CHAPTER 1

BASIC CONCEPT

BASIC CONCEPT

1.2 Pointers and Dynamic Memory Allocation

1.3 Algorithm Specification

1.4 Data Abstraction

1.5 Performance Analysis

1.6 Performance Measurement

1.2.1 Pointers

For **any type *T*** in C
there is a corresponding type **pointer-to-*T***

The actual value of a point type is …

Two operators used with the pointers

- &: Address operator
- *: Dereferencing (or indirection) operator

역참조

Ex)

```
int i,  *pi;
pi = &i;
```

// To assign a value 10 to *i* ?

```
i = 10;
```

or

```
… *pi = 10
```

Null pointer → 아무것도 가리키지 않고있음

- Represented by the integer 0
- Points to no object or function

Ex)

int *pi = 0;

Ex) To test the null pointer

if (pi == **NULL**) …

if (!pi) …

↳ 매크로 상수로 0으로 #define 되어있음

1.2.2 Dynamic Memory Allocation

When is it required ?

- … 수행중에 필요한만큼만 메모리를 할당하기위함

Allocating storage at run-time

- called **heap** mechanism → 프로그램을 종료하기 전까지 사용가능
- Using a function, **malloc**

```c
int* arr;
int size, i;

printf("Enter the size of the array: ");
scanf("%d", &size);

arr = (int*)malloc(size * sizeof(int));

if (arr == NULL) {
    printf("Error: Unable to allocate memory.\n");
    exit(1);
}

for (i = 0; i < size; i++) {
    arr[i] = i + 1;
}
printf("Array elements are: ");
for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
}
free(arr);
```

```
int i, *pi;
float f, *pf;
pi = (int *) malloc(sizeof(int));
pf = (float *) malloc(sizeof(float));
*pi = 1024;
*pf = 3.14;
printf("an integer = %d, a float = %f\n", *pi, *pf);
free(pi);
free(pf);
```

포인터를 이용한 할당

Program 1.1: Allocation and deallocation of memory

More robust version for *malloc()*

```
if ((pi = (int *) malloc(sizeof(int))) == NULL ||
    (pf = (float *) malloc(sizeof(float))) == NULL)
{fprintf(stderr, "Insufficient memory");
 exit(EXIT_FAILURE);
}
```

or by the equivalent code

```
if (!(pi = malloc(sizeof(int))) ||
    !(pf = malloc(sizeof(float))))
{fprintf(stderr, "Insufficient memory");
 exit(EXIT_FAILURE);
}
```

A macro definition for *malloc()*

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

```
MALLOC(pi, sizeof(int));
MALLOC(pf, sizeof(float));
```

Dangling reference

```
int i, *pi;
float f, *pf;
pi = (int *) malloc(sizeof(int));
pf = (float *) malloc(sizeof(float));
*pi = 1024;
*pf = 3.14;
printf("an integer = %d, a float = %f\n", *pi, *pf);

pf = (float *) malloc(sizeof(float));
....
```

1.2.3 Pointers Can Be Dangerous

C언어 → 메모리에 직접 access 가능
장점이자 단점...

Why?

- …

Set all pointers to NULL NULL로 다 초기화해주고 체크하면서 사용
when they are not actually pointing to an object

- …

Explicit type cast
when converting between pointer types

Ex)

```
pi = malloc( sizeof(int) );
pf = (float *) pi;
```

int, float 둘 다 4byte
⇒ 이렇게 짜여진 명시적 형변환을 해주어야함

BASIC CONCEPT

1.2 Pointers and Dynamic Memory Allocation

1.3 Algorithm Specification

1.4 Data Abstraction

1.5 Performance Analysis

1.6 Performance Measurement

1.3.1 Introduction

Algorithm is a finite set of instructions that accomplishes a particular task

All algorithms must satisfy the following **criteria**:

- Input (≥ 0): …
- Output (≥ 1): …
- Definiteness: Each instruction is clear and unambiguous
- Finiteness: …
- Effectiveness: Every instruction must be carried out
- …

Ex)

- Program vs Algorithm …
- Flowchart …

How to describe an algorithm?

- Use a **natural language** like English
- Use graphic representations called **flowcharts**
- Here, we use Mostly in C
  - Occasionally in a combination of English and C

How to translating a problem into an algorithm?

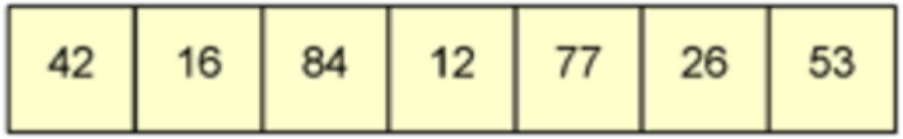
- Ex1.1, Ex1.2

정렬

Ex 1.1 Selection Sort

Devise a program that sorts a set of $n \geq 1$ integers;

– Assume that integers are stored in array *list*[*i*]

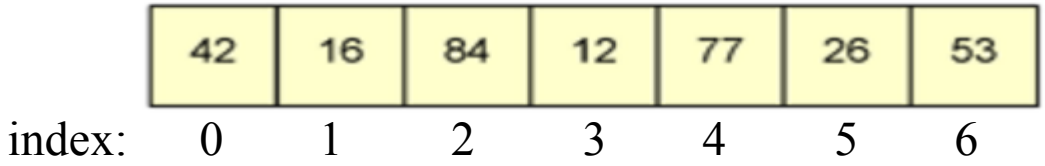


Selection sorting

A simple solution :

Find the smallest from those integers that are currently unsorted

Place it next in the sorted list




```
for (i = 0; i < n; i++) {
  Examine list[i] to list[n-1] and suppose that the
  smallest integer is  at list[min];

  Interchange list[i] and list[min];
}
```

Program 1.2: Selection sort algorithm

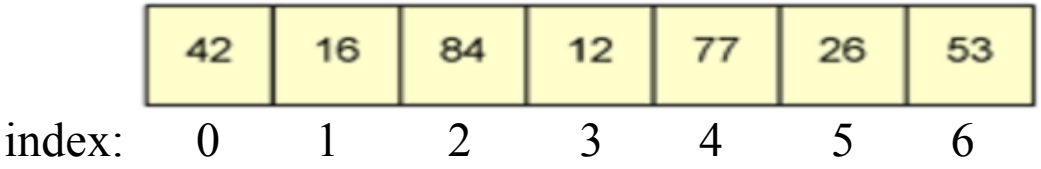
Two subtasks for a real C program

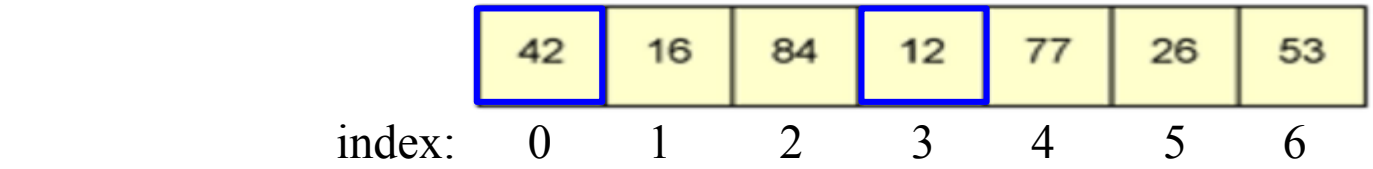
- Find the smallest integer in *list*
- Interchange *list*[*i*] and *list*[*min*]

가장작은걸 찾고 바꿈

Find the smallest integer in array *list*

– ...





Interchanging *list*[*i*] and *list*[*min*]

```
void swap(int *x, int *y)
{/* both parameters are pointers to ints */
    int temp = *x;    /* declares temp as an int and assigns
                         to it the contents of what x points to */
    *x = *y; /* stores what y points to into the location
                where x points */
    *y = temp; /* places the contents of temp in location
                  pointed to by y */
}
```

Program 1.3: Swap function

Function call to swap two values? ...

```
...
#define MAX_SIZE 101
#define SWAP(x, y, t) ((t) = (x), (x) = (y), (y) = (t))
void sort(int [], int);          /* selection sort */
void main(void)
{
     int i, n;
     int list[MAX_SIZE];
     printf("Enter the number of numbers to generate: ");
     scanf("%d", &n);
     if (n<1 || n> MAX_SIZE) {
          fprintf(stderr, "Improper value of n\n");
          exit(EXIT_FAILURE);
     }
     for (i=0; i<n; i++) {
          list[i] = rand()%1000;
          printf("%d", list[i]);
     }
    sort(list, n);
    printf("\n Sorted array:\n");
    for(i=0; i<n; i++)
          printf("%d", list[i]);
    printf("\n");
}
```

Program 1.4: Selection sort

```
#define SWAP(x, y, t) ((t) = (x), (x) = (y), (y) = (t))
```

```
void sort(int list[], int n)
{
    int i, j, min, temp;
    for(i = 0; i < n-1;  i++) {
            min = i;
            for (j = i + 1; j < n; j++)
                    if(list[j] < list[min])
                            min = j;
            SWAP(list[i], list[min], temp);
    }
}
```

min의 index를 바꿔줌

→ macro
⇒ 이 자리에 코드가 replace됨
⇒ 2 후에 컴파일

macro vs function
코드관점 / 함수호출
⇒ 고속X / ⇒ 고속O
⇒ 메모리 사용X / ⇒ 메모리 사용O

42	16	84	12	77	26	53
0	1	2	3	4	5	6

index:

12

```
#define SWAP(x,y,t) ((t) = (x), (x) = (y), (y) = (t))
```

매크로 쓰면 자나 float 등 상관없이 쓸 수있음
but, debug 측면에서는 좀 어려움

```
void swap(int *x, int *y)
{/* both parameters are pointers to ints */
    int temp = *x;    /* declares temp as an int and assigns
                         to it the contents of what x points to */
    *x = *y; /* stores what y points to into the location
                where x points */
    *y = temp; /* places the contents of temp in location
                  pointed to by y */
}
```

Program 1.3: Swap function

```
#define SWAP(x,y,t) ((t) = (x), (x) = (y), (y) = (t))
```

#define SQUARE(x) (x*x)

vs

#define SQUARE(x) ((x) * (x))

괄호를 꼭 붙여주자!

탐색

Ex 1.2 Binary Search

Assume that we have ***n*** distinct **integers** that are already **sorted** and stored in the array *list*.

정렬되어 있는 경우에 사용가능

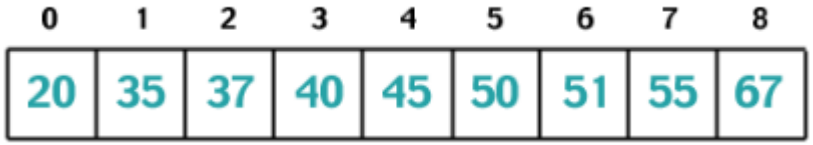


Figure out if an integer *searchnum* (ex: 37) is in *list*

If it is in *list*

→ return its index, ***i***

If it is not present

→ return **-1**

Ex 1.2

A solution : [Binary Search]

– Compare *list*[middle] with *searchnum*

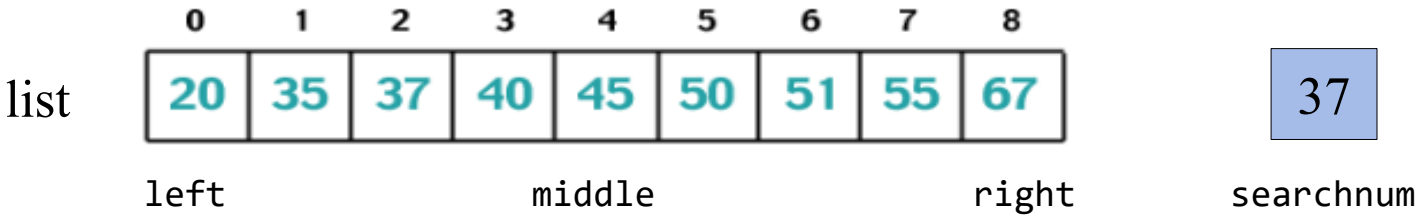
1) searchnum < list[middle] :

right = middle-1

2) searchnum == list[middle] : Find → return middle

3) searchnum > list[middle] :

left = middle+1



```
while (there are more integers to check) {
        middle = (left + right) / 2;
        if(searchnum < list[middle])
                right = middle-1;
        else if (searchnum = = list[middle])
                return middle;
        else
                left = middle +1;
}
```

Program 1.5: Searching a sorted list

Algorithm contains two **subtasks**:

(1) determining if there are any integers left to check

(2) comparing *searchnum* to list[*middle*]

```
int compare(int x, int y)
{/* compare x and y, return -1 for less than, 0 for equal,
    1 for greater */
   if (x < y) return -1;
   else if (x == y) return 0;
        else return 1;
}
```

Program 1.6: Comparison of two integers

The macro version is:

```
#define COMPARE(x,y) (((x) < (y)) ? -1: ((x) == (y)) ? 0: 1)
```

x<y 가 참이면 -1 반환

거짓이면 ((x)==(y)) ? 0 : 1 을 반환

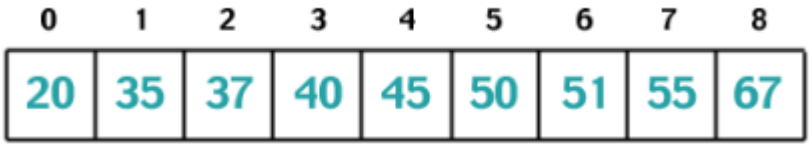
⇒ 0 or 1을 반환

```
#define COMPARE(x, y) ( ((x)<(y)) ? -1 : ((x)= = (y) ? 0 : 1) )

int binsearch(int list[], int searchnum, int left, int right)
{
    int middle;
    while (left <= right) {
        middle = (left + right) / 2;
        switch (COMPARE(list[middle], searchnum)) {
            case -1: left = middle + 1
                     break;
            case 0 : return middle;
            case 1 : right = middle -1;
        }
    }
    return -1;
}
```

Program 1.7: Searching an ordered list

Ex) Search for the value 37




```
int binsearch(int list[], int searchnum, int left, int right)
{
    int middle;
    while (left <=right) {
        middle = (left + right) / 2;
        switch (COMPARE(list[middle], searchnum)) {
            case -1: left = middle + 1
                     break;
            case 0 : return middle;
            case 1 : right = middle -1;
        }
    }
    return -1;
}
```

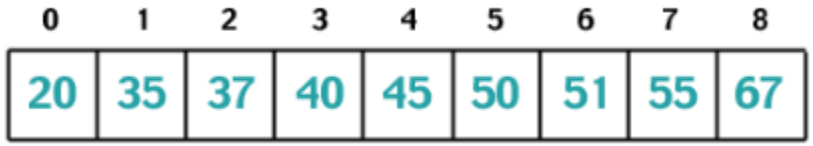
45 > 37
35 < 37

left = 0 0 2

right = 8 3 3

middle = 4 1 2

Ex) Search for the value 41




```
int binsearch(int list[], int searchnum, int left, int right)
{
    int middle;
    while (left <=right) {
        middle = (left + right) / 2;
        switch (COMPARE(list[middle], searchnum)) {
            case -1: left = middle + 1
                     break;
            case 0 : return middle;
            case 1 : right = middle -1;
        }
    }
    return -1;
}
```

45>41
35<41
37<41
40<41

left < right 아니면 return -1

left = 0 0 2 3 4

right = 8 3 3 3

middle = 4 1 2 3

1.3.2 Recursive Algorithms

```
int binsearch(int list[], int searchnum, int left, int right)
{
    int middle;
    if (left <=right) {
        middle = (left + right) / 2;
        switch (COMPARE(list[middle], searchnum)) {
            case -1: return
                     binsearch(list, searchnum, middle+1, right);
            case 0 : return middle;
            case 1 : return
                     binsearch(list, searchnum, left, middle-1);
        }
    }
    return -1;
}
```

Program 1.8: Recursive implementation of binary search

```c
int factorial(int n) {
    if (n == 0) {
        return 1;
    }
    else {
        return n * factorial(n - 1);
    }
}

int main() {
    int result = factorial(10000);
    printf("Result: %d\n", result);
    return 0;
}
```

탈출조건

Error code -1073741571 (also known as 0xC00000FD) occurs in Visual Studio Code (VSC) when a program crashes due to a stack overflow or stack exhaustion.

자료구조응용

01. Basic Concepts (10점)

2023.3.7(수)

++ lms 제출내용 : (1) 소스코드 5개 (2) 문서파일 1개 (이름_학번_01.pdf/docx)
* 문서파일: 실행결과 캡쳐
* 소스코드: 압축하지 않고 upload

1. n개의 정수 데이터를 입력받고 입력된 데이터에서 min값을 찾아서 출력하는 코드를 작성하라. (array 사용하지 않음)
   - min 값을 찾기는 함수로 작성

입력 (n, n개의 데이터)	출력
5 1 -1 0 3 100	-1

2. 1번 문제에서 입력 부분을 파일 입력으로 처리한다.

입력 (파일이름)	파일 내용 (data1.txt)	출력
data1.txt	5 1 -1 0 3 100	-1

3. n개의 데이터를 입력받아 array에 저장하고 저장된 데이터를 오름차순으로 selection sorting을 수행하고 그 결과를 출력하라.

입력 (파일이름)	파일 내용 (data2.txt)	출력
data2.txt	5 1 -1 0 3 100	-1 0 1 3 100

4. Binary search을 수행하고 수행결과를 출력하는 코드를 작성하라.
   - 검색값이 존재하는 경우 S (인덱스), 존재하지 않는 경우 F (-1) 출력

입력 (파일이름, 검색값)	파일 내용 (data2.txt)	출력
data2.txt 3	5 1 -1 0 3 100	S (3)
data2.txt 4	5 1 -1 0 3 100	F (-1)

5. 피보나치 수 F0 = 0, F1 = 1, Fi = Fi_1+Fi_2 (i>1)로 정의된다. 재귀함수를 이용해서 입력값 n에 대한 Fn를 계산후 출력하는 코드를 작성하라.
   - 계산할 수 잇는 n의 최대값은 얼마인지 comment으로 작성

입력 (n)	출력
40	102334155

BASIC CONCEPT

1.2 Pointers and Dynamic Memory Allocation

1.3 Algorithm Specification

1.4 Data Abstraction

1.5 Performance Analysis

1.6 Performance Measurement

Basic data types in C: char, int, float, double

- Modifier keywords: short, long, unsigned

For grouping data: **array, structure**

```
int list[5];
struct student {
        char lastName;
        int studentId;
        char grade;
};
```

Pointer data types : …

All programming languages

- Predefined data types + User-defined data types

What is a data type? …

A data type is a collection of objects and a set of operations that act on those objects.

Ex) data type `int` in C

- Objects
  - { 0, +1, -1, …, INT_MAX, INT_MIN}
  - Representation : 2 byte or 4 bytes of memory
- Operations
  - Arithmetic operators { +, -, *, /, % } 산술연산자
  - Testing for equality/inequality 비교연산
  - Operation that assigns an integer to a variable

ADT에 대해 서술하시오

Abstract Data Type (ADT):

- Ex) C++, Java
- The specification of the objects and the specification of the operations on the objects ***is separated from*** the representation of the objects and the implementation of the operations.
- ADT is ***implementation-independent***

구현 독립적

Typically, an **ADT definition** will include at least one function from each of these three categories:

- Creator/constructor : … 생성자
- Transformers : … 값을 바꾸는 것들
- Observers/reporters : …

ADT는 구체적인 구현의 완성과정에 독립적으로, 손쉽게 그 기능이 무엇인지를 나열한 것을 말한다

또한 object와 operation으로 이루어져있으며, Constructor, transformers, observers 와 같은 3가지 categories를 적어도 1개의 function을 가진다

구현은 없고 가능만 나열

ADT *NaturalNumber* is

objects: an ordered subrange of the integers starting at zero and ending at the maximum integer (*INT_MAX*) on the computer

functions:

for all $x, y \in$ *NaturalNumber*; *TRUE*, *FALSE* $\in$ *Boolean*
and where +, −, <, and == are the usual integer operations

NaturalNumber Zero( )	::=	0
Boolean IsZero(x)	::=	**if** (x) **return** *FALSE*
		else return *TRUE*
Boolean Equal(x, y)	::=	**if** (x == y) **return** *TRUE*
		else return *FALSE*
NaturalNumber Successor(x)	::=	**if** (x == *INT_MAX*) **return** x
		else return $x + 1$
NaturalNumber Add(x, y)	::=	**if** (($x + y$) <= *INT_MAX*) **return** $x + y$
		else return *INT_MAX*
NaturalNumber Subtract(x, y)	::=	**if** ($x < y$) **return** 0
		else return $x - y$

end *NaturalNumber*

ADT 1.1: Abstract data type *NaturalNumber*

::= is defined as

BASIC CONCEPT

1.2 Pointers and Dynamic Memory Allocation

1.3 Algorithm Specification

1.4 Data Abstraction

1.5 Performance Analysis

1.6 Performance Measurement