

1. 설치

2018년 10월 20일 토요일 오후 9:35

c 자료구조 !

공부순서 :

1. C(하드웨어,컴파일러,운영체제)

2. 자료구조(모든곳에있는요소)

-> 자료구조종류 : list, stack, queue, tree, graph

3. C++

-> 게임(구로디지털단지)

, opencv(카메라,자율주행) --> 포인터를 알아야함! -> c언어를 공부해

, qt5(자동차계기판ui)

4. STL(공동환강사님)

1. 절차지향 => 2.객체지향(c++ 을 간략화-> java -> c#) => 3.일반화(STL)

www.visualstudio.com

커뮤니티 2017 실행

C++를 사용한 데스크톱 개발 -> 설치

비주얼 스튜디오를 쓰는 이유: -> 디버깅이 편함

우리는 .c를 짜면 컴파일,링킹은 편집기가 해줌

[.c] --comple--> [.o] --linking --> [.exe]

☐ ---> ☐virtual memory (window:4기가를 잡고 유저에게 2기

가를 줌,linux : 4기가잡고 3기가를 줌) [memory☐] <---cpu가 메모리를 읽음

↑ 이걸 천천히 보는걸 debugging이라함!

2. 컴파일, 링킹, 실행

2018년 10월 20일 토요일 오후 9:36

똑같은 소스코드를 각각 다른 컴파일러로 보내

c -> window에서는 [Visual Studio 2017]이 .obj .exe로 컴파일

c -> linux에서는 [gcc]가 .o .exe로 컴파일

tcp/ip(책 : 윤성우) -> system-programming(책:에이콘출판사 -linux api) 공부하기 !

java -> class파일을 통해 어떤 플랫폼에서 돌려도 다 돌아가지만 c는 컴파일 시켜야함

파일 - > 새 프로젝트

좌측 Visual C++ 선택, 기본값 : 빈프로젝트, 위치 : D:\WCW, 이름: Day1

우측 : 좌측 : 추가 : 새항목 --> C++선택, zoo.c 확장자는 반드시 .c로 끝나야해!

3. 2차원배열

2018년 10월 20일 토요일 오후 9:36

①int ②a = 7	①int ③a[②3]={3,4,5}
□□□<-70이들어감	□□□□<-345
a = 7	a[0] = 3, a[1]=4, a[2]=5

<<<< 2차원배열 읽는 순서 >>>>

①int ④a [③2][②3] 나 는 int형 3개짜리가 총 2개필요해

a:2차원배열명 // a[0],a[1]->2차원배열 안에 있는, 1차원배열명
int a [2][3] = {1,2,3,4,5,6};
a[0] : □□□ <- 1,2,3
a[1] : □□□ <- 4,5,6

a[0][1] = 1, a[0][2] = 2, a[0][3] = 3
a[1][1] = 4, a[1][2] = 5, a[1][3] = 6

& : 주소연산자

ex) &a 붙이면 a의 주소(위치값)가 나옴 -> 배열명은 선두주소다!

a[0]의 주소(번지)에 접근하는 것이 중요함

1. &a -> 전체를 감싼 주소

[□□□
□□□]

2. a -> 2차원배열 값 선두요소(int형3개짜리의 첫번째) -> 층을 옮김 (행)

[□□□]
□□□

3. &a[0] ->2번과 완전히 똑같은 코드, 1차원배열을 통째로 감싸버릴거야!

[□□□]
□□□

4. a[0] -> 호실을 옮기는 표현(열)

[□]□□
□□□

5. &a[0][0] ->4번과 완전히 똑같은 코드임!

[□]□□
□□□

4. 포인터

2018년 10월 20일 토요일 오후 9:36

모든 메모리에는 주소가 있고

메모리에 접근하려면

접근하려는 곳의 메모리의 주소를 갖고있어야함

ex)

```
int a = 7;
```

주소:100 : □<-7

----- [* : 간접연산자] -----
: 항상 뒤에 주소가 나옴

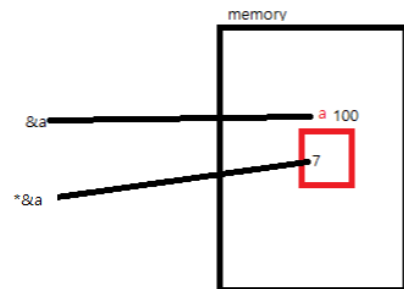
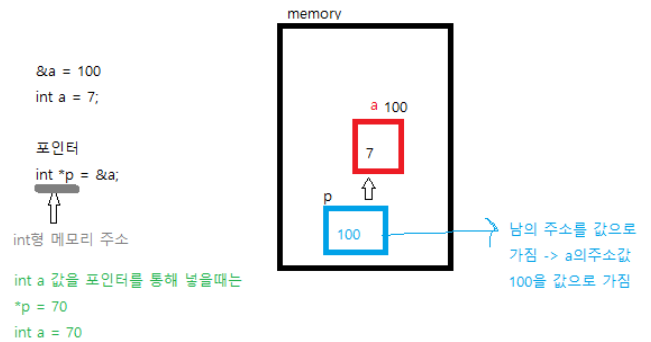
```
pf("%d", a);    -> direct Acc          => 7출력  
pf("%d", &a);   -> a네집주소 address   => 100출력  
pf("%d", *&a); -> a네집주소 100 안쪽의 값이 뭐냐! => 7출력  
간접 : indirect Operator
```

1. Day2

& 주소

* 간접

--> 그림



<call by value>

void func(int *a) //a와 b의 내용(값)을 바꾸는 함수

```
{  
    int *a = 70;  
}  
int main()  
{  
    int a = 7;  
    func(&a);    // == func() -> call by value: 값이 copy됐다.  
    printf("%d\\n", a);  
  
    return 0;  
}
```

<포인터>

```
int main(void)  
{  
    int a = 7;  
    int *p = &a; //a의 주소값을 가짐  
  
    printf("%d\\n", a); //a의 데이터 7  
    printf("%d\\n", &a); //a의 주소 54123  
    printf("%d\\n", p); //a의 주소 54123  
    printf("%d\\n", *p); //a의 데이터 7  
    printf("%d\\n", *&a); //a의 주소 7  
  
    getchar(); //콘솔 화면이 사라지는 것을 멈추게 함  
    return 0; //os에게 '프로그램이 잘 끝났다'라고 알림  
}
```

<a와 b값 바꾸기>

```

void func(int *a, int *b) //a와 b의 내용(값)을 바꾸는 함수
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}

int main()
{
    int a = 7;
    int b = 5;
    //func(&a);      // == func() -> call by value: 값이 copy됐다.
    func(&a, &b); //a와 b의 내용(값)을 바꾸는 함수
    printf("%d %d\n", a, b); //5 7

    return 0;
}

```

```

int main()
{
    int a = 7;
    int b = 5;
    int *pa = &a;
    int *pb = &b;
    // pa와 pb를 이용해 a,b를 바꿔라
    int t;
    t = *pa;
    *pa = *pb;
    *pb = t;

    printf("%d %d\n", a, b); //5 7

    return 0;
}

```

5. 싱글포인터

2018년 10월 20일 토요일 오후 9:37

`p = &a;`니까
`scanf("%d", &a);` -> 집어넣은 데이터를, `&a`자리에 `p`넣을수있음
`print("%d", a);` -> 화면에 출력 `a`자리에 `*p`넣을수있음

```
int main()
{
    int a[5] = { 1,2,3,4,5 };

    printf("%d\n", a[2]);
    a[2] = 33;
    printf("%d\n", a[2]);
}
```

>>>>>> 위와 아래 같음 <<<<<<<<

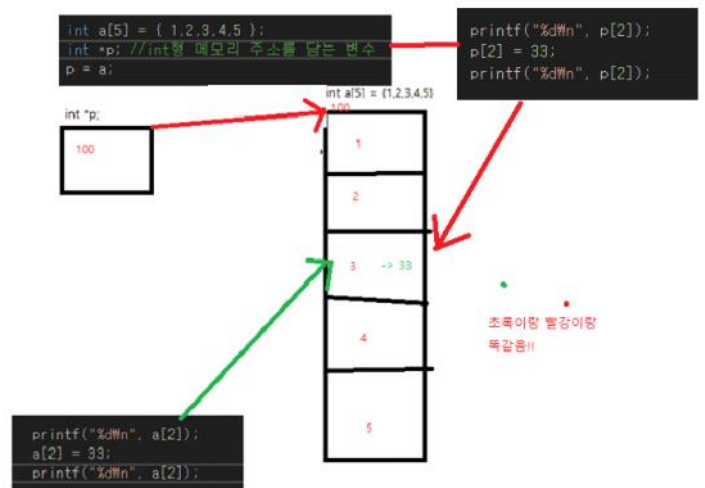
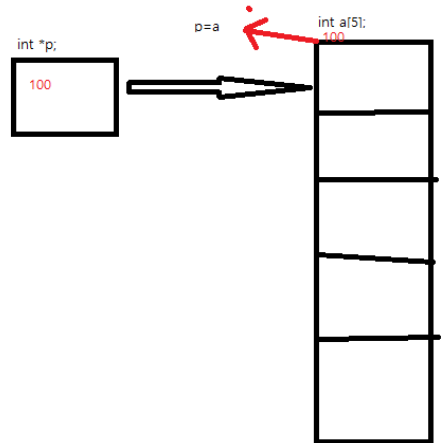
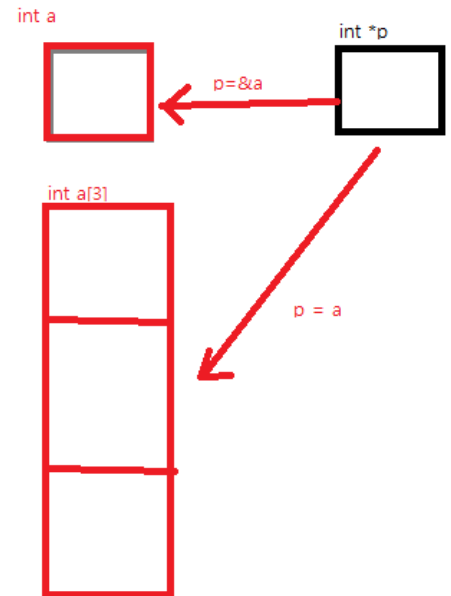
```
int main()
{
    int a[5] = { 1,2,3,4,5 };
    int *p; //int형 메모리 주소를 담는 변수
    p = a;  //&a안하는이유는 a(배열의이름)은 선두주소값이기때!

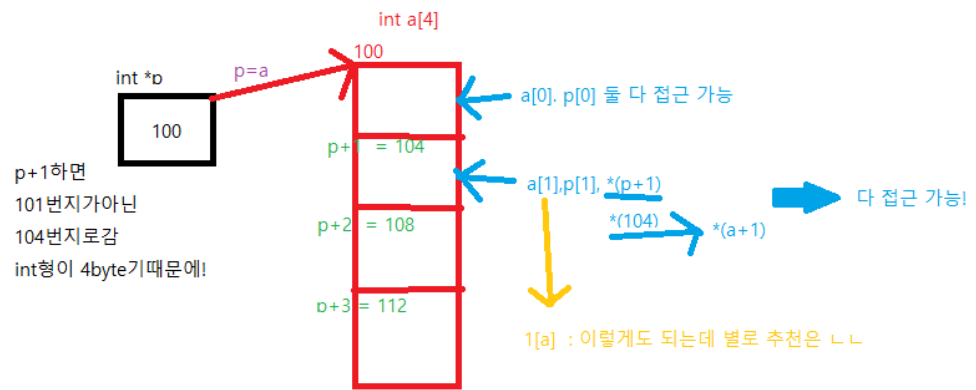
    printf("%d\n", p[2]);
    p[2] = 33;
    printf("%d\n", p[2]);
}
```

>>>>>> 위와 아래 같음 <<<<<<<<

```
func(int *p)
{
    printf("%d\n", p[2]);
    p[2] = 33;
    printf("%d\n", p[2]);
}

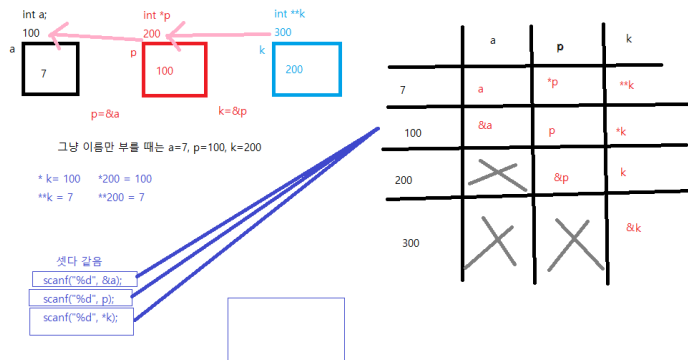
int main()
{
    int a[5] = { 1,2,3,4,5 };
    func(a);
} -> main 함수의 코드가 짧아짐
```





6. 더블포인터

2018년 10월 20일 토요일 오후 9:38

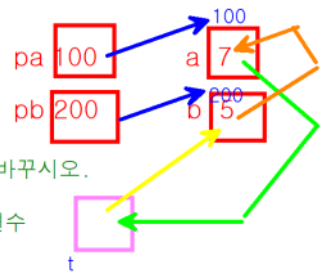


```
int main()
```

```
{
    int a = 7;
    int b = 5;
    int *pa = &a;
    int *pb = &b;
    //pa와 pb를 이용하여 a, b를 바꾸시오.

    int t; //임시변수, 도와주는 변수
    t = *pa;
    *pa = *pb;
    *pb = t;
}
```

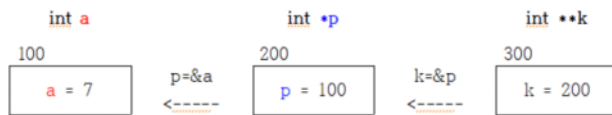
```
printf("%d %d\n", a, b);
```



Microsoft Visual Studio 디버그 콘솔

5 7

● 그림



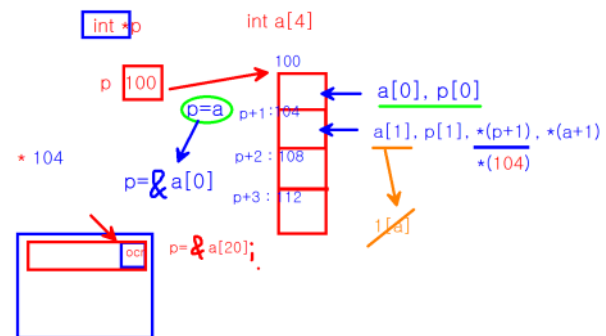
● 표

	a (사람)	p (맥배)	k (맥배사)
7	a	*p	**k
100	&a	p	*k
200		&p	k
300			&k

**k = **200

k = *100 = *&a

scanf = ("%d", &a); = scanf = ("%d", p); = scanf = ("%d", *k);



7. 포인터 배열

2018년 10월 20일 토요일 오후 9:40

1. 포인터 여러개 만들기

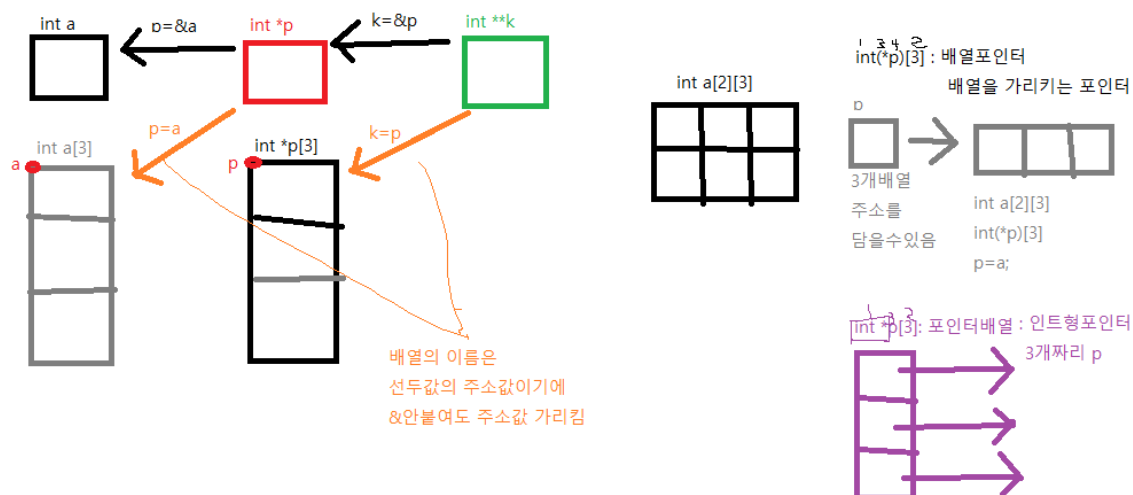
포인터 배열은 더블포인터가 볼

`int *p[3]`이런식 : `int`형 포인터가 3개있다

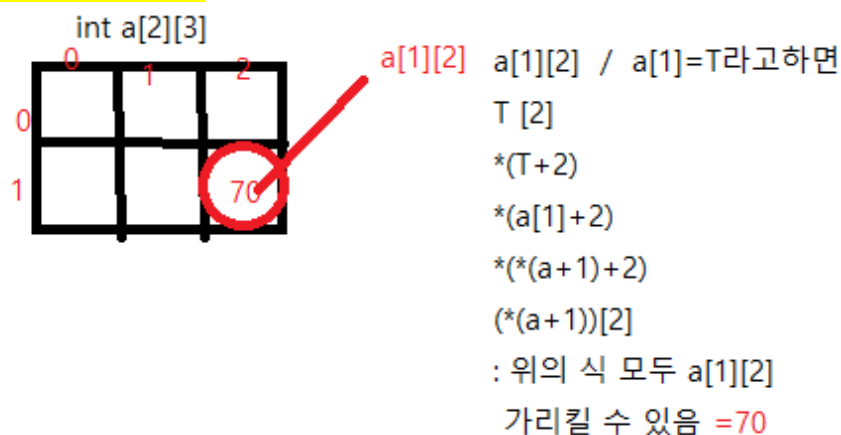
`int(*p)[3]` : 배열 포인터 인트형메모리가 3개있는데, 그 배열을 가리키는 포인터
- 인트형 3개짜리 포인터를 가리키는 포인터 `p`

`int *p[3]` : 포인터배열
인트형 포인터가 세개있구나! 그 배열 이름이 `p`구나

1-1 포인터 배열



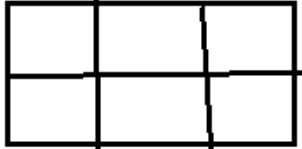
1-2 2차원배열 표현



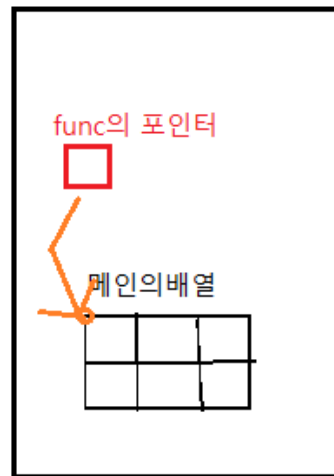
1-3 포인터사용이유

1-3 포인터사용이유

메인함수가 끝날때,
func()으로 빼면,
func의 배열포인터가
메인의 배열에 접근할수있음



메모리



8. 포인터 func으로 값 받기

2018년 10월 20일 토요일 오후 9:41

2. 포인터 func으로 값 받기 코드

```
void func_a(int a) {}
void func_b(int *pb) {}
void func_c(int (*pc)[3])//배열포인터 : 2차원배열의 선두주소는 배열포인터가 받아야함
{}
void func_d(int *d) {}
void func_e(int **e) {}
void func_f(int f) {}
int main()
{
    int *d;           //싱글포인터 //아직 값을 몰라 가리키는게 없으니까
    func_d(d);        //d:값 //&d 주소 //지금 d는 값이 없으니까 잘못된 코드임
}
```

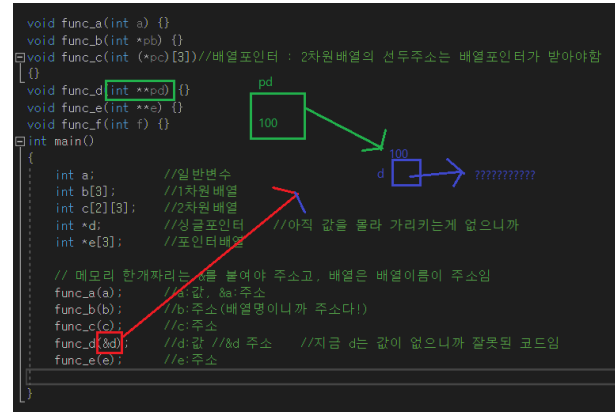
그래서 func_d(&d) 포인터 값을 보내야함

그러니까

func_d(int **pd) 이렇게 받아야함

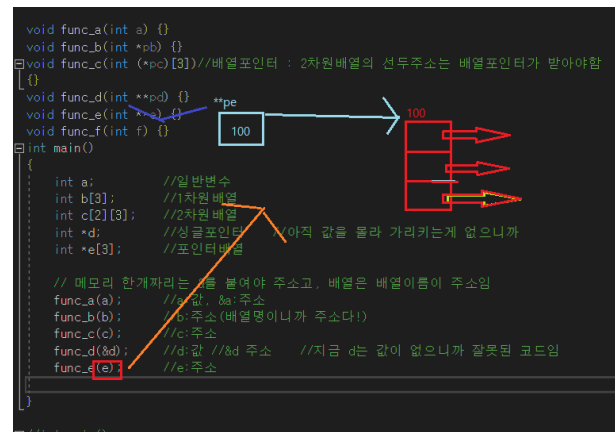
```
void func_a(int a) {}
void func_b(int *pb) {}
void func_c(int (*pc)[3])//배열포인터 : 2차원배열의 선두주소는 배열포인터가 받아야함
{}
void func_d(int **pd) {}
void func_e(int **pe) {}
int main()
{
    int a;           //일반변수
    int b[3];         //1차원배열
    int c[2][3];      //2차원배열
    int *d;           //싱글포인터 //아직 값을 몰라 가리키는게 없으니까
    int *e[3];        //포인터배열

    // 메모리 한개짜리는 &를 붙여야 주소고, 배열은 배열이름이 주소임
    func_a(a);         //a:값, &a:주소
    func_b(b);         //b:주소(배열명이나 주소다!)
    func_c(c);         //c:주소
    func_d(&d);        //d:값 //&d 주소 //지금 d는 값이 없으니까 잘못된 코드임
    func_e(e);         //e:주소
}
```



```
void func_a(int a) {}
void func_b(int *pb) {}
void func_c(int (*pc)[3])//배열포인터 : 2차원배열의 선두주소는 배열포인터가 받아야함
{}
void func_d(int **pd) {}
void func_e(int **e) {}
void func_f(int f) {}
int main()
{
    int a;           //일반변수
    int b[3];         //1차원배열
    int c[2][3];      //2차원배열
    int *d;           //싱글포인터 //아직 값을 몰라 가리키는게 없으니까
    int *e[3];        //포인터배열

    // 메모리 한개짜리는 &를 붙여야 주소고, 배열은 배열이름이 주소임
    func_a(a);         //a:값, &a:주소
    func_b(b);         //b:주소(배열명이나 주소다!)
    func_c(c);         //c:주소
    func_d(&d);        //d:값 //&d 주소 //지금 d는 값이 없으니까 잘못된 코드임
    func_e(e);         //e:주소
}
```



```
void func_a(int a) {}
void func_b(int *pb) {}
void func_c(int (*pc)[3])//배열포인터 : 2차원배열의 선두주소는 배열포인터가 받아야함
{}
void func_d(int **pd) {}
void func_e(int **pe) {}
void func_f(int f) {}
int main()
{
    int a;           //일반변수
    int b[3];         //1차원배열
    int c[2][3];      //2차원배열
    int *d;           //싱글포인터 //아직 값을 몰라 가리키는게 없으니까
    int *e[3];        //포인터배열

    // 메모리 한개짜리는 &를 붙여야 주소고, 배열은 배열이름이 주소임
    func_a(a);         //a:값, &a:주소
    func_b(b);         //b:주소(배열명이나 주소다!)
    func_c(c);         //c:주소
    func_d(&d);        //d:값 //&d 주소 //지금 d는 값이 없으니까 잘못된 코드임
    func_e(e);         //e:주소
}
```

9. 배열 복사

2018년 10월 20일 토요일 오후 9:41

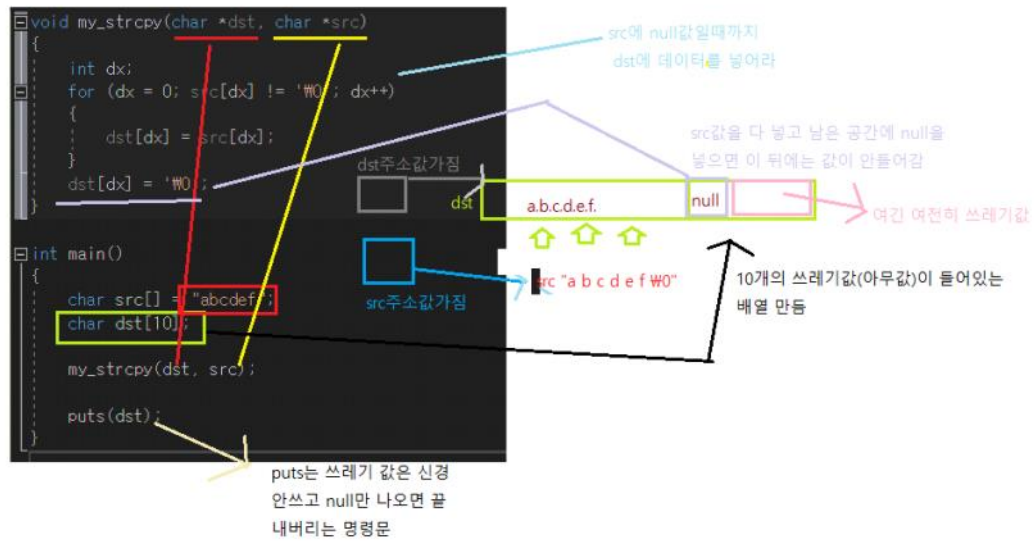
3. 배열 복사

```
void my_strcpy(char *dst, char *src)
{
    int dx;
    for (dx = 0; src[dx] != '\0'; dx++)
    {
        dst[dx] = src[dx];
    }
    dst[dx] = '\0';
}

int main()
{
    char src[] = "abcdef";
    char dst[10];

    my_strcpy(dst, src);

    puts(dst);
}
```



10. 2차원 배열을 2차원,1차원처리

2018년 10월 20일 토요일 오후 9:42

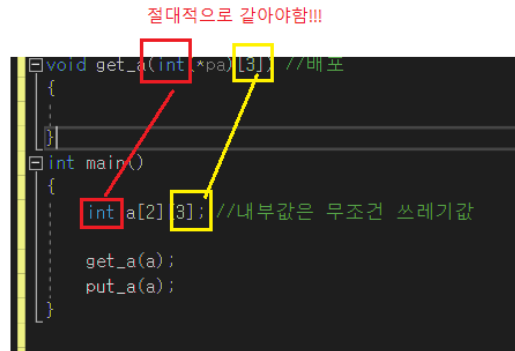
4. 2차원 배열을 2차원 답게 처리

```
void get_a(int(*pa)[3]) //배포
{
    int rx, cx;
    for (rx = 0; rx < 2; rx++)
        for (cx = 0; cx < 3; cx++)
            scanf("%d", &pa[rx][cx]);
}

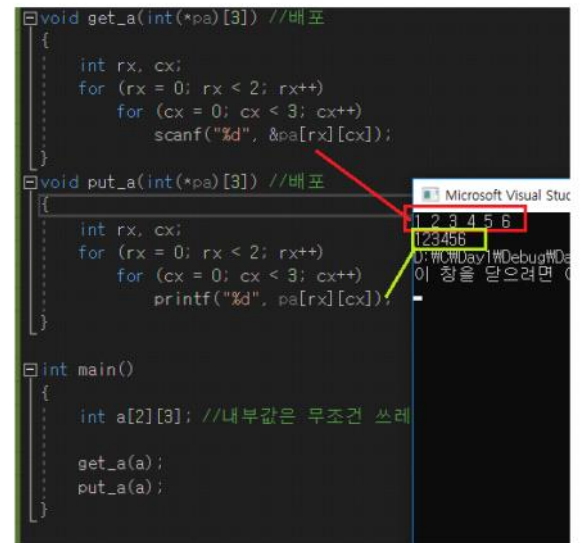
void put_a(int(*pa)[3]) //배포
{
    int rx, cx;
    for (rx = 0; rx < 2; rx++)
        for (cx = 0; cx < 3; cx++)
            printf("%d", pa[rx][cx]);
}

int main()
{
    int a[2][3]; //내부값은 무조건 쓰레기값

    get_a(a);
    put_a(a);
}
```



4-2 2차원배열 2차원으로 처리



-----1차원으로 처리-----

```
void get_a(int(*pa)[3]) //배포 : 열의 크기 절대적으로 맞추기
{
    int rx, cx;
    for (rx = 0; rx < 2; rx++)
        for (cx = 0; cx < 3; cx++)
            scanf("%d", &pa[rx][cx]);
}

void put_a(int(*pa)[3]) //배포
{
    int rx, cx;
    for (rx = 0; rx < 2; rx++)
        for (cx = 0; cx < 3; cx++)
            printf("%d", pa[rx][cx]);
}

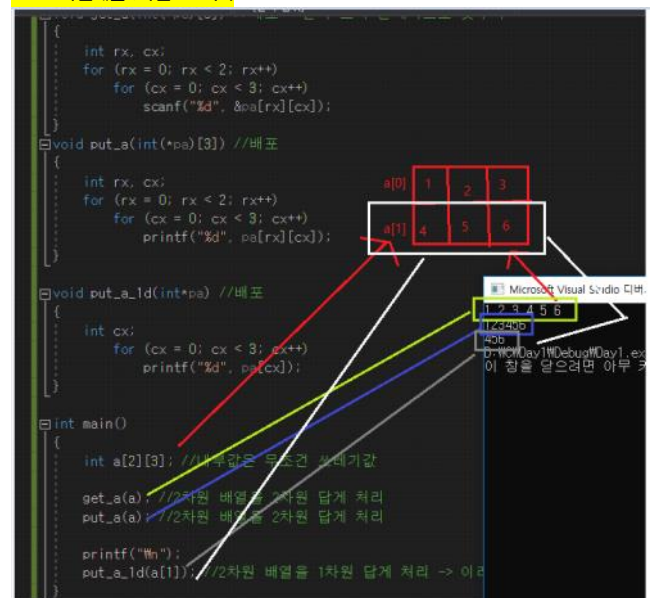
void put_a_1d(int*pa) //배포
{
    int cx;
    for (cx = 0; cx < 3; cx++)
        printf("%d", pa[cx]);
}

int main()
{
    int a[2][3]; //내부값은 무조건 쓰레기값

    get_a(a); //2차원 배열을 2차원 답게 처리
    put_a(a); //2차원 배열을 2차원 답게 처리

    printf("\n");
    put_a_1d(a[1]); //2차원 배열을 1차원 답게 처리 -> 이러면 한개짜리 메모리 날
    라갈 2번째줄의 선두데이터 주소
}
```

4-3 2차원배열 1차원으로처리



11. 포인터배열

2018년 10월 20일 토요일 오후 9:43

5. 포인터 배열

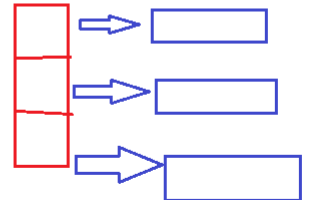
```
void put_b(char **pb)
{
    puts(pb[0]);
    puts(pb[1]);
    puts(pb[2]);
}
```

```
int main()
{
    char a[3][10] = { "aa", "bbb", "cccc" };
    char *b[3] = { "aa", "bbb", "cccc" };

    put_b(b);
}
```

```
int main()
{
    char a[3][10] = { "aa", "bbb", "cccc" };
    char *b[3] = { "aa", "bbb", "cccc" };
}
```

aa ₩0
bbb
cccc



```
int main()
{
    char a[3][10] = { "aa", "bbb", "cccc" };
    char *b[3] = { "aa", "bbb", "cccc" };
    put_b(b);
}
```

aa ₩0
bbb ₩0
cccc ₩0

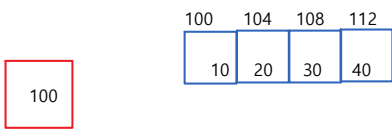
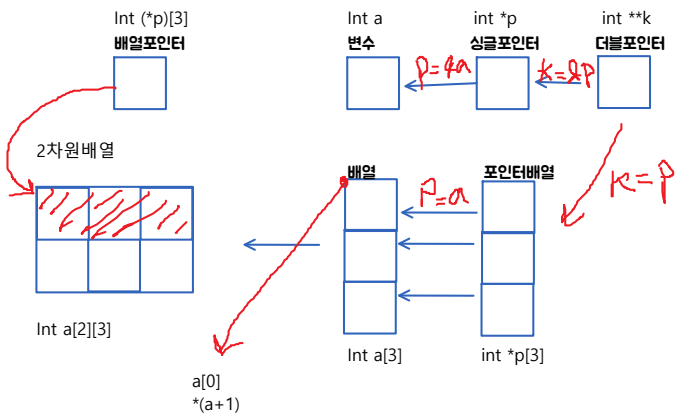
b의 주소를
가리키는 p

```
void put_b(char **pb)
{
    puts(pb[0]);
    puts(pb[1]);
    puts(pb[2]);
}
```



-----포인터 정리-----

2018년 10월 23일 화요일 오전 10:00



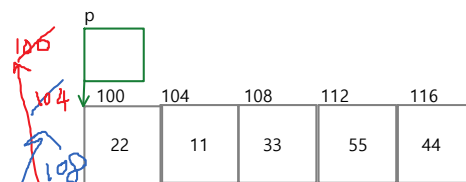
값 : 주소

*p	10:100
*(p+1)	20 : 100
*p+1	11:100
*p += 1	p = 11 :100
*p ++	10 : 104
-> p = p+1	
(*p)++	20(다음에는21) : 104
++ *p	22 : 104
* ++p	30:108

(*104)번지값 20출력하고 20에++해라

현재 포인터가 보고있는 값을 ++

<예제>



*p+=1	23:100	값 :영구	포인터 값에 +1
*p++	23:104	주소:영구	현재 p가 가리키는 주소의 값 출력하고 주소에 ++
++ *p	12:104	값:영구	104번지에 있는 값을 ++해라
*(p+1)	33 :104	주소:임시	
(*p)++	12(13):104	값:영구	값을 ++ 한다. 다음 이 주소 출력시 13
++p	33:108	주소:영구	
*p+1	34:108	값:임시	

값:주소 변한것:지속

->예제

2018년 10월 23일 화요일 오전 10:31

```
void my_strcpy(char *dst, char *src)
{
    while (*dst++ = *src++);
    //int dx;
    //for (dx = 0; src[dx] != '\0'; dx++)
    //{
    //    dst[dx] = src[dx];
    //}
    //dst[dx] = '\0';
}

int main()
{
    char src[] = "abcdef";
    char dst[10];

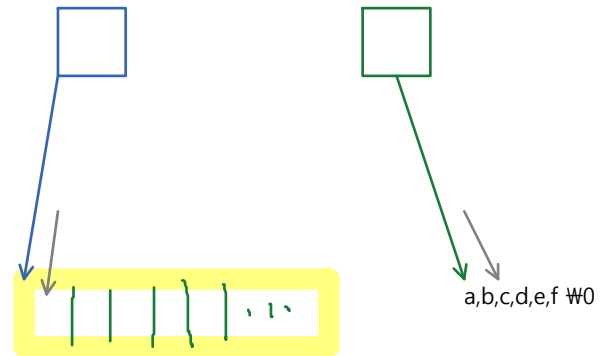
    my_strcpy(dst, src);

    puts(dst);
}
```

Microsoft Visual Studio 디버그

abcdef

D:\C\Day1\64\Debug\Day1...
이 창을 닫으려면 아무 키나



코드가 짧아지고, 걸어다니면서 수박 던지는 느낌으로 코드가 빨라짐

전처리

2018년 10월 23일 화요일 오전 10:37

`#ifndef apple` (apple이 있니?) : 만약 apple이 없다면 이 코드는 true!
위의 코드가 true가 되면 아래코드로

```
#define apple : 너 사과없어? 그럼 이거 실행해!  
--코드들  
-----  
---  
--
```

`#endif` => 이렇게 끝남 나중에

`#ifdef apple` : 이제 위에서 apple 받았으니까 이 코드는 false!!!
위의 코드가 false가 되면 밑에 코드 생략

```
#define apple  
--코드들  
-----  
---  
--
```

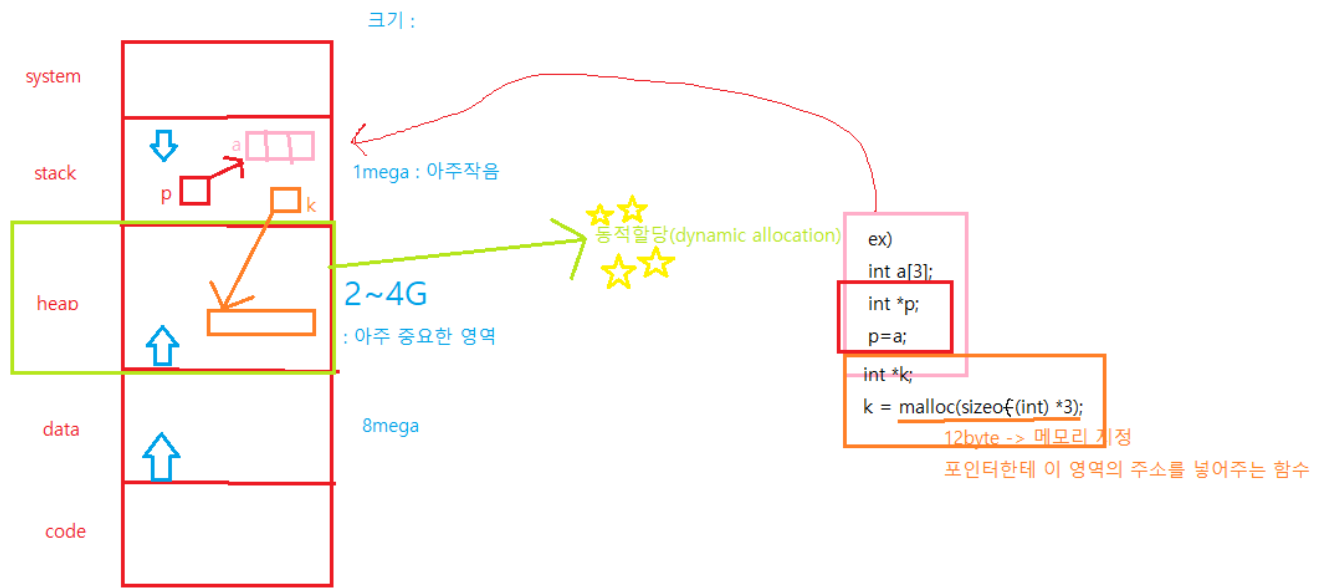
`#endif` => 이렇게 끝남 나중에

=> 정의되지 않았으면 정의 해줄게, 정의가 됐다면 안 해줘!

`#pragma once` -> 만약 `#ifndef`가 참조되지 않으면 이걸로 막겠다 !

12. 동적 메모리 할당

2018년 10월 20일 토요일 오후 9:43



12-2. 동적 할당

2018년 10월 23일 화요일 오전 10:49

변수

```
struct _list
{
    int key;
};

int main()
{
    struct _list a;
}
```

List구조에 int형 변수 메모리가 들어감



```
struct _list
{
    int key;
};

int main()
{
    struct _list a;
    a.key = 10;
    printf("%d\n", a.key);
}
```

Microsoft Visual Studio 디버그 콘솔

10

```
struct _list //서로 다른 데이터를 묶어 저장
{
    char name[20];
    char address[30];
    int age;
};

int main()
{
    struct _list a;
    //a.name = "정은"; -> 에러
    strcpy(a.name, "정은");
    a.age = 10;
    printf("%d\n", a.age);
    printf("%s\n", a.name);
}
```

Microsoft Visual Studio 디버그 콘솔

10
정은

변수 구조체 바꾸려면

```
struct _list //서로 다른 데이터를 묶어 저장
{
    char name[20];
    char address[30];
    int age;
};
```

P가 a주소에 접근

100

```
struct _list //서로 다른 데이터를 묶어 저장
{
    char name[20];
    char address[30];
    int age;
};

void change_a(struct _list *p)
```

```

char address[30];
int age;

void change_a(struct _list *p)
{
    p->a = 20; ->와실표 연산자를 통해 바꿀 수 있음
    strcpy(p->name, "이정은");
}

int main()
{
    struct _list a;
    //a.name = "정은"; -> 에러
    strcpy(a.name, "정은");
    a.age = 10;

    //change_a(a); ->에러! => 구조체 변수의 주소를 던져야 함!
    change_a(&a);

    printf("%d\n", a.age);
    printf("%s\n", a.name);
}

```

```

int age;
};

void change_a(struct _list *p)
{
    p->age = 20;
    strcpy(p->name, "은정");
}

int main()
{
    struct _list a;
    //a.name = "정은"; -> 에러
    strcpy(a.name, "정은");
    a.age = 10;

    //change_a(a);
    change_a(&a);

    printf("%d\n", a.age);
    printf("%s\n", a.name);
}

```



Microsoft Visual Studio 디버그 콘솔
D:\WC\Day1\64\Debug\WC...
이 창을 닫으려면 아무...

코드 볼 때는 struct 먼저 보고 main

```

struct _face {
    float eye;
};

struct _list { //서로 다른 데이터를 묶어 저장
    char name[20];
    char address[30];
    int age;
    struct _face fc;
};

void change_a(struct _list *p)
{
    p->age = 20;
    strcpy(p->name, "은정");
}

int main()
{
    struct _list a;
    //a.name = "정은"; -> 에러
    strcpy(a.name, "정은");
    a.age = 10;

    //change_a(a);
    change_a(&a);

    printf("%d\n", a.age);
    printf("%s\n", a.name);
}

```

main에서
a.fc.eye로 접근

```

struct _face {
    float eye;
};

struct _list { //서로 다른 데이터를 묶어 저장
    char name[20];
    char address[30];
    int age;
    struct _face fc;
};

void change_a(struct _list *p)
{
    p->age = 20;
    strcpy(p->name, "은정");
}

int main()
{
    struct _list a;
    //a.name = "정은"; -> 에러
    strcpy(a.name, "정은");
    a.age = 10;
    a.fc.eye = 2.0f;

    //change_a(a);
    change_a(&a);
    printf("%f\n", a.fc.eye);
    printf("%d\n", a.age);
    printf("%s\n", a.name);
}

```

Microsoft Visual Studio 디버그 콘솔
2.000000
20
은정
D:\WC\Day1\64\Debug\WC...
이 창을 닫으려면 아무...

시력 바꾸고 싶을 땐 change_a에서 p->fc.eye

```
[};  
void change_a(struct _list *p)  
{  
    p->age = 20;  
    strcpy(p->name, "은정");  
    p->fc.eye = 2.5f;  
}  
int main()  
{  
    struct _list a;
```

Microsoft Visual
2.500000
20
은정

12-2-1 예제1

2018년 10월 23일 화요일 오전 11:27

```
struct num {
    int fir;
    int sec;
};

int rlt;

void func1(int a, int b)
{
    rlt = a + b;
}

void func2(struct num *p)
{
    rlt = p->fir * p->sec;
}

void func3(struct num *p)
{
    rlt = p->fir + p->sec + (p + 1)->fir + (p + 1)->sec;
}
```

Diagram annotations for func3:

- `p` points to address 100.
- `p->fir` is 100.
- `p->sec` is 56.
- `(p + 1)->fir` is 108.
- `(p + 1)->sec` is 78.

printf결과

```
1 + 2 = 3
12
26
```

중요함~

```
void main()
{
    struct num a = { 1,2 }, b = { 3,4 }, c[2] = { 5,6,7,8 };
    //구조체멤버 각각 던져서 합
    func1(a.fir, a.sec);

    //구조체 변수 주소 던져서 곱
    func2(&b);

    //구조체 배열주소 던져서 합
    func3(c);
}
```

Memory layout for array `c`:

c: 100	56	c[0]
c: 108	78	c[1]

12-2-2 예제2

2018년 10월 23일 화요일 오전 11:48

문)
국어점수, 영어점수, 수학점수, 총점, 랭킹(순위)를 매기고 출력하세요.

```

struct subject{
    int subname[5]; //[국어], [영어], [수학], [총점], [랭킹]
};
struct sung{
    char name[10];
    float avg; //평균
    struct subject sub;
};
    
```

GUI Output:

```

1번째 사람이름 : 홍길동
국어점수 : 10
영어점수 : 20
수학점수 : 30
2번째 사람이름 : 이순신
국어점수 :
영어점수 :
수학점수 :
3번째 사람이름 : 둘리
국어점수 :
영어점수 :
수학점수 :
    
```

성명	국어	영어	수학	총점	순위	평균
홍길동	10	20	30	60		
이순신						
둘리						

결과)
1번째 사람이름 : 홍길동


```

struct subject {
    int subname[5]; //국 영 수 총점 랭킹
};

struct sung {
    char name[10]; //이름
    float avg;      //평균
    struct subject sub; //각과목의 점수
};

int main()
{
    struct sung man[3]; //3명이 있어야함 [0]:홍길동 [1]:이순신 [2]:둘리
    char *s[] = { "국어", "영어", "수학" };
    int dx, cx;

    // 이름-자바칩 // 국영수
    for (dx = 0; dx < 3; dx++) //3번
    {
        scanf("%s", man[dx].name); // == gets(man[dx].name); ->사람이름
        man[dx].sub.subname[3] = 0; //총점의 초기화 -> 밑에서 총점 한번 구하면 초기화하고 다시
        //국영수
        for (cx = 0; cx < 3; cx++)
        {
            printf("%s점수 : ", s[cx]);
            scanf("%d", &man[dx].sub.subname[cx]);
            man[dx].sub.subname[3] += man[dx].sub.subname[cx];
        }

        //점수 다 받으면 평균구함
        //왼쪽은 정수형이라 오른쪽 형변환(캐스팅)해줘야 값이 들어감
        man[dx].avg = man[dx].sub.subname[3]/(float)3;
    }

    for (dx = 0; dx < 3; dx++) //3번
    {
        for (cx = 0; cx < 3; cx++)
        {
            if (man[dx].sub.subname[3] < man[cx].sub.subname[3]) {
                man[dx].sub.subname[4]++;
            }
        }
    }

    for (dx = 0; dx < 3; dx++) //3번
    {
        printf("%s", man[dx].name);
        for (cx = 0; cx < 5; cx++)
        {
            printf("%d\t", man[dx].sub.subname[cx]);
        }
        puts(""); //개행문자 한칸 내림
    }
}

```

13 알고리즘

2018년 10월 25일 목요일 오후 3:27

책 : 이재규 - C로 배우는 알고리즘 1,2 - 원론적으로 파기 좋음

윤성우(자료구조) - 쉬움, 인강제공
한빛미디어(뇌를 자극하는 알고리즘)

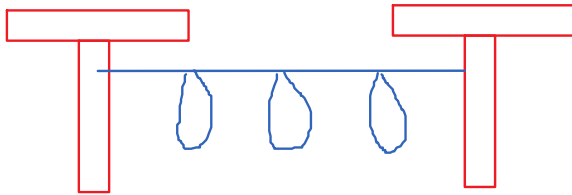
1. List(순서가있는자료구조) - Array(1,2,3), struct(linked list) --> kernel(double linked list)
2. stack - LIFO(last input first out) - 컴파일러, 함수 콜 스택 (비트연산자, 전처리기)
3. queue - FIFO - buffer, 층별순회(tree)-최단거리 알고리즘(graph)
 - 우선순위 큐(priority Queue) - 우선순위탐색알고리즘(최소비용순회)
 - heap
 - 게임 A-Star
4. tree - 기업의 70%(사원) -이진트리(균형잡힌 이진트리에 대해 설명 : Red-Black Tree)
5. graph - 다익스트라알고리즘(최단거리)
6. sort search : 알고리즘의 꽃 - 일반적 정렬, queue를 알아야, tree 를 알아야 되는 정렬이 있음
 - hash - 갤럭시 노트9,8,5는 내부구조가 hash-tree -> 속도가 항상 일정하게 빠름
 - 균형 잡힌:자동균형
7. 자료구조(저장), 알고리즘(처리)

14 - single linked List

2018년 10월 25일 목요일 오후 3:43

typedef-> 구조체의 이름 미리 정함

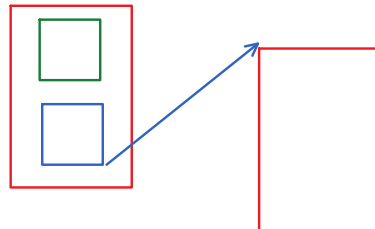
```
typedef struct _list List;
```



알고리즘? 양쪽에 기둥을 세우고 빨래 줄에 빨래를 거는 느낌
데이터를 어떻게 걸지.

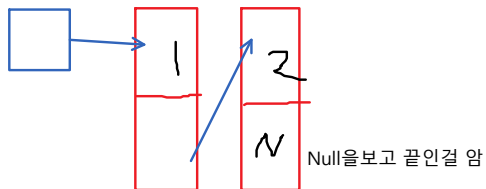
```
typedef struct _list List;
struct _list {
    int key;
    struct _list *next;
};
```

기차로 치면 key는 승객
Next는 기차와 기차간 연결

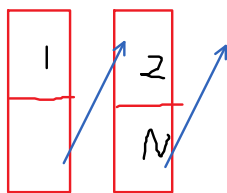


포인트 접근

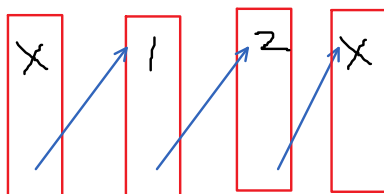
1



2

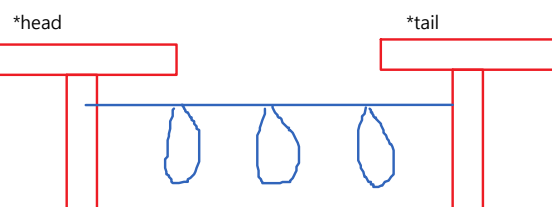


3



```
//모든 자료구조는 반드시 초기화
void init_List(void)
{
    // 초기화
}

int main()
{
    // 초기화
}
```

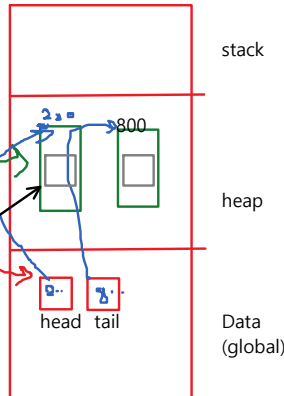


```
int main()
{
    // ...
}
```



```
List *head, *tail;
//모든 자료구조는 반드시 초기화
void init_List(void)
{
    head = (List *)malloc(sizeof(List)); //왼쪽
    tail = (List *)malloc(sizeof(List)); //오른쪽
}
int main()
{
    // ...
}
```

```
struct _list {
    int key;
    struct _list *next;
};
// *next는 기동을 만드는거
```

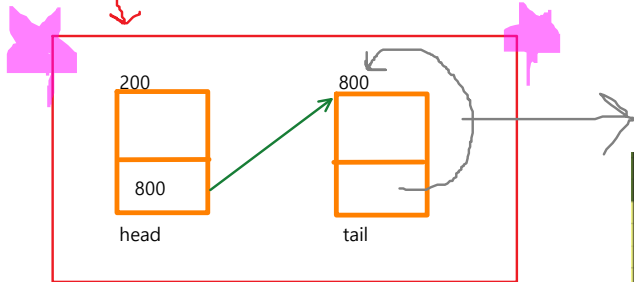


head -> key = 50; : 헤드가 보고있는 키 위치에 50을 넣겠다

```
typedef struct _list List;
struct _list {
    int key;
    struct _list *next;
};

List *head, *tail;
//모든 자료구조는 반드시 초기화
void init_List(void)
{
    head = (List *)malloc(sizeof(List)); //왼쪽
    tail = (List *)malloc(sizeof(List)); //오른쪽
    head->next = tail; //빨래줄
}
int main()
{
    init_List(); //빨래기동2개, 빨래줄
}
```

기동만들고
빨래줄만들
빨래 함수를 부르는 코드



tail->next = tail;

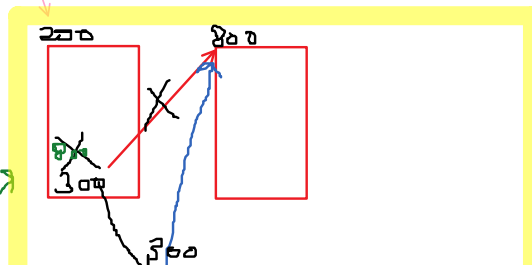
```
void init_List(void)
{
    head = (List *)malloc(sizeof(List)); //왼쪽
    tail = (List *)malloc(sizeof(List)); //오른쪽
    head->next = tail; //빨래줄
    tail->next = tail;
}
```

head

새로운 게 생기면 내가 보고있던걸
새로운 애 한테 가라리게 하고 내가 새로운 애를 봄

```
typedef struct _list List;
struct _list {
    int key;
    struct _list *next;
};

List *head, *tail;
//모든 자료구조는 반드시 초기화
void init_List(void)
{
    head = (List *)malloc(sizeof(List)); //왼쪽
    tail = (List *)malloc(sizeof(List)); //오른쪽
    head->next = tail; //빨래줄
    tail->next = tail;
}
```



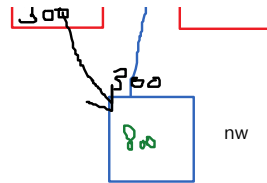
```

head = (List *)malloc(sizeof(List)); //왼쪽
tail = (List *)malloc(sizeof(List)); //오른쪽
head->next = tail; //팔래줄
tail->next = tail;

void insert_head(int data) //숫자 팔래화
{
    List *nw = (List *)malloc(sizeof(List));
    nw->key = data;
}

int main()
{
    init_List(); //팔래기동2개, 팔래줄
    insert_head(3); //첫번째 팔래 넣기
}

```



nw->next = head->next; : 내가 보는 거 니가 봐
 Head->next = nw : 새로운 건 내가 볼게

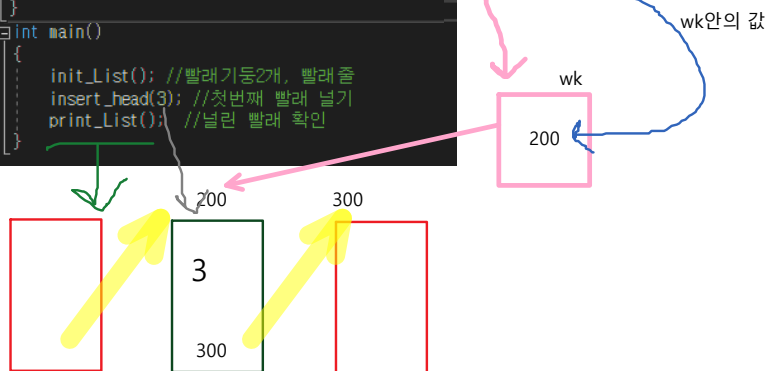
Head포인터가 가리키는 영역임

```

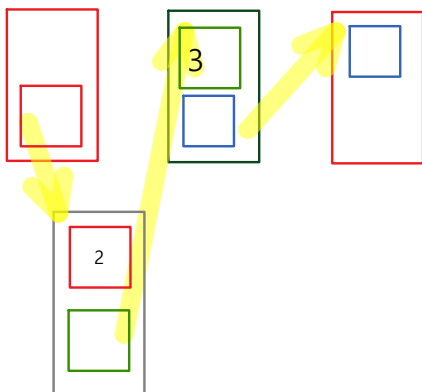
void print_List(void) //오로지 팔래만 볼
{
    List *wk; //팔래 걷는 녀석
    wk = head->next; //첫번째 팔래를 봐야함
    printf("%d\n", wk->key);
}

int main()
{
    init_List(); //팔래기동2개, 팔래줄
    insert_head(3); //첫번째 팔래 넣기
    print_List(); //날린 팔래 확인
}

```



만약 또 추가되면



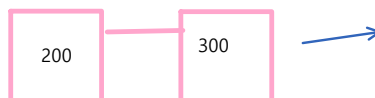
```

void print_List(void) //오로지 팔래만 볼
{
    List *wk; //팔래 걷는 녀석
    wk = head->next; //첫번째 팔래를 봐야함
    printf("%d\n", wk->key);
}

int main()
{
    init_List(); //팔래기동2개, 팔래줄
    insert_head(3); //첫번째 팔래 넣기
    print_List(); //날린 팔래 확인
}

```

이제 애가 300을 가리켜야함. wk = wk->next;



```

void print_List(void) //오로지 팔래만 볼
{
    List *wk; //팔래 걷는 녀석
    wk = head->next; //첫번째 팔래를 봐야함
    printf("%d\n", wk->key);
    wk = wk->next;
    printf("%d\n", wk->key);
}

int main()
{
    init_List(); //팔래기동2개, 팔래줄
    insert_head(3); //첫번째 팔래 넣기
    print_List(); //날린 팔래 확인
}

```

Diagram illustrating a linked list structure with nodes containing key-value pairs. The nodes are:

- Node 1: key=2, value=300
- Node 2: key=3, value=400
- Node 3: key=4, value=500
- Node 4: key=5, value=null

A pointer 'wk' is shown pointing to the first node. A blue line indicates the traversal path. A red box highlights the first node. A green box highlights the fourth node. A yellow box highlights the fifth node. A blue box highlights the sixth node. A red box highlights the seventh node. A blue box highlights the eighth node. A red box highlights the ninth node. A blue box highlights the tenth node. A red box highlights the eleventh node. A blue box highlights the twelfth node. A red box highlights the thirteenth node. A blue box highlights the fourteenth node. A red box highlights the fifteenth node. A blue box highlights the sixteenth node. A red box highlights the seventeenth node. A blue box highlights the eighteenth node. A red box highlights the nineteenth node. A blue box highlights the twentieth node. A red box highlights the twenty-first node. A blue box highlights the twenty-second node. A red box highlights the twenty-third node. A blue box highlights the twenty-fourth node. A red box highlights the twenty-fifth node. A blue box highlights the twenty-sixth node. A red box highlights the twenty-seventh node. A blue box highlights the twenty-eighth node. A red box highlights the twenty-ninth node. A blue box highlights the thirtieth node. A red box highlights the thirty-first node. A blue box highlights the thirty-second node. A red box highlights the thirty-third node. A blue box highlights the thirty-fourth node. A red box highlights the thirty-fifth node. A blue box highlights the thirty-sixth node. A red box highlights the thirty-seventh node. A blue box highlights the thirty-eighth node. A red box highlights the thirty-ninth node. A blue box highlights the fortieth node. A red box highlights the forty-first node. A blue box highlights the forty-second node. A red box highlights the forty-third node. A blue box highlights the forty-fourth node. A red box highlights the forty-fifth node. A blue box highlights the forty-sixth node. A red box highlights the forty-seventh node. A blue box highlights the forty-eighth node. A red box highlights the forty-ninth node. A blue box highlights the fiftieth node. A red box highlights the fifty-first node. A blue box highlights the fifty-second node. A red box highlights the fifty-third node. A blue box highlights the fifty-fourth node. A red box highlights the fifty-fifth node. A blue box highlights the fifty-sixth node. A red box highlights the fifty-seventh node. A blue box highlights the fifty-eighth node. A red box highlights the fifty-ninth node. A blue box highlights the sixtieth node. A red box highlights the sixty-first node. A blue box highlights the sixty-second node. A red box highlights the sixty-third node. A blue box highlights the sixty-fourth node. A red box highlights the sixty-fifth node. A blue box highlights the sixty-sixth node. A red box highlights the sixty-seventh node. A blue box highlights the sixty-eighth node. A red box highlights the sixty-ninth node. A blue box highlights the seventieth node. A red box highlights the seventy-first node. A blue box highlights the seventy-second node. A red box highlights the seventy-third node. A blue box highlights the seventy-fourth node. A red box highlights the seventy-fifth node. A blue box highlights the seventy-sixth node. A red box highlights the seventy-seventh node. A blue box highlights the seventy-eighth node. A red box highlights the seventy-ninth node. A blue box highlights the eightieth node. A red box highlights the eighty-first node. A blue box highlights the eighty-second node. A red box highlights the eighty-third node. A blue box highlights the eighty-fourth node. A red box highlights the eighty-fifth node. A blue box highlights the eighty-sixth node. A red box highlights the eighty-seventh node. A blue box highlights the eighty-eighth node. A red box highlights the eighty-ninth node. A blue box highlights the ninetieth node. A red box highlights the ninety-first node. A blue box highlights the ninety-second node. A red box highlights the ninety-third node. A blue box highlights the ninety-fourth node. A red box highlights the ninety-fifth node. A blue box highlights the ninety-sixth node. A red box highlights the ninety-seventh node. A blue box highlights the ninety-eighth node. A red box highlights the ninety-ninth node. A blue box highlights the hundredth node.

2,3,4,5 다 출력됨

```
void print_List(void) //오로지 팔래만 봄
{
    List *wk; //팔래 걷는 녀석
    wk = head->next; //첫번째 팔래를 봐야함
    while (wk != tail)//wk가 끝기둥이 아닐때만 볼거야
    {
        printf("%d\n", wk->key);
        wk = wk->next;
    }
}
```

```
void print_List(void) //오로지 팔래만 봄
{
    List *wk; //팔래 걷는 녀석
    wk = head->next; //첫번째 팔래를 봐야함
    while (wk != tail) //wk가 끝기둥이 아닐때만 볼거야
    {
        printf("%d\n", wk->key);
        wk = wk->next;
    }
    //printf("%d\n", wk->key);
    //wk = wk->next;
    //printf("%d\n", wk->key);
}
```

Diagram illustrating a linked list deletion operation. The linked list contains nodes with values 200, 300, 400, and 500. The head pointer points to the node with value 200. The diagram shows the process of deleting the node with value 300. The next pointer of the node with value 200 is updated to point to the node with value 400, bypassing the node with value 300. The next pointer of the node with value 300 is crossed out, and the next pointer of the node with value 400 is updated to point to the node with value 500. The next pointer of the node with value 500 is crossed out, indicating the end of the list. The diagram also shows the state of the linked list after deleting the node with value 300, where the head pointer still points to the node with value 200, and the next pointer of the node with value 200 points to the node with value 400. The next pointer of the node with value 400 points to the node with value 500, and the next pointer of the node with value 500 is null.

==> 이 방법은 200을 그냥 메모리 뺏두니까 손해 -> 완전히 처리
kill = head->next; 기억했다가(불잡고있다가)

이 주황부분을 도려내고 싶다!!

```

graph LR
    head[head] -- next --> 200[200]
    200 --> 300[300]
    300 --> empty_box[ ]
    style 200 fill:#fff,stroke:#f96,stroke-width:2px
    style 300 fill:#fff,stroke:#f96,stroke-width:2px
    linkStyle 0 stroke:#f96,stroke-width:2px
    linkStyle 1 stroke:#f96,stroke-width:2px
    linkStyle 2 stroke:#f96,stroke-width:2px
  
```

head -> next
 200->next
 head -> next -> next -> 300

head의 next는 200이다. head의 next의 next는 300 이기에
 head -> next -> next -> 300이 된다.

```
void delete_head(void)
{
    List *kill;
    kill = head->next;
    head->next = head->next->next;
}
```

kill = head->next; 기억했다가(붙잡고있다가)

```

{
    List *kill;
    kill = head->next;

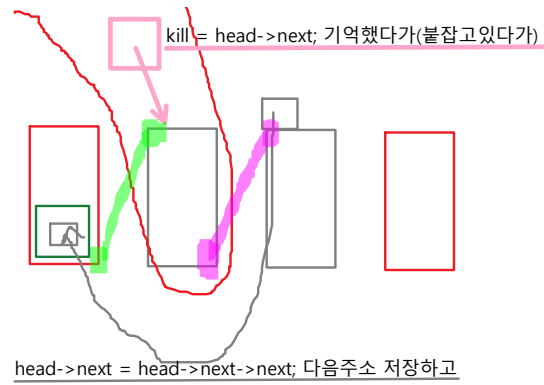
    head->next = head->next->next;

    free(kill);
}

int main()
{
    init_List(); //팔래기등2개, 팔래 줄
    insert_head(3); //첫번째 팔래 넣기
    insert_head(2); //첫번째 팔래 넣기
    print_List(); //넣린 팔래 확인

    delete_head();
}

```

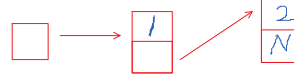


리스트

- 순서가 있는 자료구조

리스트 구조

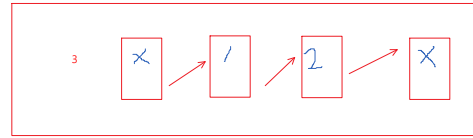
1.



2.



3.



우린 이런 배열을 거다!!

자료구조(List) 코드 적용(예제)

1. 메모리 할당하고 변수 설정하기

자료구조 (List)

```
typedef struct List {
    int key;
    struct List *next;
} List;

// 모든 자료구조는 반드시 초기화
void init_List(void)
{
    // malloc으로 메모리를 할당하는 것임
    head = (List *) malloc(sizeof(List)); // 왼쪽 할당 가능
    tail = (List *) malloc(sizeof(List)); // 오른쪽 할당 가능
    head->next = tail; // 할래용
    tail->next = tail;
}

// 이렇게 초기화 한다.
```

head->key=50;

200 800

stack

heap

data (global)

200 800

head tail

malloc 메모리 할당 명령어

malloc으로 메모리를 할당하고 head와 tail의 메모리를 할당하고 주솟값으로 받는 구조이다.

200과 800은 예시다.

2. 초기화 및 두개의 메모리 생성

2 메모리 할당 선언 예시

```
typedef struct {
    int key;
    struct List *next;
} List;

// 모든 자료구조는 반드시 초기화
void init_List(void)
{
    // malloc으로 메모리를 할당하는 것임
    head = (List *) malloc(sizeof(List)); // 왼쪽 할당 가능
    tail = (List *) malloc(sizeof(List)); // 오른쪽 할당 가능
    head->next = tail; // 할래용
    tail->next = tail;
}

// 이렇게 초기화 한다.
```

선언 예시

200 800

head tail

자기 자신을 받는다.

이런 느낌이다.

기초 세팅을 서로가 서로에게 연결될 수 있게 해주었다.

3. 값 넣어주며 구조 살펴보기

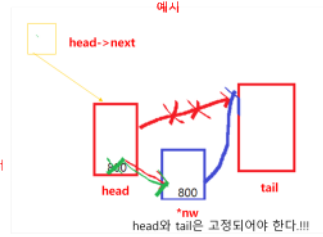
3. 값 넣어주며 구조 살펴보기

```
void insert_List(int data)
{
    List *new = (List *) malloc(sizeof(List));
    new->key = data;
    new->next = head;
    head = new;
    tail = tail->next;
}

// 모든 자료구조는 반드시 초기화
void init_List(void)
{
    // malloc으로 메모리를 할당하는 것임
    head = (List *) malloc(sizeof(List)); // 왼쪽 할당 가능
    tail = (List *) malloc(sizeof(List)); // 오른쪽 할당 가능
    head->next = tail; // 할래용
    tail->next = tail;
}

// 이렇게 초기화 한다.
```

중간에 포인터가 있어서 메모리를 받아주는거!!



4. 도와주는 포인터 만들기(메모리)

이 포인터 때문에 대부분의 공학도가 포기한다!!

4. 도와주는 포인터 만들기(메모리)

```
void insert_head(int data)
{
    List *new = (List *) malloc(sizeof(List));
    new->key = data;
    new->next = head;
    head = new;
    tail = tail->next;
}

// 모든 자료구조는 반드시 초기화
void init_List(void)
{
    // malloc으로 메모리를 할당하는 것임
    head = (List *) malloc(sizeof(List)); // 왼쪽 할당 가능
    tail = (List *) malloc(sizeof(List)); // 오른쪽 할당 가능
    head->next = tail; // 할래용
    tail->next = tail;
}

// 이렇게 초기화 한다.
```

원하고 있는 녀석의 next를 가르킨다.

이 주황부분을 도려내고 싶다!!

이 주황부분을 도려내고 싶다!!

5. 리스트로 모든 값 보여주기

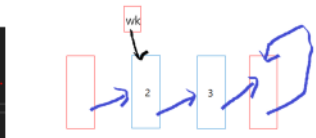
5.리스트 형태로 모든 값 보여주기

```
void print_List(void)
{
    List *p = head;
    while (p != NULL)
    {
        printf("%d ", p->key);
        p = p->next;
    }
}

// 모든 자료구조는 반드시 초기화
void init_List(void)
{
    // malloc으로 메모리를 할당하는 것임
    head = (List *) malloc(sizeof(List)); // 왼쪽 할당 가능
    tail = (List *) malloc(sizeof(List)); // 오른쪽 할당 가능
    head->next = tail; // 할래용
    tail->next = tail;
}

// 이렇게 초기화 한다.
```

tail이 아니하면 계속 하겠다.



6-1 값을 없애고 싶을 때 예시

6.값을 없애고 싶을 때 예시



env(1636)

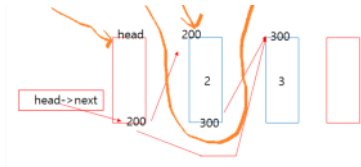
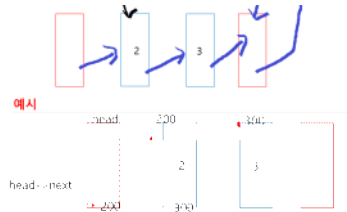
```

List *wk; //할때같은 노드
while(head->next != 0) { //다음의 노드를 꺼내서 처리한다
    wk = head->next; //다음의 노드를 꺼내서 처리한다
    head->next = wk->next; //다음의 노드를 꺼내서 처리한다
    delete head; //다음의 노드를 꺼내서 처리한다
    head = wk; //다음의 노드를 꺼내서 처리한다
}

//main()
init_List(); //할때같은 노드, 할때같은
insert_head(2); //가장 먼저 들어 올리기
insert_head(2); //가장 먼저 들어 올리기
print_List(); //출력 할때 보기

//int main()
{
    init_List(); //할때같은 노드, 할때같은
    insert_head(2); //가장 먼저 들어 올리기
    insert_head(2); //가장 먼저 들어 올리기
    delete_head(); //가장 먼저 들어 올리기
    print_List(); //출력 할때 보기
}

```



head->next = 200->next
head->next->next->next = 300
head의 next는 200이다. head의 next의 next는 300이다. head->next->next->next = 300이 된다.

6-2리스트로 모든 값 보여주기

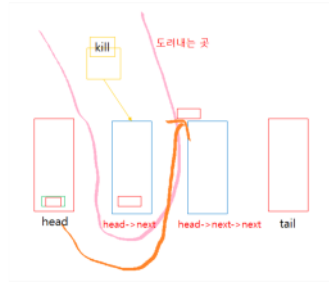
6-2. 값을 없애고 시킬 때 적용

```

void delete_head(void)
{
    delete head;
    head = head->next; //head에 next를 넣어준다.
    head->next = head->next->next;
    print_List();
}

//int main()
{
    init_List(); //할때같은 노드, 할때같은
    insert_head(2); //가장 먼저 들어 올리기
    insert_head(2); //가장 먼저 들어 올리기
    delete_head(); //가장 먼저 들어 올리기
    print_List(); //출력 할때 보기
}

```



7. 값찾기 예제

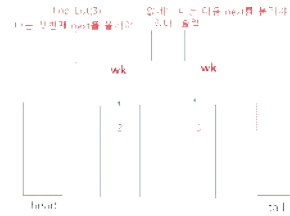
7. 값찾기

```

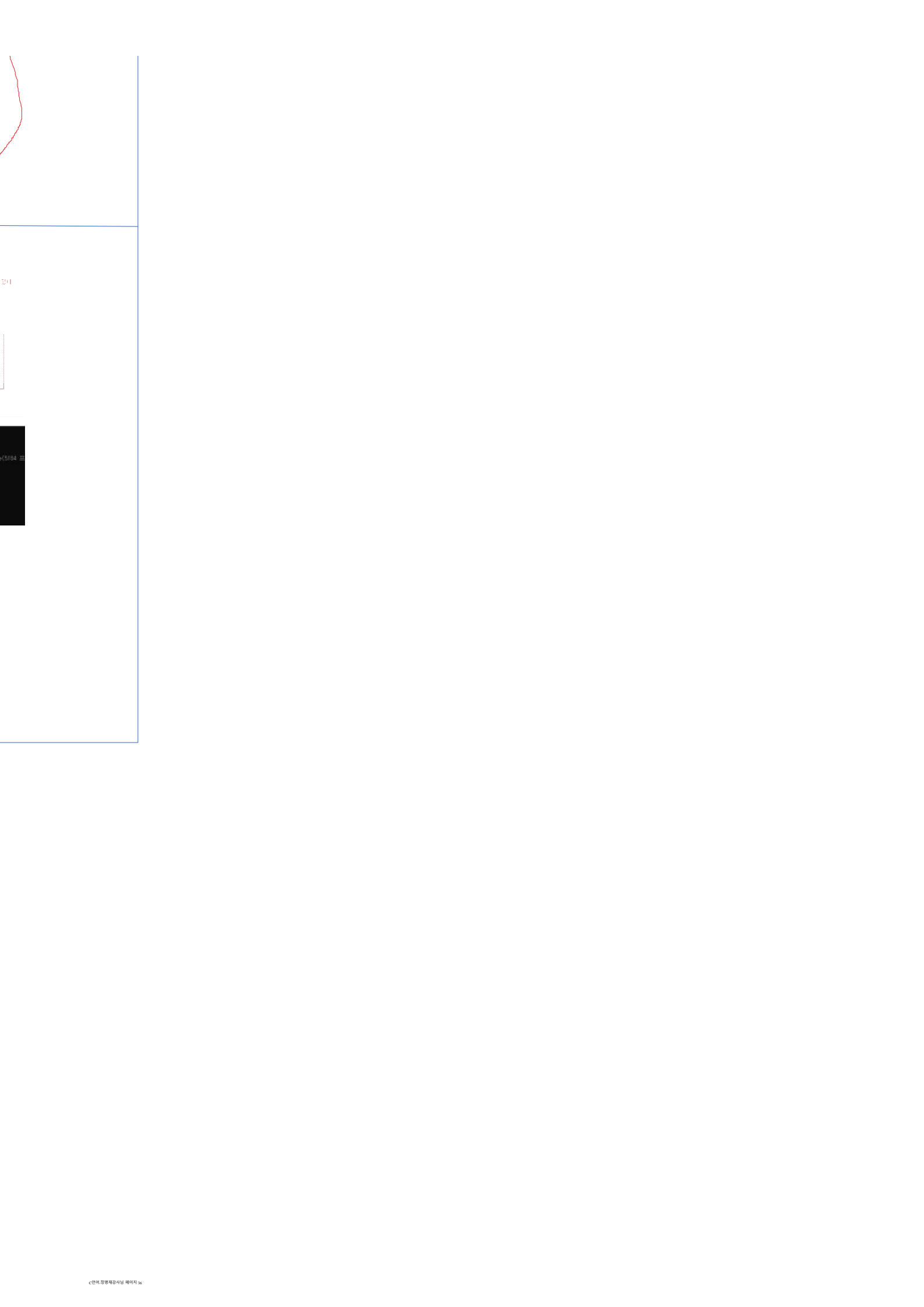
//int find_List(int find)
{
    List *wk; //할때같은 노드
    wk = head; //가장 먼저 들어 올리기
    while(wk != 0)
    {
        if(wk->data == find)
        {
            return wk;
        }
        wk = wk->next;
    }
    return 0;
}

//int main()
{
    init_List(); //할때같은 노드, 할때같은
    insert_head(2); //가장 먼저 들어 올리기
    insert_head(2); //가장 먼저 들어 올리기
    print_List(); //출력 할때 보기
    find_List(2); //가장 먼저 들어 올리기
    find_List(3); //가장 먼저 들어 올리기
}

```



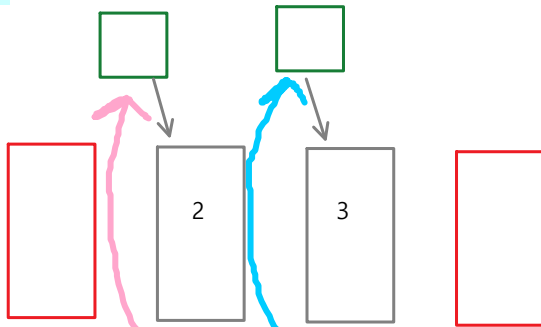
Microsoft Visual Studio 2019 콘솔
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 520
 521
 522
 523
 524
 525
 526
 527
 528
 529
 530
 531
 532
 533
 534
 535
 536
 537
 538
 539
 540
 541
 542
 543
 544
 545
 546
 547
 548
 549
 550
 551
 552
 553
 554
 555
 556
 557
 558
 559
 560
 561
 562
 563
 564
 565
 566
 567
 568
 569
 570
 571
 572
 573
 574
 575
 576
 577
 578
 579
 580
 581
 582
 583
 584
 585
 586
 587
 588
 589
 590
 591
 592
 593
 594
 595
 596
 597
 598
 599
 600
 601
 602
 603
 604
 605
 606
 607
 608
 609
 610
 611
 612
 613
 614
 615
 616
 617
 618
 619
 620
 621
 622
 623
 624
 625
 626
 627
 628
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 640
 641
 642
 643
 644
 645
 646
 647
 648
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 660
 661
 662
 663
 664
 665
 666
 667
 668
 669
 670
 671
 672
 673
 674
 675
 676
 677
 678
 679
 680
 681
 682
 683
 684
 685
 686
 687
 688
 689
 690
 691
 692
 693
 694
 695
 696
 697
 698
 699
 700
 701
 702
 703
 704
 705
 706
 707
 708
 709
 710
 711
 712
 713
 714
 715
 716
 717
 718
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 740
 741
 742
 743
 744
 745
 746
 747
 748
 749
 750
 751
 752
 753
 754
 755
 756
 757
 758
 759
 760
 761
 762
 763
 764
 765
 766
 767
 768
 769
 770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809
 810
 811
 812
 813
 814
 815
 816
 817
 818
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 910
 911
 912
 913
 914
 915
 916
 917
 918
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 930
 931
 932
 933
 934
 935
 936
 937
 938
 939
 940
 941
 942
 943
 944
 945
 946
 947
 948
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959
 960
 961
 962
 963
 964
 965
 966
 967
 968
 969
 970
 971
 972
 973
 974
 975
 976
 977
 978
 979
 980
 981
 982
 983
 984
 985
 986
 987
 988
 989
 990
 991
 992
 993
 994
 995
 996
 997
 998
 999
 1000
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1098
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1139
 1140
 1141
 1142
 1143
 1144
 1145
 1146
 1147
 1148
 1149
 1150
 1151
 1152
 1153
 1154
 1155
 1156
 1157
 1158
 1159
 1160
 1161
 1162
 1163
 1164
 1165
 1166
 1167
 1168
 1169
 1170
 1171
 1172
 1173
 1174
 1175
 1176
 1177
 1178
 1179
 1180
 1181
 1182
 1183
 1184
 1185
 1186
 1187
 1188
 1189
 1190
 1191
 1192
 1193
 1194
 1195
 1196
 1197
 1198
 1199
 1200
 1201
 1202
 1203
 1204
 1205
 1206
 1207
 1208
 1209
 1210
 1211
 1212
 1213
 1214
 1215
 1216
 1217
 1218
 1219
 1220
 1221
 1222
 1223
 1224
 1225
 1226
 1227
 1228
 1229
 1230
 1231
 1232
 1233
 1234
 1235
 1236
 1237
 1238
 1239
 1240
 1241
 1242
 1243
 1244
 1245
 1246
 1247
 1248
 1249
 1250
 1251
 1252
 1253
 1254
 1255
 1256
 1257
 1258
 1259
 1260
 1261
 1262
 1263
 1264
 1265
 1266
 1267
 1268
 1269
 1270
 1271
 1272
 1273
 1274
 1275
 1276
 1277
 1278
 1279
 1280
 1281
 1282
 1283
 1284
 1285
 1286
 1287
 1288
 1289
 1290
 1291
 1292
 1293
 1294
 1295
 1296
 1297
 1298
 1299
 1300
 1301
 1302
 1303
 1304
 1305
 1306
 1307
 1308
 1309
 1310
 1311
 1312
 1313
 1314
 1315
 1316
 1317
 1318
 1319
 1320
 1321
 1322
 1323
 1324
 1325
 1326
 1327
 1328
 1329
 1330
 1331
 1332
 1333
 1334
 1335
 1336
 1337
 1338
 1339
 1340
 1341
 1342
 1343
 1344
 1345
 1346
 1347
 1348
 1349
 1350



- 예제

2018년 10월 25일 목요일 오후 5:04

찾기



```
void find_List(int data)
{
    List *wk;
    wk = head->next;
    while (wk != tail)
    {
        if (wk->key == data)
            puts("있다.");
        wk = wk->next;
    }
}

int main()
{
    init_List(); //팔래기동2개, 팔래줄
    insert_head(3); //첫번째 팔래 넣기
    insert_head(2); //첫번째 팔래 넣기
    print_List(); //널린 팔래 확인

    //delete_head(); //삭제

    find_List(3); //3이 있는지 없는지
}
```

```
//버전 1
void find_List(int data)
{
    List *wk;
    wk = head->next;
    while (wk != tail)
    {
        if (wk->key == data)
        {
            puts("있다.");
            break;
        }
        wk = wk->next;
    }
    if (wk == tail)
        puts("없다");
}

int main()
{
    init_List(); //팔래기동2개, 팔래줄
    insert_head(3); //첫번째 팔래 넣기
    insert_head(2); //첫번째 팔래 넣기
    print_List(); //널린 팔래 확인

    //delete_head(); //삭제
    find_List(3);
    find_List(4); //3이 있는지 없는지
}
```

```
//버전 2
void find_List2(int data)
{
    List *wk;
    wk = head->next;
    while (wk->key != data && wk != tail)
    {
        wk = wk->next;
    }
    if (wk == tail)
        puts("없다");
    else
        puts("있다");
}

int main()
{
    init_List(); //팔래기동2개, 팔래줄
    insert_head(3); //첫번째 팔래 넣기
    insert_head(2); //첫번째 팔래 넣기
    print_List(); //널린 팔래 확인

    //delete_head(); //삭제
    //find_List(3);
    //find_List(4); //3이 있는지 없는지
    find_List2(3);
}
```

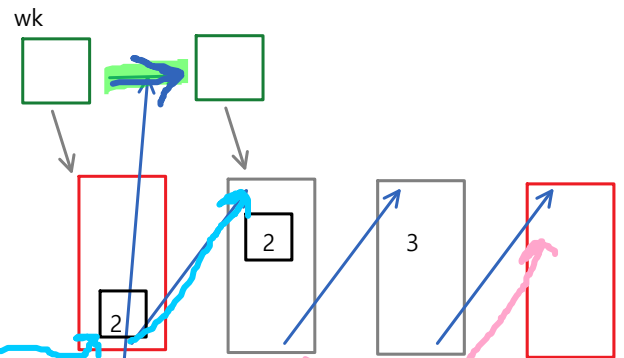
```
//delete_head(); //삭제
//find_List(3);
//find_List(4); //3이 있는지 없는지
find_List2(3);
find_List2(4);
```

3
있다
없다
D:₩₩₩Day1
이 창을 열

찾아서 삭제하기

```
//찾아서 삭제
void find_delete_List(int fd)
{
    List *wk;
    wk = head; //찾아서 삭제해야 하기에 head값을 줘야함

    while (wk->next->key != fd) //내가 찾는 값이 아니라면
    {
        wk = wk->next; //wk를 다음값으로 옮겨
    }
}
```



2가
3 이어야니까

```
//찾아서 삭제
void find_delete_List(int fd)
{
    List *wk;
    wk = head; //찾아서 삭제해야 하기에 head값을 줘야함

    while (wk->next->key != fd && wk->next != tail) //내가 찾는 값이 아니라면
    {
        wk = wk->next; //wk를 다음값으로 옮겨
    }

    if (wk->next != tail) //tail이면 찾았다는거 //위에서 tail을 만나고 나왔다는 뜻은 fd를 못찾았다는거잖아
        wk->next = wk->next->next;
}
```

찾거나, 못찾으면 while 끝,
wk->next->key != fd : 찾았다
wk->next != tail : 못찾았다

Tail을 만나면 while문을 나오니까 -> 값을 못 찾았다

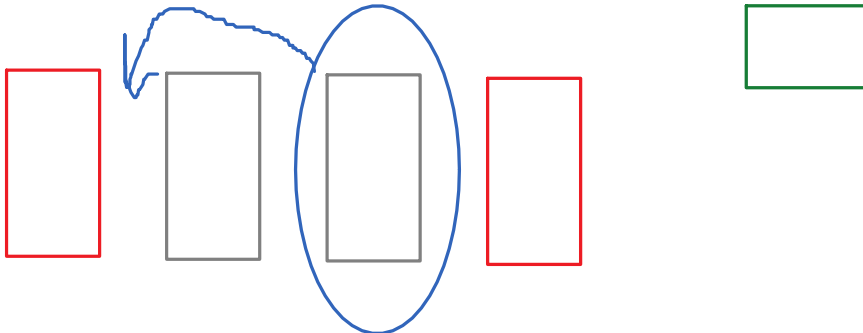
wk->next != tail이면 tail을 못 만나고 나온 거니까 중간에 fd가 있다는 뜻! 그럼 건너뛰라는 뜻

- 기능/예제

2018년 10월 26일 금요일 오전 10:04

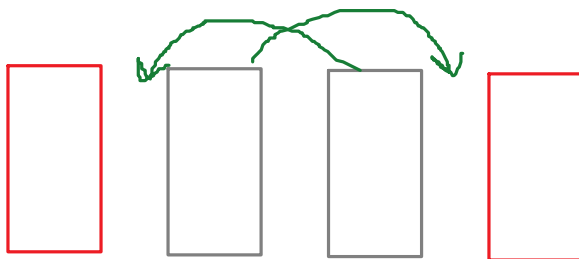
1. Move to front

찾아서 맨앞으로 보내기



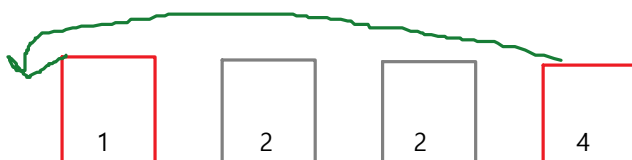
```
void movetofront(int ff)
{
    List *front;
    List *wk;
    wk = head;
    while (wk->next->key != ff && wk->next != tail)
    {
        wk = wk->next;
    }
    if (wk->next != tail)
    {
        front = wk->next; //찾은 값을 기억해 앞으로 넣을거야 잡ㅇㅏ놔
        wk->next = wk->next->next;
        insert_head(front->key); //노드를 넣으면 안됨
    }
}
```

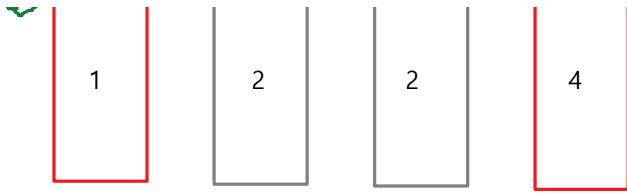
2. transpose method //과제 ! 값 하나로 앞의 값이랑 무조건 바꿔게



3. Frequency count

각 노드의 조희 횟수를 세서 제일 많이 검색한 것을 제일 앞으로 정렬





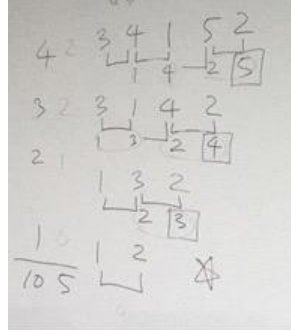
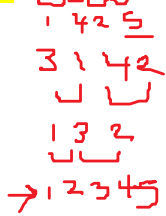
15 sort

2018년 10월 26일 금요일 오전 11:09

sort

-버블, 선택, 삽입, 간접 삽입, 쉘, 퀵, 병합, 버킷, 분도수세기, tree, heap ... 정렬

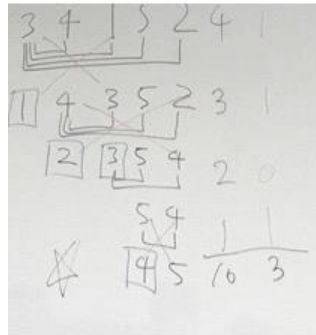
버블 정렬 : 3 4 1 5 2 다 비교해서 바꾸는 거



선택 정렬 : 3 4 1 5 2 다 비교해서 바꾸는 거



하나씩 비교해서 제일 큰 놈 하나만 기억 or
제일 작은 놈 하나 기억해서 앞으로

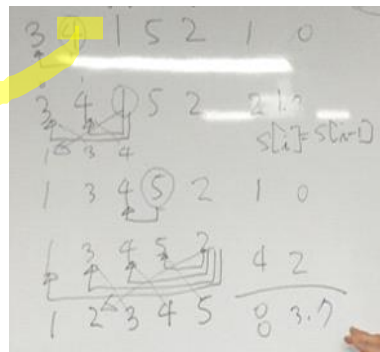


삽입 정렬 : 3 4 1 5 2 하나씩 비교해서 작으면 제일 앞으로 정렬

1번지 값이 기준임

2번지에 1번지값을 넣겠다

$s[i] = s[i-1]$



삽입 간접 정렬 :

3 4 1 5 2

0 1 2 3 4 -> 위 배열의 index를 저장하는 배열을 하나 만들

2 1 3

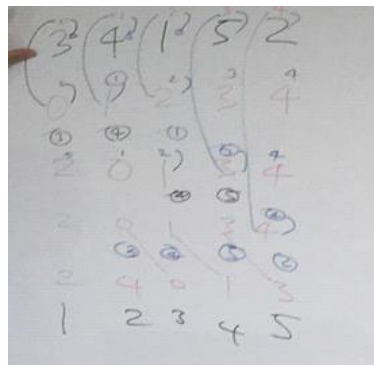
4 5 1 -> 1이 작으니까 1을 제일 앞 번지로

2 0 1 3 4 => 실제 배열 말고 번지수를 바꾸는 거임

4 5

2 0 1 3 4

2 4 0 1 3



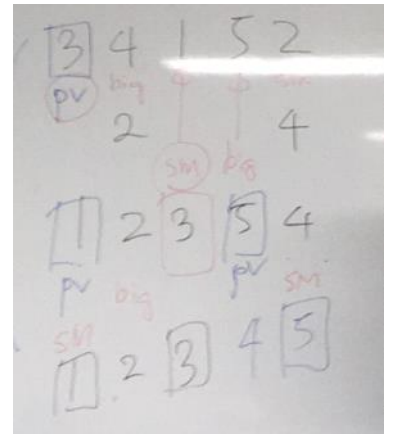
퀵 정렬 : 2. bsearch()

1. Quick sort() : 3을 기준으로

3 4 1 5 2 앞에서 부터 3보다 큰 값을 big
뒤에서 부터는 3보다 작은 값 찾으러 다님 small
찾으면 small을 제일 앞으로 big을 제일 뒤로
바꿔버림 ==> 자기 정렬 끝나면 고정됨.

-> 1 2 3 5 4 이 상태로 정렬 끝나고 이제 1을 기준으로 정렬 -> 자기 뒤에가 big이니까 바로 끝

-> 1 2 3 5 4 -> 2 3 끝났으니까 이제 5를 기준으로 정렬 => 1 2 3 4 5



- 함수 포인터

2018년 10월 26일 금요일 오전 11:36

callback

```
cal(10,5,add)
=> cal(a,b,p(add함수의주소))
p(a,b); == add(10,5);
```

Ex)

```
int add(int a, int b);
```

`int(*p)(int,int);` -> add함수의 주소를 담을 수 있는 포인터

`1 3 4 2` -> int를 리턴하는 함수를 담은 포인터 p

==> `p = add;`

```
int add(int a, int b) { return a + b; }
int sub(int a, int b) { return a - b; }
int mul(int a, int b) { return a * b; }
int div(int a, int b) { return a / b; }

int main()
{
    int(*p)(int a, int b); // add,sub,mul,div 4개 다 저장 할 수 있는 함수
    p = add;
    printf("%d\n", p(10, 5));
    p = sub;
    printf("%d\n", p(10, 5));
    p = mul;
    printf("%d\n", p(10, 5));
    p = div;
    printf("%d\n", p(10, 5));
}
```

반복문으로 돌릴 수 있음

```
int add(int a, int b) { return a + b; }
int sub(int a, int b) { return a - b; }
int mul(int a, int b) { return a * b; }
int div(int a, int b) { return a / b; }

int main()
{
    int(*p[4])(int, int) = { add,sub,mul,div }; //함수 포인터 배열
    int dx;
    for (dx = 0; dx < 4; dx++)
    {
        printf("%d\n", p[dx](10, 5));
    }
}
```

Leaf function call

```
int add(int a, int b) { return a + b; }
int sub(int a, int b) { return a - b; }
int mul(int a, int b) { return a * b; }
int div(int a, int b) { return a / b; }

int middle(int a, int b, int(*p)(int, int))
{
    return p(a, b);
}

int main()
{
    printf("%d\n", middle(10, 5, add));
}
```

add(10,5)
↳ Call back

Main() middle() add()

Leaf function call
바로 main으로

- qsort

2018년 10월 26일 금요일 오후 4:19

```
int main()
{
    int a[] = { 3,4,1,5,2 };
    int dx;
    // 배열 이름, 데이터 수, 하나의 데이터의 메모리 4byte
    // qsort(a, 5, 4, int cmp);

    qsort(a, sizeof(a) / sizeof(a[0]), sizeof(a[0]), int cmp);

    for (dx = 0; dx < 5; dx++)
        printf("%d ", a[dx]);
}
```

Handwritten note: 10 / 4 = 5

```
int int_cmp(const void *a, const void *b)
{
    return *(int *)a - *(int *)b;
}

int main()
{
    int a[] = { 3,4,1,5,2 };
    int dx;
    // 배열 이름, 데이터 수, 하나의 데이터의 메모리 4byte
    // qsort(a, 5, 4, int cmp);

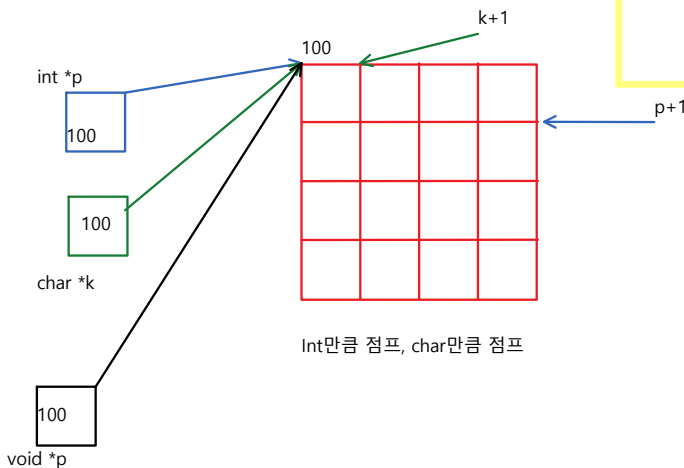
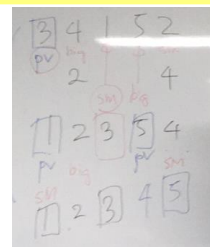
    qsort(a, sizeof(a) / sizeof(a[0]), sizeof(a[0]), int_cmp);

    for (dx = 0; dx < 5; dx++)
        printf("%d ", a[dx]);
}
```

Handwritten note: 너가 갖고있는 100번지 int형 주소야

이 주소 안의 값 의미

qsort 내부 알고리즘을 통해 정렬됨



strcmp("Abc", "ABC");

string 비교해주는 함수

반드시 캐리터형 주소로 받으라고 알려줘야함

```
int str_cmp(const void *a, const void *b)
{
    return strcmp((char *)a, (char *)b);
}

int main()
{
    char b[][10] = {
        "곽병문", "김대민", "김제희", "문성수", "박상현",
        "서연주", "송원석", "안병선", "이정은", "임지영",
        "채희승", "최규성" };

    int dx;

    qsort(b, sizeof(b) / sizeof(b[0]), sizeof(b[0]), str_cmp); // b, 120/12 = 10, 10, 정렬

    for (dx = 0; dx < 5; dx++)
        printf("%d ", a[dx]);
}
```

구조체 정렬

```

int PS_cmp(const void *a, const void *b)
{
    return ((PS *)a)->age - ((PS *)b)->age;
}

int main()
{
    PS c[3] = { {"bbb", 22}, {"aaa", 25}, {"ccc", 23} };

    int a[] = { 3, 4, 1, 5, 2 };
    char b[][10] = {
        "이정은", "임지영", "곽병문", "김대민", "김제희", "문성수", "박상현", "채희승", "최규성",
        "서연주", "송원석", "안병선" };

    int dx;

    qsort(a, sizeof(a) / sizeof(a[0]), sizeof(a[0]), int_cmp);
    qsort(b, sizeof(b) / sizeof(b[0]), sizeof(b[0]), str_cmp);
    qsort(c, sizeof(c) / sizeof(c[0]), sizeof(c[0]), PS_cmp);

    for (dx = 0; dx < 5; dx++)
        printf("%d\n", a[dx]);

    for (dx = 0; dx < 12; dx++)
        printf("%s\n", b[dx]);

    for (dx = 0; dx < 3; dx++)
        printf("%d\n", c[dx].age);
}

```

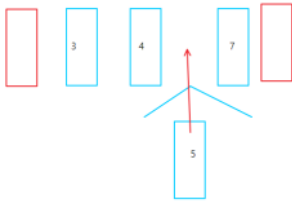
bbb 22	aaa 25	ccc 23
-----------	-----------	-----------

Micros
22
23
25
1

16. double linked list

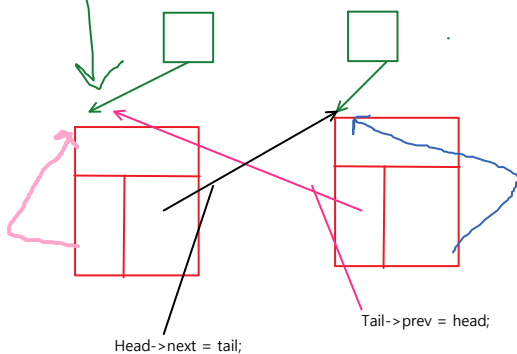
2018년 10월 26일 금요일 오후 5:08

정렬로 넣기

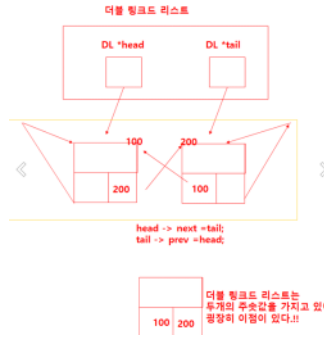


```
typedef struct _dlist DL;
struct _dlist { //double linked list
    int key;
    struct _dlist *prev;
    struct _dlist *next;
};

DL *head, *tail; //팔래 양쪽 기둥
//모든 자료구조는 반드시 초기화
void init_head(void)
{
    head = (DL*)malloc(sizeof(DL)); //왼기둥
    tail = (DL*)malloc(sizeof(DL)); //오른기둥
}
```



```
head->next = tail;
tail->prev = head;
head->prev = head;
tail->next = tail;
```



새로운 데이터 등장

```
typedef struct _dlist DL;
struct _dlist { //double linked list
    int key;
    struct _dlist *prev;
    struct _dlist *next;
};

DL *head, *tail; //팔래 양쪽 기둥
//모든 자료구조는 반드시 초기화
void init_head(void)
{
    head = (DL*)malloc(sizeof(DL)); //왼기둥
    tail = (DL*)malloc(sizeof(DL)); //오른기둥

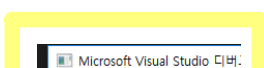
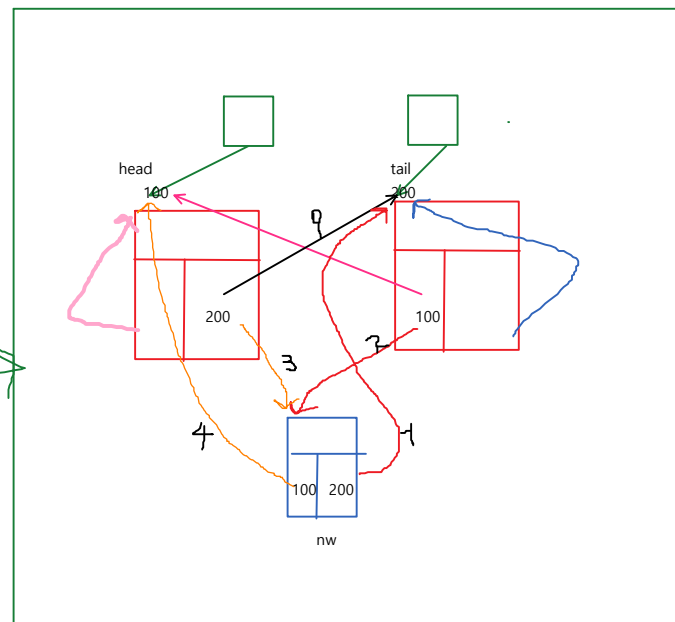
    head->next = tail;
    tail->prev = head;
    head->prev = head;
    tail->next = tail;
}

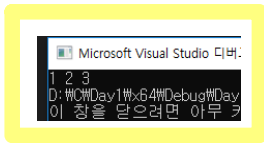
void insert_dlist(int data)
{
    DL *nw = (DL *)malloc(sizeof(DL));
    nw->key = data;

    nw->next = head->next; //0,1 nw의 next 자리에 head의 next 값을 넣어라
    head->next->prev = nw; //2 헤드 넥스트가 보는 위치 값의 prev 위치가 nw를 바라
    head->next = nw; //3 헤드 넥스트가 nw를 바라
    nw->prev = head; //nw의 prev가 head를 바라

}

int main()
{
    init_dlist(); //팔래 기둥과 팔래 줄 완성
    insert_dlist(3);
}
```





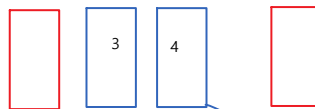
```
void print_dlist(void)
{
    DL *wk = head->next;
    while (wk != tail)
    {
        printf("%d ", wk->key);
        wk = wk->next;
    }
    puts("\n-----");
    wk = tail->prev;
    while (wk != head)
    {
        printf("%d ", wk->key);
        wk = wk->prev;
    }
}

int main()
{
    init_dlist(); //팔레 가동과 팔레 줄 완성
    insert_dlist(3);
    insert_dlist(2);
    insert_dlist(1);
    print_dlist();
}
```

Ordered insert

- 나보다 큰놈 앞에다 데이터 넣어

```
void ordered_insert(int data)
{
    DL *nw = (DL *)malloc(sizeof(DL));
    nw->key = data;
}
```



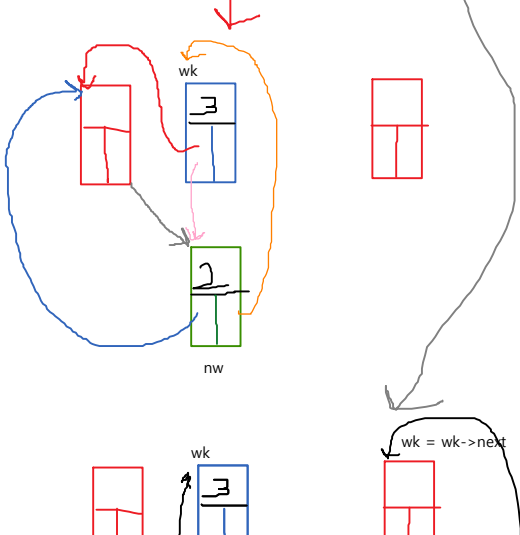
```
void ordered_insert(int data)
{
    DL *wk;
    wk = head->next; //첫번째 팔레 보기

    DL *nw = (DL *)malloc(sizeof(DL));
    nw->key = data;

    while (wk->key < data && wk != tail) //data보다 큰 wk->key를 만나면 멈춤
    {
        wk = wk->next;
    }

    //wk의 앞쪽에 nw를 삽입해야함
    nw->prev = wk->prev;
    wk->prev->next = nw;
    wk->prev = nw;
    nw->next = wk;
}
```

초록색보다 파란색 값이 작으면
wk->next로 넘어가



```
int main()
{
    init_dlist(); //팔레 가동과 팔레 줄 완성
    //insert_dlist(3);
    //insert_dlist(2);
    //insert_dlist(1);
    //print_dlist();

    ordered_insert(3);
    ordered_insert(4);
    ordered_insert(1);
    ordered_insert(2);
    print_dlist();
}
```

