CHAPTER 2

ARRAYS AND STRUCTURES

ARRAYS AND STRUCTURES

2.1 Arrays

2.2 Dynamically Allocated Array

2.3 Structures and Unions

2.4 Polynomials

2.5 Sparse Matrices

2.6 Representation of Multidimensional Arrays

Array

- A consecutive set of memory locations
  - emphasis on implementation issues

- A set of pairs, <*index*, *value*>
  - set of *mappings* (or *correspondence*) between index and values
  - *array* : $i \rightarrow a_i$

ADT *Array* is

objects: A set of pairs <*index*, *value*> where for each value of *index* there is a value from the set *item*. *Index* is a finite ordered set of one or more dimensions, for example, {0, $\cdots$, n–1} for one dimension, {(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)} for two dimensions, etc.

functions:

for all $A \in$ *Array*, $i \in$ *index*, $x \in$ *item*, j, *size* $\in$ integer

Array Create(*j*, *list*)	::=	**return** an array of *j* dimensions where *list* is a *j*-tuple whose *i*th element is the the size of the *i*th dimension. *Items* are undefined.
Item Retrieve(*A*, *i*)	::=	**if** ($i \in$ *index*) **return** the item associated with index value *i* in array *A* **else return** error
Array Store(*A*,*i*,*x*)	::=	**if** (*i* in *index*) **return** an array that is identical to array *A* except the new pair <*i*, *x*> has been inserted **else return** error.

end *Array*

ADT 2.1: Abstract Data Type *Array*

Arrays in C

- One-dimensional array in C

```
int list[5], *plist[5];
```

- Arrays start at index 0 in C
  - list[0], list[1], list[2], list[3], list[4]: …
  - plist[0], plist[1], plist[2], plist[3], plist[4]: …

Variable	Memory address
list[0]	base address = α
list[1]	α + sizeof(int)
list[2]	α + 2·sizeof(int)
list[3]	α + 3·sizeof(int)
list[4]	α + 4·sizeof(int)

- Compare list1 and list2 in C:

```
int *list1, list2[5];
```

  - Same: .... 둘다 포인터!
  - Difference: .... list1은 변수로 list2는 상수!

- Notations:

```
list2 =
(list2+i) =
*(list2+i) =
```

list2 = list2[0]의 주소 값
(list2+i) = list2[i]의 주소 값
*(list2+i) = list2[i]의 값
↳ 변수가 하는일 그대로 할 수있다

```
#define MAX_SIZE 100
float sum(float [], int);
float input[MAX_SIZE], answer;
void main(void)
{
   int i;
   for (i = 0; i < MAX_SIZE; i++)
      input[i] = i;
   answer = sum(input, MAX_SIZE);
   printf("The sum is: %f\n", answer);
}
float sum(float list[], int n)
{
   int i;
   float tempsum = 0;
   for (i = 0; i < n; i++)
      tempsum += list[i];
   return tempsum;
}
```

Program 2.1: Example array program

```
118 float sum(float list[], int n)
119 {
120    int i;
121    float tempsum = 0;
```

100 %

지역

이름	값	형식
list	0x00047560 _input	float *
0.00000000		float
n	100	int

호출 스택

이름
Chapter 02.exe!sum(flo
Chapter 02.exe!main()
Chapter 02.exe!__tmain
Chapter 02.exe!mainCR
kernel32.dll!771f919f()

자동 지역 스레드 모듈 조사식 1 호출 스택 중단점

parameter passing : ...

↳ 주소는 배열이지만, 사실 주소값을 parameter로 받음

lvalue vs rvalue:

list[i] = … ; ← 메모리에 저장

… = list[i] ; ← value 값을 가져옴

- Ex 2.1) [1D array addressing]
  Write a function that prints out as below:

```
int one[] = {0, 1, 2, 3, 4};
```

Address	Contents
12244868	0
12344872	1
12344876	2
12344880	3
12344884	4

```
int one[] = {0, 1, 2, 3, 4};
```

Address	Contents
12244868	0
12344872	1
12344876	2
12344880	3
12344884	4

```
void print1 (int *ptr, int rows)
{
    int i;
    printf ("Address Contents\n");
    for (i=0; i<rows; i++)
        printf( "%8u%5d\n", ptr + i, *(ptr + i) );
    printf("\n");
}
```

The function is invoked as …

ARRAYS AND STRUCTURES

2.1 Arrays

2.2 Dynamically Allocated Array

2.3 Structures and Unions

2.4 Polynomials

2.5 Sparse Matrices

2.6 Representation of Multidimensional Arrays

One-Dimensional Arrays

```
int i, n, *list;
printf("Enter the number of number to generate: ");
scanf("%d", &n");
if (n < 1) {
      fprintf(stderr, "Improper value of n \n");
      exit (EXIT_FAILURE);
}
MALLOC( list, n*sizeof(int) );
```

```
#define MALLOC(p,s) \
   if (!((p) = malloc(s))) {\
      fprintf(stderr, "Insufficient memory"); \
      exit(EXIT_FAILURE);\
   }
```

Two-Dimensional Arrays

- To represent a multidimensional array, C uses the array of arrays representation

- 2D Array
  int x[3][5];
  - Represented as a 1D array in which each element is, itself, a 1D array

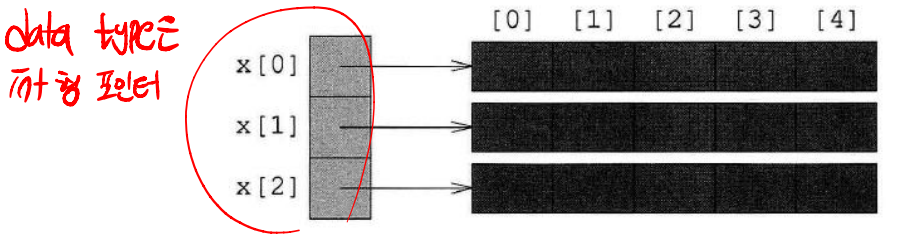


Figure 2.2: Array-of-arrays representation

int x[3][5];

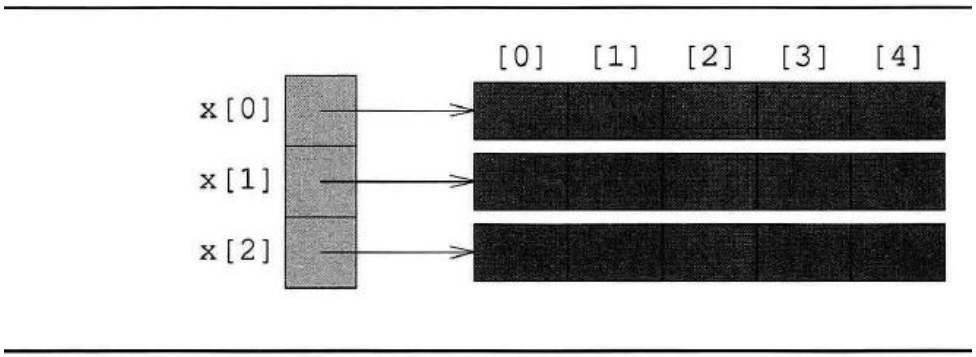


Figure 2.2: Array-of-arrays representation

What is the element x[i]?

→ ... x[i]의 원소는 i행의 0번째 원소의 주소값을 가지고있음

How C finds the element x[i][j]?

→ ...

① 행 접근
② 열 접근
) MATRIX로 생각할때..

int x[3][5];

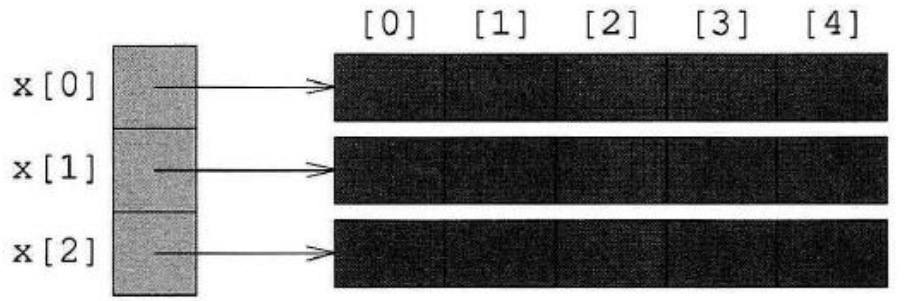


Figure 2.2: Array-of-arrays representation

dynamic memory allocation for 2d-array: …

int x[3][5];

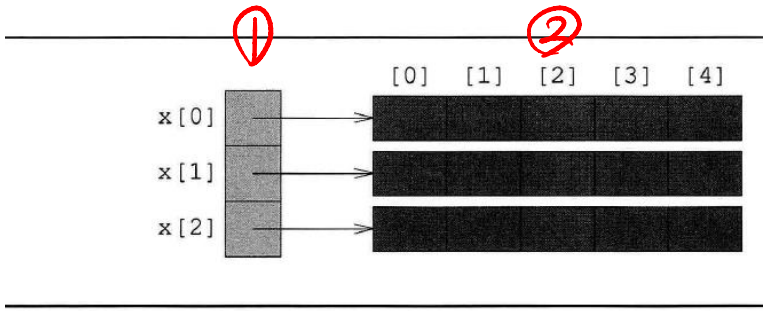
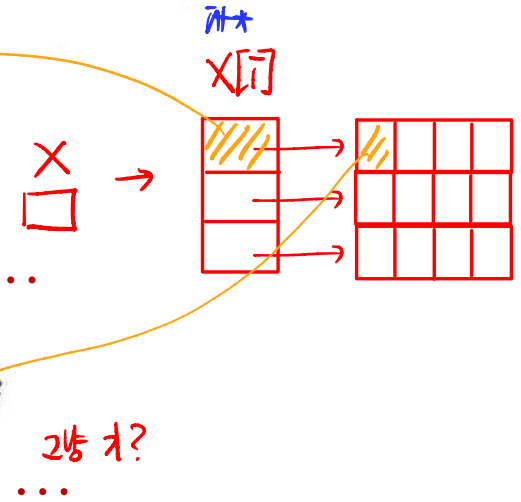


Figure 2.2: Array-of-arrays representation

```
int** make2dArray(int rows, int cols)
{/* create a two dimensional rows × cols array */
   int **x, i;

   /* get memory for row pointers */
   MALLOC(x, rows * sizeof (*x));

   /* get memory for each row */
   for (i = 0; i < rows; i++)
     MALLOC(x[i], cols * sizeof(**x));
   return x;
}
```



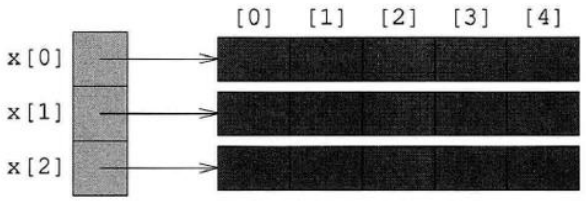
Program 2.3: Dynamically create a two-dimensional array

```
int** make2dArray(int rows, int cols)
{/* create a two dimensional rows × cols array */
   int **x, i;

   /* get memory for row pointers */
   MALLOC(x, rows * sizeof (*x));;

   /* get memory for each row */
   for (i = 0; i < rows; i++)
     MALLOC(x[i], cols * sizeof(**x));
   return x;
}
```

Program 2.3: Dynamically create a two-dimensional array



This function may be used in the following way :

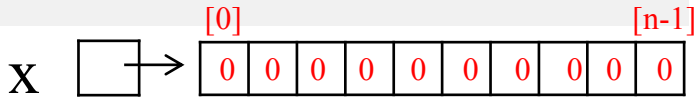
```
int **myArray;
myArray = make2dArray(5, 10);
myArray[2][4] = 6
```

- calloc/realloc
  - Memory allocation functions

- ex)

```
int *x;
x = (int *) calloc(n, sizeof(int));
```

→ 모든 bit를 0으로 초기화




```
#define CALLOC(p, n, s)\
        if(!((p) = calloc(n, s))) {\
                fprintf(stderr, "Insufficient memory");\
                exit(EXIT_FAILURE);\
        }
```

```
CALLOC( x, n, sizeof(int) );
```

```
#define REALLOC(p, s)\
        if(!((p) = realloc(p, s))) {\
                fprintf(stderr, “Insufficient memory”):\
                exit(EXIT_FAILURE):\
        }
```

ARRAYS AND STRUCTURES

2.1 Arrays

2.2 Dynamically Allocated Array

2.3 Structures and Unions

2.4 Polynomials

2.5 Sparse Matrices

2.6 Representation of Multidimensional Arrays

Structures (Records)

- Arrays
  - Collections of data of the same type
- Structure
  - Collection of data items
  - Permits the data to vary in type

```
struct {
        char name[10];
        int age;
        float salary;
} person;
```

…

```
strcpy(person.name, “james”);
person.age=10;
person.salary=35000;
```

. 도트연산자 → 구조체 멤버에 접근

→ 화살표연산자 → 구조체 포인터를 이용하여 접근

- Create structure data type using **typedef**

```
typedef struct {
        char name[10];
        int age;
        float salary;
} human_being;
```

…

typedef를 주는것만 char, float과 같은 데이터 타입이 됨

- Declare variables

human_being person1, person2 ;

```
if ( strcmp(person1.name, person2.name) )
        printf("not same name\n");
else
        printf("same name\");
```

- Structure assignment :
  person1 = person2;
  - ANSI C: OK!
  - older versions of C: NOT OK!

```
strcpy(person1.name, person2.name);
person1.age = person2.age;
person1.salary = person2.salary;
```

- Check of equality or inequality :
  if (person1==person2)
  - Cannot be checked directly

- Function to check equality of structures:

```
#define FALSE 0
#define TRUE 1
int humansEqual (humanBeing person1, humanBeing person2)
 {
        if (strcmp(person1.name, person2.name))
                return FALSE;
        if (person1.age != person2.age)
                return FALSE;
        if (person1.salary != person2.salary)
                return FALSE;
        return TRUE;
}
```

```
if ( humansEqual(person1,person2) ) …
else …
```

- A structure within a structure :

```
typedef struct {
        int month;
        int day;
        int year;
} date;

typedef struct {
        char name [10];
        int age;
        float salary;
        date dob;
} humanBeing;
```

```
humanBeing person1;
```

```
person1.dob.month = 12;
person1.dob.day = 8;
person1.dob.year = 1993;
```

Unions

- Similar to struct, but the fields must **share** their memory space;

```
union Data {
        int i;
        float f;
        char str[20];
};
```

- Only one field is “active” at any given time

(= #define female 0
#define male 1)

enum의 첫번째는 자동으로 0

```
typedef struct {
        enum tag_field {female, male} sex;
        union {
            int children;
            int beard;
        } u;
} sex_type;
```

```
typedef struct {
        char name[10];
        int age;
        float salary;
        date dob;
        sex_type sex_info;
} human_being;
```

```
human_being person1, person2;
```

```
person1.sex_info.sex=male;
person1.sex_info.u.beard=FALSE;

person2.sex_info.sex = female;
person2.sex_info.u.children = 4;
```

```c
#include <stdio.h>

enum Month {
    January = 1,
    February,
    March,
    April,
    May,
    June,
    July,
    August,
    September,
    October,
    November,
    December
};
```

```c
int main() {
    enum Month month;

    printf("Enter a month number (1-12): ");
    scanf("%d", &month);

    switch (month) {
        case January:
            printf("January has 31 days.\n");
            break;
        case February:
            printf("February has 28 or 29 days.\n");
            break;
        case March:
            printf("March has 31 days.\n");
            break;
        //  ...
        case October:
            printf("October has 31 days.\n");
            break;
        case November:
            printf("November has 30 days.\n");
            break;
        case December:
            printf("December has 31 days.\n");
            break;
        default:
            printf("Invalid month number.\n");
            break;
    }
    return 0;
}
```

Structures: Internal Implementation

- The size of an object of a struct
  - A multiple of 4, 8, or 16

실제는 9가 될 수 있지만
실제는 내 메모로 할당됨

```
typedef struct {
    char a;
    double d;
} st;
printf("%d\n", sizeof(st));
```

Self-Referential Structures

- One or more of its components is a **pointer to itself**

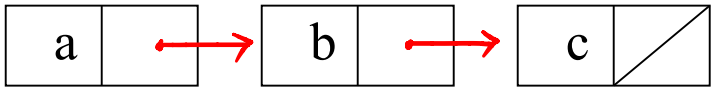
```
typedef  struct list{
         char data;
         struct list *link;
} list;
```

```
list item1, item2, item3;

item1.data='a';
item2.data='b';
item3.data='c';

item1.link=item2.link=item3.link=NULL;
```

```
// attach these together
```




```
item1.link = &item2;
item2.link = &item3;
```

ARRAYS AND STRUCTURES

2.1 Arrays

2.2 Dynamically Allocated Array

2.3 Structures and Unions

2.4 Polynomials

2.5 Sparse Matrices

2.6 Representation of Multidimensional Arrays