` 함수

```
void qsort(void *arr, int num, int size, int (*fp)(const void *pa, const void *pb));
```

  - 어떤 형의 배열이든 `qsort()`를 통해 정렬할 수 있음
  - `qsort()`는 배열의 요소를 비교해야 할 때마다 비교 함수를 호출함
  - 비교 함수는 매개변수의 2개를 받아 비교하여 비교 결과에 따라 (`<0, 0, >0`)와 같은 3가지 값을 리턴함

함수의 포인터

- 오름차순 비교 함수

```
int ascend(const void *pa, const void *pb) {
    return (*(int*)pa - *(int*)pb);
}
```

- 내림차순 비교 함수

```
int descend(const void *pa, const void *pb) {
    return (*(int*)pb - *(int*)pa);
}
```

- qsort()로 전달함

```
int arr[] = {10, 9, 8, 1, 2, 3, 5};
qsort(arr, sizeof(arr)/sizeof(int), sizeof(int), ascend);
qsort(arr, sizeof(arr)/sizeof(int), sizeof(int), descend);
```

함수의 포인터

- 연결리스트와 같이 사용하면 더 유용해짐

```
typedef struct node {
    int data;
    struct node *next;
} node;

void apply(node *head, void (*fp)(void*, void*), void *arg) {
    node *p = head;
    while (p!=NULL) {
        fp(p, arg);
        p = p->next;
    }
}
```

apply() 함수는 리스트의 모두 노드에 fp 함수를 적용함

함수의 포인터

- 예로, 리스트 종주(traversal)

```
void print(void *p, void *arg) {
    node *pnode = (node*)p;
    printf("%d ", pnode->data);
}

node *head = createNode(10);
head->next = createNode(20);
head->next->next = createNode(30);

apply(head, print, NULL); // 10 20 30 출력
```

함수의 포인터

- 예로, 리스트의 노드 합

```
void addnode(void *p, void *arg) {
    node *pnode = (node*)p;
    int *psum = (int*)arg;
    *psum += pnode->data;
}

node *head = createNode(10);
head->next = createNode(20);
head->next->next = createNode(30);

int sum = 0;
apply(head, addnode, &sum);
printf("%d", sum); // 60 출력
```

함수의 포인터

- 구조체 속에 함수를 정의할 수 없지만, 함수의 포인터를 정의하여 그 포인터가 외부 함수를 가리키도록 할 수 있음

```
typedef struct Rect {
    int w,  h;
    void (*set)(struct Rect*, int, int);
    int (*area)(struct Rect*);
} rect;
```

```
void set(rect *r, int w, int h) {
    r->w = w;
    r->h = h;
}

int area(rect *r) {
    return r->w*r->h;
}
```

```
rect *createRect() {
    rect *r = (rect*)malloc(sizeof(rect));
    r->w = 0;
    r->h = 0;
    r->set = set;
    r->area = area;
    return r;
}
```

함수의 포인터

- 구조체 속에 함수를 정의할 수 없지만, 함수의 포인터를 정의하여 그 포인터가 외부 함수를 가리키도록 할 수 있음

```
typedef struct Rect {
    int w,  h;
    void (*set)(struct Rect*, int, int);
    int (*area)(struct Rect*);
} rect;
```

```
void set(rect *r, int w, int h) {
    r->w = w;
    r->h = h;
}

int area(rect *r) {
    return r->w*r->h;
}
```

```
rect *createRect() {
    rect *r = (rect*)malloc(sizeof(rect));
    r->w = 0;
    r->h = 0;
    r->set = set;
    r->area = area;
    return r;
}
```

함수의 포인터

- 구조체 속에 함수를 정의할 수 없지만, 함수의 포인터를 정의하여 그 포인터가 외부 함수를 가리키도록 할 수 있음

```
int main() {
    rect *r = createRect();
    r->set(r, 10, 5);
    printf("%d\n", r->area(r)); // 50 출력
    return 0;
}
```

함수 포인터의 배열

- 함수의 포인터로 이루어지는 배열을 활용하면 어떤 상황에 알맞은 함수를 선택할 수 있음

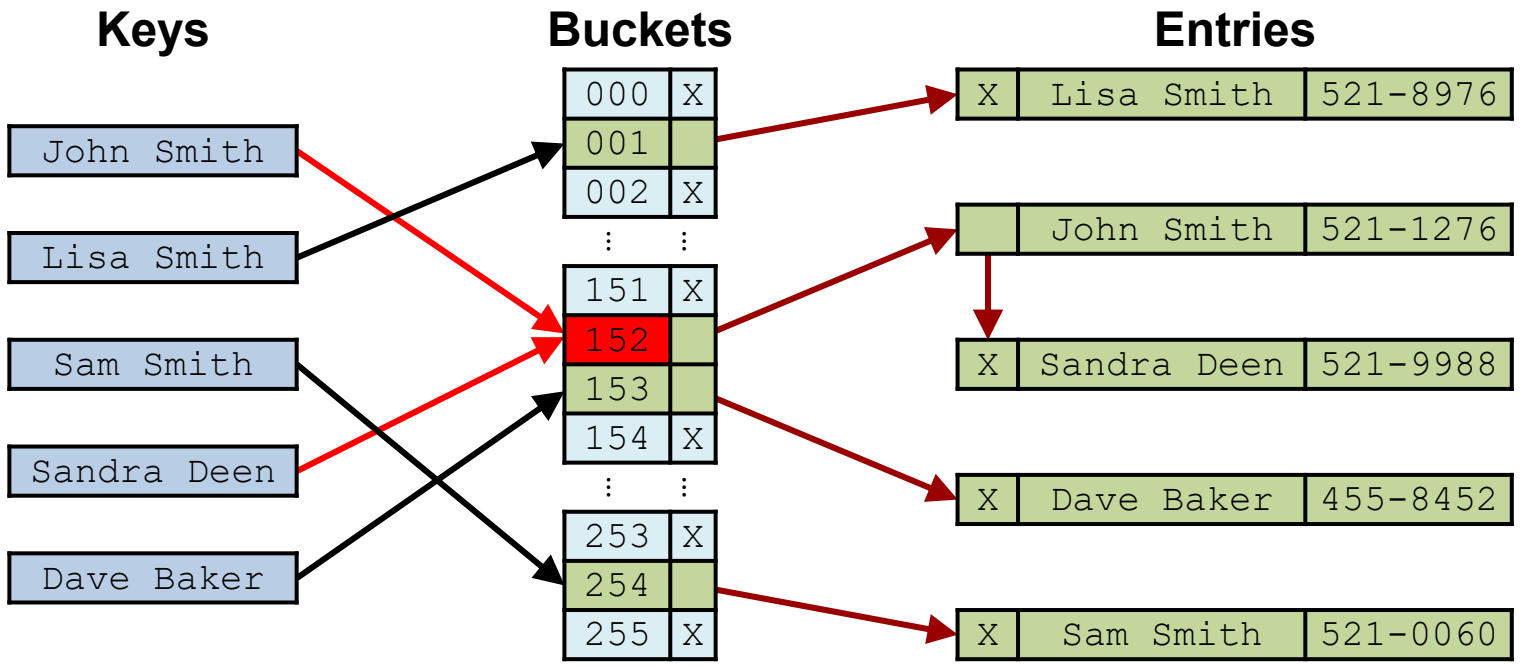
```
enum TYPE{SQUARE, RECT, CIRCLE, POLY};
typedef struct {
    …
    enum TYPE type;
} shape;

typedef void (*fp)(shape *ps) drawfn;
drawnfn fp[4] = {&draw_square, &draw_rect, &draw_circle, &draw_poly};

shape *ps;
(*fp[ps->type])(ps); // ps의 type에 따라 해당되는 함수를 호출함
```

해시 테이블

- 해시 테이블(hash table) 또는 해시 맵(hash map)은 보통 연결리스트로 이루어지는 배열로 구현됨



해시 테이블

- 배열의 각 요소에 인덱스를 알아낼 수 있는 키를 할당함
- 해시 함수(hash function)는 키-인덱스 변환을 담당함
- 충돌(hash collision)은 서로 다른 키를 갖고 있지만 같은 인덱스로 매핑된 것임
- 충돌 가능성을 줄이기 위해 해시 함수가 **균일 분포**에 따르면 좋음
- 특정 요소 삽입, 삭제, 탐색이 `O(1)` 이며, 최악의 경우는 `O(n)` 이 됨

해시 테이블

- 예로, 해시 테이블 만들기

```
#define MAX_BUCKETS 1000
#define MULTIPLIER 31

typedef struct wordrec {
    char *word;
    unsigned long count;
    struct wordrec *next;
} wordrec;

wordrec *table[MAX_BUCKETS];
```

해시 테이블

- 해시 함수 정의

```
unsigned long hashstring(const char *str) {
    unsigned long hash = 0;
    while (*str) {
        hash = hash * MULTIPLIER + *str;
        str++;
    }
    return hash%MAX_BUCKETS;
}
```

해시 테이블

- 해시 테이블 탐색

```
wordrec *lookup(const char *str, int create) {
    wordrec *curr = NULL;
    unsigned long hash = hashstring(str);
    wordrec *wp = table[hash];
    for (curr=wp; curr!=NULL; curr=curr->next)
        if (strcmp(str, curr->word)) return curr;
    // 찾고 싶은 것이 테이블에서 없는 경우
    if (create) {
        // 테이블에 추가
    }
    return curr;
}
```