자료구조 (Data Structure)
1주차: 배열 리스트

지난 시간 요약

- 배열 리스트 정보: data / size / capacity
- 배열 리스트 기능: INSERT / DELETE / RETRIEVE
- 가변 크기 아이디어(RESIZE)와 다항식 예제

오늘 목표

- 1) ArrayList 기능 구현
- 2) 다항식 구현

필수 헤더

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

배열 리스트

- 데이터를 배열에 순서대로 저장한다
- 저장된 데이터 수와 최대 저장량을 기록해둔다
- data: 배열
- size: 저장된 데이터 수
- capacity: 최대 저장량

배열 리스트

```
struct ArrayList {
        int *data;
        int size;          /* 현재 원소 수 */
        int capacity;  /* 최대 수용량 */
};
```

INITIALIZE(A, cap)

- cap개 데이터를 저장가능하게 A를 초기화한다
- 조건: cap > 0
- 동작: A.data ← 새로운 배열(크기 : cap + 1)

  A.size ← 0

  A.capacity ← cap

INITIALIZE(A, cap)

```
void arraylist_initialize(struct ArrayList *A, int cap)
{
    if (cap <= 0) {
        fprintf(stderr, "initialize: invalid cap\n");
        exit(1);
    }
    if (!A) {
        fprintf(stderr, "initialize: A is null\n");
        exit(1);
    }
```

INITIALIZE(A, cap)

```
    A->data = (int*)malloc(sizeof(int) * (cap + 1));
    if (!A->data) {
        fprintf(stderr, "initialize: malloc failed\n");
        exit(1);
    }

    A->size = 0;
    A->capacity = cap;
}
```

INSERT(A, k, val)

- A의 k번째 위치($1 \le k \le A.size + 1$)에 val을 추가한다
- 조건: $A.size < A.capacity$
- 동작: A.data[A.size부터 k까지 역순]를 뒤로 이동

  $A.data[k] \leftarrow x$

  $A.size \leftarrow A.size + 1$

INSERT(A, k, val)

```
void arraylist_insert(struct ArrayList *A, int k, int val)
{
    if (k < 1 || k > A->size + 1) {
        fprintf(stderr, "insert: invalid k\n");
        exit(1);
    }
    if (A->size >= A->capacity) {
        fprintf(stderr, "insert: array is full\n");
        exit(1);
    }
```

INSERT(A, k, val)

```
    for (int i = A->size; i >= k; --i)
        A->data[i + 1] = A->data[i];

    A->data[k] = val;
    A->size++;
}
```

DELETE(A, k)

- A의 k번째 위치(1 ≤ k ≤ A.size)의 원소를 삭제한다
- 조건: A.size > 0
- 동작: A.data[k부터 A.size-1까지]를 앞으로 이동

  A.size ← A.size - 1

DELETE(A, k)

```
void arraylist_delete(struct ArrayList *A, int k)
{
    if (k < 1 || k > A->size) {
        fprintf(stderr, "delete: invalid k\n");
        exit(1);
    }
    if (A->size <= 0) {
        fprintf(stderr, "delete: no data to delete\n");
        exit(1);
    }
```

DELETE(A, k)

```
    for (int i = k; i < A->size; ++i)
        A->data[i] = A->data[i + 1];

    A->size--;
}
```

RETRIEVE(A, k)

- A의 k번째 위치($1 \le k \le A.size$)의 원소를 반환한다
- 조건: $A.size > 0$
- 동작: A.data[k]를 반환

RETRIEVE(A, k)

```
int arraylist_retrieve(struct ArrayList *A, int k)
{
	if (k < 1 || k > A->size) {
		fprintf(stderr, "retrieve: invalid k\n");
		exit(1);
	}
	if (A->size <= 0) {
		fprintf(stderr, "retrieve: no data to retrieve\n");
		exit(1);
	}
```

RETRIEVE(A, k)

```
    return A->data[k];
}
```

RESIZE(A)

- A의 capacity를 두배 늘리고, 배열을 새로 만든다
- 기존 데이터는 새로운 배열에 모두 복사한다
- 동작: A.capacity ← A.capacity * 2

  tmp ← 새로운 배열(크기 : A.capacity + 1)

  A.data에서 tmp로 A.size개 데이터 복사

  A.data ← tmp

RESIZE(A)

```
void arraylist_resize(struct ArrayList *A)
{
    int new_cap = A->capacity * 2;
    int *new_data
        = (int*)malloc(sizeof(int) * (new_cap + 1));
    if (!new_data) {
        fprintf(stderr, "resize: malloc failed\n");
        exit(1);
    }
```

RESIZE(A)

```
    for (int i = 1; i <= A->size; ++i)
        new_data[i] = A->data[i];

    int *old_data = A->data;

    A->data = new_data;
    A->capacity = new_cap;
    free(old_data);
}
```

INSERT_VAR(A, k, val)

- A의 k번째 위치(1 ≤ k ≤ A.size + 1)에 원소 val을 추가한다

- 동작: 만약 A.size == A.capacity면 RESIZE(A)

  INSERT(A, k, val)

INSERT_VAR(A, k, val)

```
void arraylist_insert_var(struct ArrayList *A, int k
                                        , int val)
{
    if (A->size == A->capacity)
        arraylist_resize(A);

    arraylist_insert(A, k, val);
}
```

예제: 다항식 저장하기

- $2x^2 + 3x$를 저장하려면,
- 1. 항 리스트를 만든다 (계수와 지수 저장)
- 2. 항 리스트를 크기 2로 초기화한다
- 3. 항 리스트에 계수 2, 지수 2을 추가한다
- 4. 항 리스트에 계수 3, 지수 1을 추가한다

고정 크기 항 리스트

- coeff: 계수를 저장할 배열
- exp: 지수를 저장할 배열
- size: 저장된 항의 개수
- capacity: 최대 저장량

고정 크기 항 리스트

```
struct TermList {
        int *coeff;       // 계수 데이터
        int *exp;         // 지수 데이터
        int size;         // 현재 원소 수
        int capacity;  // 최대 수용량
};
```

INITIALIZE(A, cap)

- A.coeff ← 새로운 배열(크기 : cap + 1)
- A.exp ← 새로운 배열(크기 : cap + 1)
- A.capacity ← cap
- A.size ← 0

INITIALIZE(A, cap)

```
void termlist_initialize(struct TermList *A, int cap)
{
    A->coeff = (int*)malloc(sizeof(int) * (cap + 1));
    A->exp = (int*)malloc(sizeof(int) * (cap + 1));

    if (!A->coeff || !A->exp) {
        fprintf(stderr, "initialize: malloc failed\n");
        exit(1);
    }
```

INITIALIZE(A, cap)

```
    A->size = 0;
    A->capacity = cap;
}
```

INSERT(A, k, new_coeff, new_exp)

반복: i는 A.size부터 k까지 역순으로

A.coeff[i + 1] ← A.coeff[i]

A.exp[i + 1] ← A.exp[i]

A.coeff[k] ← new_coeff

A.exp[k] ← new_exp

A.size ← A.size + 1

INSERT(A, k, new_coeff, new_exp)

```
void termlist_insert(struct TermList *A, int k
                                , int new_coeff, int new_exp)
{
      for (int i = A->size; i >= k; --i) {
            A->coeff[i + 1] = A->coeff[i];
            A->exp[i + 1]   = A->exp[i];
      }
```

INSERT(A, k, new_coeff, new_exp)

```
    A->coeff[k] = new_coeff;
    A->exp[k]   = new_exp;
    A->size++;
}
```

ADD(X, Y) 1/6

- A ← 새로운 항 리스트
- INITIALIZE(A, X.size + Y.size)
- iX ← 1, iY ← 1
- 반복: iX ≤ X.size 그리고 iY ≤ Y.size

ADD(X, Y) 1/6

```
struct TermList *termlist_add(struct TermList *X
                                        , struct TermList *Y)
{
    struct TermList *A
        = malloc(sizeof(struct TermList));

    if (!A) return NULL;

    termlist_initialize(A, X->size + Y->size);
```

ADD(X, Y) 1/6

```
int iX = 1;
int iY = 1;

while (iX <= X->size && iY <= Y->size) {
```

ADD(X, Y) 2/6

만약: X.exp[iX] > Y.exp[iY]

INSERT(A, A.size + 1, X.coeff[iX], X.exp[iX])

iX ← iX + 1

다음 반복으로

ADD(X, Y) 2/6

```
if (X->exp[iX] > Y->exp[iY]) {
        termlist_insert(A,
                                A->size + 1,
                                X->coeff[iX],
                                X->exp[iX]);
        ++iX;
        continue;
}
```

ADD(X, Y) 3/6

만약: X.exp[iX] < Y.exp[iY]

INSERT(A, A.size + 1, Y.coeff[iY], Y.exp[iY])

iY ← iY + 1

다음 반복으로

ADD(X, Y) 3/6

```
if (X->exp[iX] < Y->exp[iY]) {
        termlist_insert(A,
                                A->size + 1,
                                Y->coeff[iY],
                                Y->exp[iY]);
        ++iY;
        continue;
}
```

ADD(X, Y) 4/6

만약: X.exp[iX] == Y.exp[iY]

new_coeff ← X.coeff[iX] + Y.coeff[iY]

INSERT(A, A.size + 1, new_coeff, X.exp[iX])

iX ← iX + 1

iY ← iY + 1

다음 반복으로

ADD(X, Y) 4/6

```
        /* 같을 때: 계수 합침 (합이 0이면 항 생략) */
        int new_coeff = X->coeff[iX] + Y->coeff[iY];
        if (new_coeff != 0)
            termlist_insert(A,
                            A->size + 1,
                            new_coeff,
                            X->exp[iX]);
        ++iX;
        ++iY;
}
```

ADD(X, Y) 5/6

만약: iX ≤ X.size

반복: i는 iX부터 X.size까지

INSERT(A, A.size + 1, X.coeff[iX], X.exp[iX])

ADD(X, Y) 5/6

```
/* X 잔여분 */
if (iX <= X->size) {
        for (int i = iX; i <= X->size; i++)
                termlist_insert(A,
                                A->size + 1,
                                X->coeff[i],
                                X->exp[i]);
}
```

ADD(X, Y) 6/6

- 만약: iY ≤ Y.size
- 반복: i는 iY부터 Y.size까지
- INSERT(A, A.size + 1, Y.coeff[iY], Y.exp[iY])
- 반환: A

ADD(X, Y) 6/6

```
    /* Y 잔여분 */
    if (iY <= Y->size) {
        for (int i = iY; i <= Y->size; i++)
            termlist_insert(A,
                            A->size + 1,
                            Y->coeff[i],
                            Y->exp[i]);
    }
    return A;
}
```

요약

- 고정 크기 배열 리스트 C로 구현
- 가변 크기 배열 리스트 C로 구현
- 다항식 예제 C로 구현
- 다음 시간: 링크드 리스트

(강의 컨설팅 관계로 강의 촬영)

C 코딩 스타일

- 리눅스 커널 코딩 스타일
- 이유: 세계에서 가장 큰 C 프로젝트
- C 코딩을 매우 많이 하고, 많이 읽은 프로그래머들이 정한 스타일
- https://github.com/torvalds/linux/blob/master/Documentation/process/coding-style.rst

C 코딩 스타일 - 들여쓰기

- 들여쓰기: 8칸
- 이유: 들여쓰기 되어 있음을 확실하게 볼 수 있다.
- #include <stdio.h>

```
int main()
{
	printf("Hello, world!");
}
```

C 코딩 스타일 - switch문의 들여쓰기

- case 라벨의 들여쓰기: 없음
- 이유: 지나친 들여쓰기를 방지한다.

```
switch (suffix) {
case 'G':
case 'g':
        mem <<= 30;
        break;
default:
        break;
}
```

C 코딩 스타일 - 함수의 중괄호

- 중괄호 여는 위치: 헤더 다음 줄
- 이유: 함수는 특별하다. C에서는 함수 정의 안에 함수 정의를 못한다.
- C 언어 제작자들이 쓰는 방식

```
int function(int x)
{
    /* body of function */
}
```

C 코딩 스타일 - 모든 문장 블록의 중괄호

- 중괄호 여는 위치: 줄의 오른쪽 끝
- 중괄호 닫는 위치: 줄의 왼쪽 끝 (대부분의 경우 다른 문자 없음)

```
if (x_is_true) {
    do_y();
    do_z();
}
```

C 코딩 스타일 - do-문의 중괄호

- 중괄호를 닫고 나서 같은 줄에 while을 쓴다.

```
do {
        body of do-loop
} while (condition);
```

C 코딩 스타일 - if-문의 중괄호

- 중괄호를 닫고 나서 같은 줄에 else를 쓴다.

```
if (x == y) {
        ..
} else if (x > y) {
        ...
} else {
        ....
}
```

C 코딩 스타일 - if-문의 중괄호

- if와 else에서 모두 문장 하나만 실행할 때는 중괄호를 쓰지 않는다.

```
if (condition)
    do_this();
else
    do_that();
```

C 코딩 스타일 - if-문의 중괄호 (나쁜 예)

- if의 문장을 숨길 것이 아닌 한 한줄에 쓰지 않는다.

```
if (condition) do_this;
do_something_everytime;
```

C 코딩 스타일 - if-문의 중괄호 (나쁜 예)

- 괄호를 쓰지 않기 위해서 쉼표를 쓰지 않는다.

```
if (condition)
        do_this(), do_that();
```

C 코딩 스타일 - if-문의 중괄호

- if나 else 중 하나에만 문장이 두개 이상 있어도 모두 중괄호를 쓴다.

```
if (condition) {
        do_this();
        do_that();
} else {
        otherwise();
}
```

C 코딩 스타일 - 반복문의 중괄호

- 반복문은 단순 문장 한개만 있지 않은 한 항상 중괄호를 쓴다.

```
while (condition) {
    if (test)
        do_something();
}
```

C 코딩 스타일 - 띄어쓰기 - 키워드

- 다음 키워드 다음에는 띄어쓰기를 한칸 한다.

  if, switch, case, for, do, while

- sizeof 다음에는 띄어쓰기를 하지 않는다.

  좋은 예: s = sizeof(struct file);

- 괄호 안쪽 주변에는 띄어쓰기를 하지 않는다.

  나쁜 예: s = sizeof(struct file);

C 코딩 스타일 - 띄어쓰기 - 포인터

- 포인터 변수를 선언하거나, 포인터를 반환하는 함수를 선언할 때,
  - * 은 변수 이름이나 함수 이름에 붙인다.
  - * 을 타입 옆에 붙이지 않는다.

```
char *linux_banner;

unsigned long long memparse(char *ptr,
                            char **retptr);

char *match_strdup(substring_t *s);
```

C 코딩 스타일 - 띄어쓰기 - 이항연산자

- 다음 이항 연산자 / 삼항 연산자 주변에는 띄어쓰기를 한칸 한다.

  = + - * / % < > <= >= == != & | ? :

- 다음 구조체 멤버 연산자 주변에는 띄어쓰기를 하지 않는다.

  . ->

- area = rect.x * rect.y;

C 코딩 스타일 - 띄어쓰기 - 단항연산자

- 다음 단항 연산자 뒤에는 띄어쓰기를 하지 않는다.

  & * + - ~ ! sizeof

- 다음 postfix 증감 단항 연산자 주변에는 띄어쓰기를 하지 않는다.

  ++ --

- 다음 prefix 증감 단항 연산자 주변에는 띄어쓰기를 하지 않는다.

  ++ --

C 코딩 스타일 - 이름

- C 언어는 단순하고 미니멀한 것을 추구한다.
- 어떤 프로그래밍 언어에서는 ThisVariableIsATemporaryCounter로 쓰지만
- C 언어에서는 쓰기 훨씬 쉽고, 이해하기 많이 어렵지 않은 tmp로 쓴다.
- 한편, 전역 범위에 있는 이름은 자세해야 한다.

C 코딩 스타일 - 전역 변수 이름, 함수 이름

- 전역 변수는 다른 방법이 전혀 없어서 어쩔 수 없이 꼭 필요할 때만 쓴다.
- 전역 변수와 함수의 이름은 단어들을 _로 구분해서 자세하게 만든다.
- 좋은 예: count_active_users()
- 나쁜 예: cntusr()

C 코딩 스타일 - 지역 변수 이름

- 지역 변수 이름은 짧고, 정보를 담고 있어야 한다.
- 좋은 예: 루프 변수 i
- 나쁜 예: 루프 변수 loop_counter (명확한 상황에서)
- 좋은 예: 임시 변수 tmp

C 코딩 스타일 - 함수

- 함수는 짧고 한가지 일만 해야 한다.
- 함수에서 지역 변수를 10개 이하로 써야 한다.
- 함수 정의들 간에는 한줄을 띄워서 쓴다.
- 함수 선언에서 타입과 이름을 모두 쓴다.

C 코딩 스타일 - 함수 관련 goto

- goto는 많은 프로그래머들이 더이상 쓰지 않는다.

- 여러 군데에서 함수가 종료되는데 공통된 정리 코드가 있을 때 유용하다.

```
int fun(int a)
{
	int result = 0;
	char *buffer;
```

C 코딩 스타일 - 함수 관련 goto

```
buffer = malloc(SIZE);
if (!buffer)
        return -ENOMEM;

if (condition1) {
        while (loop1) {
                ...
        }
        result = 1;
        goto out_free_buffer;
}
...
```

C 코딩 스타일 - 함수 관련 goto

```
out_free_buffer:
        free(buffer);
        return result;
}
```

C 코딩 스타일 - 함수 관련 goto 에러

- one err bugs로 알려진 흔한 버그

```
err:
        free(foo->bar);
        free(foo);
        return ret;
```

- foo가 NULL인 상태에서 err로 이동했을 때 버그 발생

C 코딩 스타일 - 함수 관련 goto 에러 해결책

- err_free_bar:와 err_free_foo:로 라벨을 분리한다.

```
err_free_bar:
        free(foo->bar);
err_free_foo:
        free(foo);
        return ret;
```

- foo가 NULL이 아닐 때만 err_free_bar로 이동

C 코딩 스타일 - 주석

- 주석은 함수 앞에서 무엇을 왜 하는지 설명한다.
- 아주 복잡한 함수의 경우 본문 안에도 구획을 나눠서 주석을 달 수 있다.

```
/*
 * This is the preferred style for multi-line
 * comments in the Linux kernel source code.
 */
```

- 데이터에 대해 짧은 주석을 다는 것도 중요하다.

C 복습 - 이름의 범위

```
int x = 0;
void function(void)
{
    int a = x;        /* 전역 x(=0) */
    int x = 1;        /* 함수 지역 x */
    for (int x = 2; x < 10; x++) {
        int b = x;  /* for-블록의 x */
    }
    a = x;          /* 여기서의 x는 함수 지역 x(=1) */
} /* 함수가 끝나기 전 a에 저장된 값은? */
```

C 복습 - 배열.c

- #include <stdio.h>

```
void array_test()
{
	int arr[3];

	arr[0] = 1;
	arr[1] = 2;
	arr[2] = 3;
```

C 복습 - 배열.c

```
    for (int i = 0; i < 3; i++)
        printf("%d ", arr[i]);

    printf("\n");
}
```

- array_test()를 실행했을 때 화면에 출력되는 문자열은?

C 복습 - 배열_포인터.c

```
#include <stdio.h>


void array_ptr_test()
{
	int arr[3];
	int *arr_ptr = arr;


	arr_ptr[0] = 1;
	arr_ptr[1] = 2;
	arr_ptr[2] = 3;
```

C 복습 - 배열_포인터.c

```
    for (int i = 0; i < 3; i++)
        printf("%d ", arr_ptr[i]);

    printf("\n");
}
```

- array_ptr_test()를 실행했을 때 화면에 출력되는 문자열은?

C 복습 - 배열_포인터_파라미터.c

```
#include <stdio.h>


void array_set_elements(int *arr_ptr)
{
	arr_ptr[0] = 1;
	arr_ptr[1] = 2;
	arr_ptr[2] = 3;
}
```

C 복습 - 배열_포인터_파라미터.c

```
void array_fun_test()
{
	int arr[3];
	array_set_elements(arr);
	for (int i = 0; i < 3; i++)
		printf("%d ", arr[i]);

	printf("\n");
}
```

- array_fun_test()를 실행했을 때 화면 출력은?

C 복습 - 구조체.c

```
#include <stdio.h>

struct Point {
	int x;
	int y;
};
```

C 복습 - 구조체.c

```
void struct_test()
{
	struct Point p;
	p.x = 1;
	p.y = 2;
	printf("(%d, %d)\n", p.x, p.y);
}
```

- struct_test()를 실행했을 때 화면에 출력되는 문자열은?

C 복습 - 구조체_포인터.c

```
#include <stdio.h>

struct Point {
	int x;
	int y;
};
```

C 복습 - 구조체_포인터.c

```
void struct_ptr_test()
{
	struct Point p;
	struct Point *p_ptr = &p;
	p_ptr->x = 1;
	p_ptr->y = 2;
	printf("(%d, %d)\n", p_ptr->x, p_ptr->y);
}
```

- struct_ptr_test()를 실행했을 때 화면에 출력되는 문자열은?

C 복습 - 구조체_포인터_파라미터.c

```
#include <stdio.h>

struct Point {
	int x;
	int y;
};
```

C 복습 - 구조체_포인터_파라미터.c

```
void struct_set_members(struct Point *p_ptr)
{
	p_ptr->x = 1;
	p_ptr->y = 2;
}
```

C 복습 - 구조체_포인터_파라미터.c

```
void struct_ptr_parameter_test()
{
	struct Point p;
	struct_set_members(&p);
	printf("(%d, %d)\n", p.x, p.y);
}
```

struct_ptr_parameter_test()를 실행했을 때 화면에 출력되는 문자열은?

C 복습 - 메모리_할당_해제.c

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

struct Point {
	int x;
	int y;
};
```

C 복습 - 메모리_할당_해제.c

```
struct Point *create_point()
{
    return malloc(sizeof(struct Point));
}

void free_point(struct Point *p_ptr)
{
    free(p_ptr);
}
```

C 복습 - 메모리_할당_해제.c

```
void struct_set_members(struct Point *p_ptr)
{
	p_ptr->x = 1;
	p_ptr->y = 2;
}
```

C 복습 - 메모리_할당_해제.c

```
void struct_ptr_malloc_test()
{
    struct Point* p_ptr = create_point();
    if (!p_ptr)
        return;
    struct_set_members(p_ptr);
    printf("(%d, %d)\n", p_ptr->x, p_ptr->y);
    free_point(p_ptr);
}
```

- struct_ptr_malloc_test() 실행 후 화면 출력은?

C 키워드 _Bool

- 0 또는 1을 저장할 수 있는 타입
- 거짓 또는 참을 저장하기 위해 쓰임