C – 알고리즘 기초

Algorithm

알고리즘 기초

- 알고리즘

  어떤 문제를 해결하기 위한 절차나 방법이다. 주어진 입력을 출력으로 만드는 과정을 구체적이고 명료하게 표현한 것이다.

  - 1부터 5까지의 합

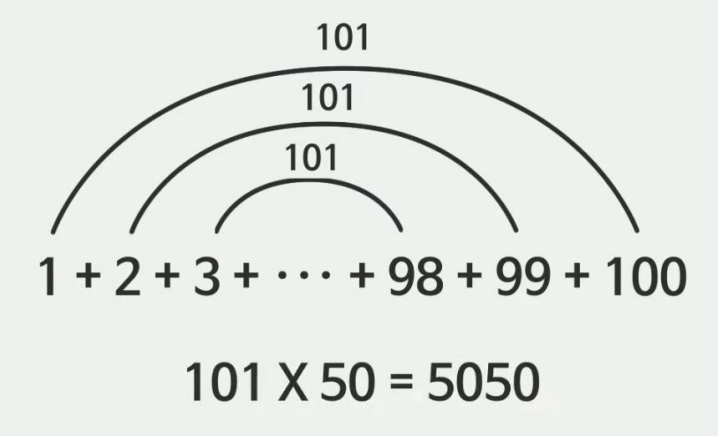
```c
//단순 합계
printf("%d\n", 1 + 2 + 3 + 4 + 5);

//for문 사용
int i, sum = 0;

for (i = 1; i <= 5; i++) {
    sum += i;
}
printf("%d\n", sum);
```

알고리즘 기초

- 1부터 n까지의 합 계산하기

1 + 2 + 3 + ··· + n ⟶ 방법 1

n x (n + 1) ÷ 2 ⟶ 방법 2



알고리즘 기초

- 1부터 n까지의 합 계산하기

```
int sumN(int n) {
    int i, sum = 0;

    for (i = 1; i <= n; i++) {
        sum += i;
    }
    return sum;
}
```

```
int sumN2(int n) {
    int sum = 0;
    sum = (n * (n + 1)) / 2;

    return sum;
}
```

알고리즘 기초

- 1부터 n까지의 합 계산하기

```
//합계 1
int result1;
result1 = sumN(10);

printf("합계: %d\n", result1);

//합계 2
int result2;
result2 = sumN2(10);

printf("합계: %d\n", result2);
```

알고리즘 기초

- 계산 복잡도

  - **입력 크기와 계산 횟수**
    - 첫 번째 알고리즘 : 덧셈 n번
    - 두 번째 알고리즘 : 덧셈, 곱셈, 나눗셈(총 3번)

  - **대문자 O표기법(Big O) : 계산 복잡도 표현**
    - O(n) : 필요한 계산횟수가 입력 크기 n과 비례할 때
    - O(1) : 필요한 계산횟수가 입력 크기 n과 무관할 때
  - **판단**
    - 두번째 방법이 계산 속도가 더 빠름

알고리즘 기초

- 배열에서 값 찾기 1

```c
int a[] = { 9, 8, 7, 6, 7 };
int i;
int count = 0;

//7이 몇 개인지 세기
for (i = 0; i < 5; i++){
    if (a[i] == 7) {
        printf("7 발견!\n");
        count++;
    }
}
printf("7을 %d개 발견!", count);
```

알고리즘 기초

- 배열에서 값 찾기 2

```
//7을 하나 발견하면 종료
int sw = 0;  //상태(토글) 변수
for (i = 0; i < 7; i++) {
    if (a[i] == 7) {
        printf("7 발견!\n");
        sw = 1;
        break;
    }
}

if(sw == 0)
    printf("7을 발견 못함!\n");
```

알고리즘 기초

- 평균 구하기

```c
//평균 구하기
int a[] = { 70, 80, 65, 90 };
int sum = 0;
int i;

//단순 평균
printf("%d\n", (a[0] + a[1] + a[2] + a[3]) / 4);

//for문 사용
for (i = 0; i < 4; i++) {
    sum += a[i];
}
printf("평균은 %.1f입니다.\n", sum / 4.0);
```

알고리즘 기초

- 막대 그래프 그리기

```c
//int arr[] = { 3, 6, 4, 2 };
int arr[4];
int i, j;

//사용자 입력
for (i = 0; i < 4; i++) {
    printf("arr[%d] 입력: ", i);
    scanf("%d", &arr[i]);
}
printf("\n");

//막대 그래프 출력
for (i = 0; i < 4; i++) {
    printf("arr[%d]=%d|", i, arr[i]);
    for (j = 1; j <= arr[i]; j++) {
        printf("*");
    }
    printf("\n");
}
```

```
arr[0] 입력: 3
arr[1] 입력: 6
arr[2] 입력: 4
arr[3] 입력: 2

arr[0]=3|***
arr[1]=6|******
arr[2]=4|****
arr[3]=2|**
```

알고리즘 기초

- 막대 그래프 그리기

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>  //malloc(), free()
#include <stdbool.h> //true/false 사용
```

```c
//동적 메모리 할당
int* arr = NULL; //포인터 선언
int size;  //배열의 크기
int i, j;

printf("배열의 크기 입력: ");
scanf("%d", &size);

arr = (int*)malloc(sizeof(int) * size);
if (arr == NULL) {
    puts("메모리 할당에 실패했습니다.\n");
    return 1;
}
```

알고리즘 기초

- 막대 그래프 그리기

```
// 사용자 값 입력 및 유효성 검사
for (i = 0; i < size; i++) {
    while (true) {
        printf("arr[%d] 값 입력 (0 이상 정수): ", i);
        if (scanf("%d", &arr[i]) != 1 || arr[i] < 0) {
            printf("잘못된 입력입니다. 다시 입력하세요.\n");

            // 입력 버퍼 비우기
            while (getchar() != '\n');
        }
        else { //scanf("%d", &arr[i]) == 1
            break; //값 입력후 while문 빠져나옴
        }
    }
}
```

알고리즘 기초

- 막대 그래프 그리기

```
/*
   i=0, arr[0]=3, i++ , ***
   i=1, arr[1]=6, i++ , ******
   i=2, arr[1]=a, 잘못된 입력입니다.
   i=2, arr[2]=4, i++ , ****
   i=3, arr[3]=2, i++ , **
   i=4, 반복 종료
*/

//막대 그래프 출력
for (i = 0; i < size; i++) {
    printf("arr[%d]=%d|", i, arr[i]);
    for (j = 1; j <= arr[i]; j++) {
        printf("*");
    }
    printf("\n");
}
free(arr); //메모리 반납
```

```
배열의 크기 입력: 4
arr[0] 값 입력 (0 이상 정수): 3
arr[1] 값 입력 (0 이상 정수): 6
arr[2] 값 입력 (0 이상 정수): f
잘못된 입력입니다. 다시 입력하세요.
arr[2] 값 입력 (0 이상 정수): 4
arr[3] 값 입력 (0 이상 정수): 2

arr[0]=3|***
arr[1]=6|******
arr[2]=4|****
arr[3]=2|**
```

알고리즘 기초

- 문자열 역순으로 읽기

```
char a1[] = "DOG";
char a2[10];  //충분한 크기 확보
int i;

//a1을 a2에 거꾸로 복사
for (i = 0; i < 4; i++) {
    a2[i] = a1[2 - i];
}
a2[3] = '\0'; //문자열 끝에 널문자 추가

printf("%s를 거꾸로 읽으면 %s\n", a1, a2);
```

```
DOG를 거꾸로 읽으면 GOD
```

알고리즘 기초

- 문자열 역순으로 읽기

```
int n;
char a[] = "DOG";
char b[10];

n = strlen(a); //3 - a의 개수

for (i = n-1; i >= 0; i--) {
    b[n-1-i] = a[i];
}
b[n] = '\0';

printf("%s를 거꾸로 읽으면 %s\n", a, b);
```

알고리즘 기초

- 두 문자열 연결하기

```
/*
  - 두 개의 문자열을 연결하여 하나의 문자열로 만들기
    a 배열은 충분히 커야 합니다.
    "smart"(5자) + "phone"(5자) + '\0' = 총 11바이트가 필요
*/
```

```
int i = 0, j = 0;
char a[12] = "smart";
char b[] = "phone";

printf("%s+%s=", a, b);
while (a[i] != '\0')
    i++;
//printf("\ni=%d\n", i); //5
```

알고리즘 기초

- 두 문자열 연결하기

```
while (b[j] != '\0') {
    a[i] = b[j];
    i++;
    j++;
}
a[i] = '\0';
printf("%s\n", a);

/*
   i=5, a[5]=b[0], smartp
   i=6, a[6]=b[1], smartph
   i=7, a[7]=b[2], smartpho
   i=8, a[8]=b[3], smartphon
   i=9, a[9]=b[4], smartphone
   i=10, a[10] = '\0'
*/
```

```
smart+phone=smartphone
smartphone
```

알고리즘 기초

- 두 문자열 연결하기 – strcat() 사용

```
/*
  strcart() 함수
  문자열 끝('\0')을 찾아 자동으로 뒤에 붙여주며,
  복사 후 마지막에 '\0'도 추가합니다.
*/
char str1[20] = "smart";
char str2[] = "phone";

strcat(str1, str2);
printf("%s\n", str1);
```

재귀 알고리즘

- 재귀 호출

  어떤 함수 안에서 자기 자신을 부르는 것을 말한다.

  재귀 호출은 무한 반복하므로 종료 조건이 필요하다.

```
func(입력값):
    if 입력값이 충분히 작으면:  //종료 조건
        return 결과값
    else
        func(더 작은 입력값)  //자기 자신 호출
```

재귀 알고리즘

- SOS 구현

```
void func(int n)
{
    //방법 2
    if (n <= 0) { //종료 조건
        return;
    }
    else {
        printf("Help Me!\n");
        func(n - 1);
    }

    //방법 1
    /*printf("Help Me!\n");
    n--;
    if (n <= 0)
        return; //종료 조건
    else
        func(n);
    */
```

재귀 알고리즘

- SOS 구현

```
int main()
{
    int count = 4;

    func(count);

    return 0;
}
```

재귀 알고리즘

- 팩토리얼 계산하기 – 일반적인 계산

```c
#include <stdio.h>
/*
    - 1부터 5까지 곱하기
      1x2x3x4x5 -> 5!
*/
int factorial(int n) {
    int i, facto = 1;
    for (i = 1; i <= n; i++) {
        facto *= i;
    }
    return facto;
}
```

재귀 알고리즘

- 팩토리얼 계산하기 – 일반적인 계산

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

    a = factorial(1);  // 1 * factorial(0), 1 * 1 = 1
    b = factorial(2);  // 2 * factorial(1), 2 * 1 = 2
    c = factorial(3);  // 3 * factorial(2), 3 * 2 = 6

    printf("1!=%d, 2!=%d, 3!=%d\n", a, b, c);

    return 0;
}
```

재귀 알고리즘

- 팩토리얼 계산하기 – 재귀 알고리즘

```
/*
    4! = 4 x 3 x 2 x 1
    4! = 4 x 3!
    3! = 3 x 2 x 1
    3! = 3 x 2!

*/

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

재귀 알고리즘

- 팩토리얼 계산하기

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

    a = factorial(1);  // 1 * factorial(0), 1 * 1 = 1
    b = factorial(2);  // 2 * factorial(1), 2 * 1 = 2
    c = factorial(3);  // 3 * factorial(2), 3 * 2 = 6

    printf("1!=%d, 2!=%d, 3!=%d\n", a, b, c);

    return 0;
}
```

재귀 알고리즘

- 십진수를 이진수로 변환하기

```
int printBin(int a){
    if (a == 0 || a == 1)
        printf("%d", a);
    else{
        printBin(a/2);
        printf("%d", a%2);
    }
}
```

```
/*
          a 값                          몫  나머지
    printBin(11), printBin(11/2),  5    1
    printBin(5), printBin(5/2),    2    1
    printBin(2), printBin(2/2),    1    0
    printBin(1), printBin(1/2),    0    1
    //아래서 위로 기록 - 1011
    //가중치 방식
     11 -> 8 4 2 1
           1 0 1 1
*/
```

재귀 알고리즘

- 십진수를 이진수로 변환하기

```
int main()
{
    int x = 11;
    printBin(x);  //1011

    return 0;
}
```

정렬 알고리즘

- 단순 정렬(Simple Sort)

```c
//단순 정렬 - 내림차순
int a[5] = { 3, 2, 5, 1, 4 };
int i, j, temp;

for (i = 0; i < 4; i++) {
    for (j = i + 1; j < 5; j++) {
        if (a[i] < a[j]) { //'>'면 오름차순
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
}

for (i = 0; i < 5; i++) {
    printf("%-2d", a[i]);
}
```

```c
/*
  i=0, j=1, 3<2, 교환없음
       j=2, 3<5, 5,2,3,1,4
       j=3, 3<1, 교환없음
       j=4, 3<4, 5,2,4,1,3
  i=1, j=2, 2<4, 5,4,2,1,3
       j=3, 2<1, 교환없음
       j=4, 2<3, 5,4,3,1,2
  i=2, j=3, 3<1  교환없음
       j=4, 3<2  교환없음
  i=3, j=4, 1<2  5,4,3,2,1
*/
```

정렬 알고리즘

- 버블 정렬(Bubble Sort)

```c
//버블 정렬 - 오름차순
int a[5] = { 3, 2, 5, 1, 4 };
int i, j, temp;

for (i = 0; i < 5; i++) {
    for (j = 0; j < 4; j++) {
        if (a[j] > a[j+1]) { //'<'면 내림차순
            temp = a[j];
            a[j] = a[j+1];
            a[j+1] = temp;
        }
    }
}

for (i = 0; i < 5; i++) {
    printf("%-2d", a[i]);
}
```

```c
/*
  i=0, j=0, 3>2, 2,3,5,1,4
       j=1, 3>5, 교환없음
       j=2, 5>1, 2,3,1,5,4
       j=3, 5>4, 2,3,1,4,5
  i=1, j=0, 2>3, 교환없음
       j=1, 3>1, 2,1,3,4,5
       j=2, 1>4, 교환없음
       j=3, 4>5, 교환없음
  i=2, j=0, 2>1, 1,2,3,4,5 //최종 저장
  i=3, 교환없음
  i=4, 교환없음
  i=5, 반복종료
*/
```

검색 알고리즘

- 순차 검색(Sequential Search)

  처음부터 끝까지 순서대로 찾음

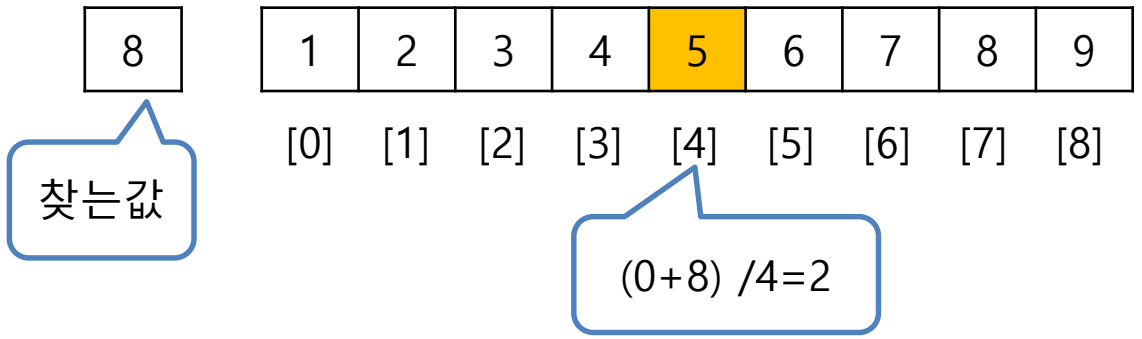
```c
int a[9] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
int i; //인덱스
int x = 8;  //찾을 값
int found = 0;  //상태 변수(찾음, 못찾음)

printf("===== 순차 검색 =====\n");
for (i = 0; i < 9; i++) {
    if (a[i] == x) {
        printf("%d은 a[%d]에 있습니다.\n", x, i);
        found = 1;  //찾음
        break;
    }
}
if (!found) { //못찾음
    printf("%d는 없습니다.\n", x);
}
```

검색 알고리즘

- 이분 검색(Binary Search)

  정렬된 데이터를 좌우 둘로 나눠서 찾는 값의 검색 범위를 좁혀가는 방법이다.

  - 찾을 값 < 가운데 요소 -> 오른쪽 반을 검색 범위에서 제외시킴
  - 찾을 값 > 가운데 요소 -> 왼쪽 반을 검색 범위에서 제외시킴
  - 찾을값 = 가운데 요소 -> 검색을 완료함



검색 알고리즘

- 이분 검색(Binary Search)

```
printf("===== 이분 검색 =====\n");
int low, high, mid;
int x;  //검색할 값
int found; //상태 플래그

int a[9] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

low = 0;  //첫 인덱스
high = 8; //마지막 인덱스
x = 8;
found = 0;

while (low <= high) {
    mid = (low + high) / 2; //중간값
    if (a[mid] == x) {
        found = 1;
        break;
    }
```

검색 알고리즘

- 이분 검색(Binary Search)

```
        else if (a[mid] < x) {
            low = mid + 1;
        }
        else {
            high = mid - 1;
        }
        /*
            a[4] < 8, low=5, mid=6
            a[6] < 8, low=7, mid=7
            a[7] = 8  찾음
        */
    }
    if (a[mid] == x) {
        printf("%d은 a[%d]에 있습니다.\n", x, mid);
    }
    else {
        printf("%d는 없습니다.\n", x);
    }
```