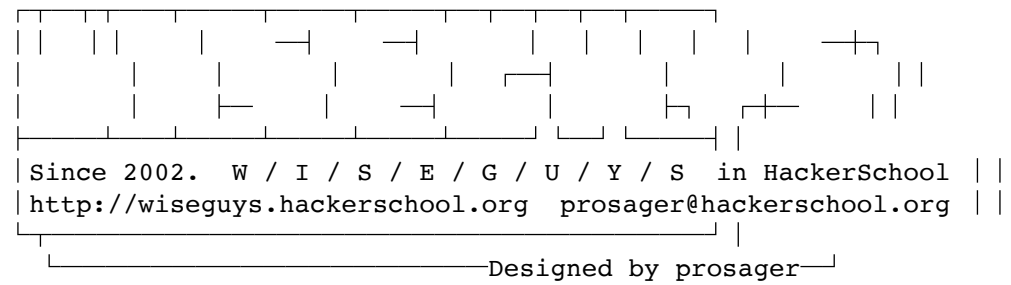


—[Document Infomation] ——————————————————————————

:: Title :: Data Structure (I)
:: Date :: 2004. 4. 6.
:: Author :: prosager
:: Contact:: E-Mail(pr0z4g3r@hotmail.com)
Homepage(http://prosager.hackerschool.org)

——

—[Notice]————————————————————————————————————

본 문서의 잘못된 부분이나 지적이나 추가하고 싶은 내용은
저자에게 메일을 보내기 바랍니다.

——

—[Index] ————————————————————————————————————

——

—[0x00 시작하면서..]————————————————————————————

언제나 이번달은 무엇을 적을까? 라는 고민에 빠지며, 다시한번 무언가를 열심히

나를 뛰게 만드는 계기가 되는 wgd 입니다. 최근에 상당히 바쁜 일상을 보내는 관계로

크게 기술적이거나, 혹은 최신기술, 혹은 삽-_-질이 크게 필요한 문서들은 제가 쓰지 못하고

있네요, 머리속에 있는 여러 가지를 정리하지 못해서, 이렇게 기초적인 자료 구조라는 주제로

이번달 wgd 를 작성하고자 합니다. 아마도 조금은 긴~ 연재가 될지도 모르고, 짧은 연재가

될지도 모르겠지만, 기본을 다지고, 저 역시 대강 넘어갔어던 부분을 활용하려고 합니다.

중간중간의 소스코드는 c 로 짜여진 것입니다. 다들 재밌게 보세요~

(너무 쉽다고 구박하지 말기!! ^^);

—[0x01. 자료 구조란 무엇인가?]————————————————————————————

자료 구조를 설명하기란 참으로 애매하다. 단순히 자료 구조. 뭐 어쩌란 말인가-_-;

하지만 우선, 차근히 하나씩 살펴보면, 무엇이든지 이해할 수 있는 법. 자료 구조를 보기전에

알고르즘을 확인하자.

알고리즘 : 주어진 문제를 효율적으로 해결하기 위하여 단계별로 나열한 명령들의 집합

이라고 나와있는데, 무슨 뜻인지 알겠는가? 나도 모르겠다-_-

상세히 서술된 문장을 살펴보자.

알고르즘(algorithm).

알고리즘은 어떤 작업을 수행하기 위하여 단계별로 작성된 유한개의 명령들의 집합으로서

다음과 같은 조건을 만족하여야 한다.

(1) 입력 조건 - 외부에서 입력이 제공될 수 있다.

(2) 출력 조건 - 적어도 하나 이상의 출력이 생성되어야 한다.

(3) 명확성 - 각 명령은 모호하지 않고 명확해야 한다.

(4) 유한성 - 어떠한 경우라도 알고리즘이 수행되면 유한개의 스텝 이내에 반드시 종료되어야 한다.

(5) 효율성 - 알고리즘은 실용적으로 그 문제를 해결할 수 있도록 시간과 공간적인 면에 있어서

효율적이어야 한다.

자, 위와 같은 결론이 있는데 이제 조금 이해가 가는가?

즉, 알고리즘이라는 것은 어떤 문제를 해결하는데 있어서, 최적이 방법을 내게 하여주는

일종의 함수라고 볼 수 있다.

자 위의 알고리즘의 개념을 이해하고, 이제 자료 구조로 들어가 보자.

알고리즘과 자료구조는 어떻게 보면 전혀 관계 없는것 처럼 보인다. 하지만, 정확하게 살펴보자.

예를 드는것이 사람들이 가장 쉽게 이해를 할 수 있을것이다.

prosager 가 hacking 과목에서 A+ 를 받았다.

이것은 A+ 을 받은것이 중요하지만, 누가? 어떤 과목에서? 가 더욱더 중요하다.

wantstar 가 받은 A+ 인지 누가 받은 A+ 인지를 알지 못하게 될 때가 있다.

이렇게 자료 자체의 의미도 중요하지만, 자료들 사이의 연계 관계 역시나 아주 중요하다.

이러한 자료 객체 원소 사이의 연관 관계를 "자료 구조" 라고 부른다.

설명하지 않은 부분은 추상자료형 같은 부분인데, 이것은 우리가 지금은 크게 다루지 않아도
되는 부분이기 때문에, 생략하고 다음으로 넘어가도록 하겠다.

—[0x02. 알고리즘과 차수 분석.]————————————————————————————

알고리즘과 차수 분석은 대부분의 사람들이 알것이라고 생각이 된다. 그룹 스터디 시간에도
몽이님이 시간을 측정하기 위해서 어떠한 알고리즘을 중간에 넣은후에 그 알고리즘이
완전히 실행되는데 걸리는 시간을 측정하여 본 것처럼 어떠한 연산 혹은 알고리즘이 결과를
내기까지의 과정에 걸리는 시간을 차수로서 분석하여, 알고리즘의 성능을 파악하는 방법이다.

여기에는 수행시간 비교법, 공간 복잡도, 차수 표기법 등과 같은 방법으로 많이 성능을
파악하는데, 여기서는 수행시간, 공간복잡도 보다는 차수를 중점적으로 설명할까 한다.
수행시간 비교법은 말 그대로 수행되는 시간을 비교하여서 어떠한 연산의 수행이 더 빠른가? 혹은
더 느린가에 대한 시간을 비교하는 방식이다. 이것은

```
for(i=0; i < 100000; i++) {
        for(j=0; j<100000; j++) {
                i +=k;
        }
}
```

위와 같은 연산의 걸리는 시간과 다른 간단한 for 문의 연산시 걸리는 시간을 비교하여 어떠한
연산의 시간이 얼마나 걸리느냐를 비교하는 방법이다. 이것은 해봐야한다는 단점이 있다.

다음은 공간 복잡도 인데, 이거을 가장 잘 설명하는 함수는 바로, 재귀함수 이다. 요즘은
재귀함수가 많이 쓰이지 않고 있지만, 내가 처음에 프로그램을 배울 시기에만 해도 재귀함수는
상당히 좋고, 꽤나 유용한 함수 였다고 기억이 된다. 하지만 왜 좋지 않은지는 이제 설명을 하겠다.
공간 복잡도라는 말은 함수라 호출 되었을 시에 스택에 할당되는 공간의 복잡도에 대해서 말한다.
스택에 대해서는 다음 장에서 자세히 다루기로 하고, 우선은 공간 복잡도에 대해서만 설명을
하기로 하겠다. 공간 복잡도의 가장 좋은 예는 재귀함수의 호출을 들수 있다.

```
int func(int i){
        return i * func(i-1);
}
```

위와 같은 방식으로 return 하게 되면, 계속 func 를 호출하게 되고, 스택에는 계속 -1 씩
되면서 func 를 호출하게 된다. 이 상황에서 큰 숫자의 재귀함수를 호출하게 된다면, 스택은
func(1) 을 만나기 전까지 계속 스택에 쌓게 될 것이다.
func(10000000000) 와 같이 아주 큰 수를 사용하게 되면, 스택에 쌓이는 양이 극도로 많아지고,

이러한 상황은 공간 복잡도를 아주 크게 사용하게 된다. 이러한 방식으로 메모리에 쌓이는 양에

대해서 알고리즘의 효율을 나타내는 방법이 공간 복잡도라는 방법이다.

이제 우리가 봐야 할 부분은 차수 표기법이라는 것이다. 이것으로 소스나 혹은 다른 방법보다는

조금더 가시적으로 알고리즘의 효율성을 나타낼수 있을것이다.

표기법에는 O 표기법, 오메가, 쎄타와 같은 방법이 있지만, 여기서는 간단하게 O 형만 다루어서

설명을 해 주겠다.

```
A(n) = 2,000n + 10,000
B(n) = 50n^2 + 200n + 1,000
C(n) = 3n^3
```

위에서 뒤의 숫자는 차수이다.

이렇게 나올때 O 표기법으로 알고리즘의 효율을 표현하게 되면,

```
A = O(n)
B = O(n^2)
C = O(n^3)
```

과 같이 표현된다. 여기서 궁금해 할 것이 왜 B 에서 n 이 포함되지 않는냐? 라고 물을수 있다.

그 이유는 n^2 에 비해서 n 은 아주 큰 숫자를 대입하면, 미미한 정도이기 때문에 모든 수학적인

수식에서 가장 차수가 큰 하나의 표현에 대해서 차수를 표기하는 방식이다.

이러한 방식으로 우리는 알고리즘의 효율에 대해서 알아 볼 수 있으며, 다른 방법도 분명히

존재할 것이다. 우선은 이렇게 간단한 방식에 대해서만 알아놓고, 다음으로 넘어가자.

─[0x03. 기본 자료 구조.(스택,큐)]──────────────────────────────────────

자, 이제 기본적인 우리가 잘 안다고 할 수 있는 스택과 큐에 대해서 알아보기로 하자.

스택, 우리에게 친숙할 수 밖에 없는 자료형이다. 리눅스에서의 bof, fsb, 등등 모든 해킹의

테크닉을 사용할때 스택을 예측하고, 파악하지 않고서, 어떻게 공격코드를 작성하고, 공격에 대한

삽질을 할 수 있겠는가? 우리는 이 부분에서 스택은 단순히 사용되는 방식이라고 알고만 있지

이것이 자료의 구조라고는 생각하지 않았을 것이다. 자 이제 스택에 대해서 알아보도록 하자.

다들 잘 알겠지만 스택은 LIFO(last input first out)이다. 즉 나중에 들어온 데이터가 가장 먼저

나가게 된다는 말이다. 잘 알듯이 pop, push 가 있다.

이 두가지에 대해서만 c 에서의 구현과 더불어서 설명을 하겠다.

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

/** 객체 표현 *****************************************************/
#define MaxStackSize 1000               /* 스택을 위한 배열의 최대 길이 설정 */
```

```
typedef struct stack {
        int top;
        ObjectStack elements[MaxStackSize];
} *Stack;


/** 함수 선언 ****************************************************/
Stack newStack(void);
…
ObjectStack popStack(Stack S);
/** 함수의 표현 ***************************************************/
Stack newStack()
{       Stack S = (Stack)malloc(sizeof(struct stack));
        S->top = -1;            /* 공백 스택 설정 */
        return S;
}

void freeStack(Stack S){ free(S); }

int emptyStack(Stack S){ return (S->top == -1); }

ObjectStack topStack(Stack S)
{       if (emptyStack(S)) {
                fprintf(stderr, "Error:: Stack Empty\n");
                exit(1);
        }
        return S->elements[S->top];
}

void pushStack(Stack S, ObjectStack item)
{       if (S->top >= MaxStackSize-1) {         /* 스택 오버플로어 검사 */
                fprintf(stderr, "Error:: Stack Overflow\n");
                exit(1);
        }
        S->elements[++S->top] = item;
}

ObjectStack popStack(Stack S)
{       ObjectStack item = topStack(S);
        S->top--;
        return item;
}
```

위의 소스는 단순하게 스택을 구현해 본것인데, 정확하게 말해서 안 돌아간다-_-;

단지 위와 같은 소스를 이용해서 스택을 구현이 가능하다는 말을 하려고 한 것이다.

스택은 뒤에서 배울 linked list 를 이용해서 구현이 가능한데, 이것은 지금 단계에서는

다루지 아니하고, 조금은 뒤로가서 다루도록 하겠다.

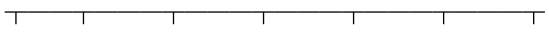
자~ 이제 조금은 익숙하고, 컴터를 접하면서 꼭 한번씩은 들어봤을 큐 라는 놈에 대해서 알아보겠다.

큐란 과연 무엇일까? 뭐에요?-_-aa

큐는...queue -_-;;

큐는 대기행렬이라고 불린다. FIFO(First input first out) 방식인데 이것은 먼저 들어오면, 먼저

나가버린다는 말이다. 생김새를 설명하면,



위와 같은 방식으로 앞에서 부터 뒤로 이렇게 된다는 말이다.
(그림 그리기 힘들다-_-;;)

저것은 단순한 큐이며, 원형큐가 있는데, 원형큐는 원형을 이루면서 돌아가면서, 데이터는 저장하고,

그 데이터는 다시 return 하기도 하는 방식으로 돌아가는 것이다.

큐에 대한 소스를 대강 보면,

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

/** 객체 표현 *******************************************************/
#define MaxQueueSize 1000               /* 큐를 위한 배열의 최대 길이 설정 */

typedef struct queue {
        int front, rear;
        ObjectQueue elements[MaxQueueSize];
} *Queue;


/** 함수 선언 *******************************************************/
Queue newQueue(void);
…
ObjectQueue deQueue(Queue Q);

/** 함수의 표현 *****************************************************/
Queue newQueue()
{       Queue Q = (Queue)malloc(sizeof(struct queue));
        Q->front = Q->rear = 0;         /* 공백 큐 설정 */
        return Q;
}

void freeQueue(Queue Q){ free(Q); }

int emptyQueue(Queue Q){ return (Q->front == Q->rear); }

ObjectQueue frontQueue(Queue Q)
{       if (emptyQueue(Q)) {
                fprintf(stderr, "Error:: Queue Empty\n");
                exit(1);
        }
        return Q->elements[(Q->front+1) % MaxQueueSize];
}

void enQueue(Queue Q, ObjectQueue item)
{       Q->rear = (Q->rear + 1) % MaxQueueSize;
        if (Q->front == Q->rear) {      /* 큐 오버플로어 검사 */
                fprintf(stderr, "Error:: Queue Overflow\n");
                exit(1);
        }
        Q->elements[Q->rear] = item;
}

ObjectQueue deQueue(Queue Q)
{       ObjectQueue item = frontQueue(Q);
        Q->front = (Q->front + 1) % MaxQueueSize;
        return item;
}
```

큐에 대한 구현은 다음과 같지만, 예전에 하도 짠거라-_- 틀릴지도 모릅니다-0-

그냥 봐주세요..ㅎㅎ

—[0x04. Linked list.]————————————————————————————————

  드디어 linked list 에 왔습니다. c 를 배울때 절대로 피해갈수 없는 관문!

그것이 이 linked list 인데요, 이것에 대해서는 간단하게 소스만 보여드리면서 설명하겠습니다.

우선 3가지의 연결 리스트에 대해서 살펴보겠습니다.

1, 단일 연결

2, 원형 연결

3, 이중 연결

여기서 단일 연결이라는 말은 다음에 오는 자료에 대한 pointer 만을 가지고 있습니다.

즉 next 만 있고, prev 가 없다는 말이구요, 원형 연결은 말 그래도 원형을 만들어서,

next 가 다음것을 가르키고 있지만, 제일 마지막의 자료형의 next 가 제일 처음의 위치를 가르키게

되면서, 원형을 유지하게 되는 것입니다. 이중 연결은 다들 아시듯이, prev, next 가 둘다 가지고 있는

저희가 잘 아는 연결 리스트이지요. 소스를 살짝~ 보겠습니다.

```
typedef struct dnode {
        int item;
        struct dnode *prev;     /* 이전 노드의 링크를 위한 멤버 */
        struct dnode *next;     /* 다음 노드의 링크를 위한 멤버 */
} Dnode;

void insertNode(Dnode *prev, Dnode *ptr)        /* prev의 뒤에 ptr을 삽입 */
{
        ptr->prev = prev;
        ptr->next = prev->next;
        prev->next->prev = ptr;
        prev->next = ptr;
}

void deleteNode(Dnode *ptr)     /* 노드 ptr을 삭제 */
{
        ptr->prev->next = ptr->next;
        ptr->next->prev = ptr->prev;
        free(ptr);
}
```

위의 소스는 단순하게 자료의 구조를 정의하고, 삽입 삭제하는 소스이다. 너무 간단한가?-_-

어쨌든 위와 같은 자료 구조가 있다는 것을 꼭! 기억해 두자!

—[0x05. 마치면서..]————————————————————————————————

  자~ 이것으로 지금까지 자료구조 1탄을 마치겠습니다. ^^

자료 구조에 대해서는 크게 알 필요는 없습니다. 하지만, 조금은 더 효율적이고, 멋진 프로그램을

짜기 위해서는 반드시 필요한 부분이 아닐까요? 해커 혹은 해킹을 하기 위해서는 기본적인

코딩능력이 필요합니다. 취약점은 알고, 어떻게 공격도 진행하면, exploit 이 되겠다. 라고 생각은

되지만, 그것을 컴이 알아 들을수 있는 언어로 구현하지 못한다는 것은 치명적이겠지요? ^^

모두들 조금씩 공부해서 나중에 크게 성장하는 Wiseguys 가 되기를..

나름대로 날림강의를 지금까지 읽어 주셔서 감사드립니다..^^
