Chapter 6
Structures
스트럭처

Part 1
6.1~6.5

CSE2018 시스템프로그래밍기초
2016년 2학기

한양대학교 ERICA
컴퓨터공학과 => 소프트웨어학부
도경구

1. Basics of Structures
2. Structures and Functions
3. Arrays of Structures
4. Pointers to Structures
5. Self-referential Structures
6. Table Lookup
7. Typedef
8. Unions
9. Bit-fields

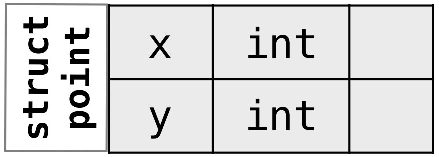
Structure

다른 타입의 변수를 모아서 이름 붙인 구조

structure tag

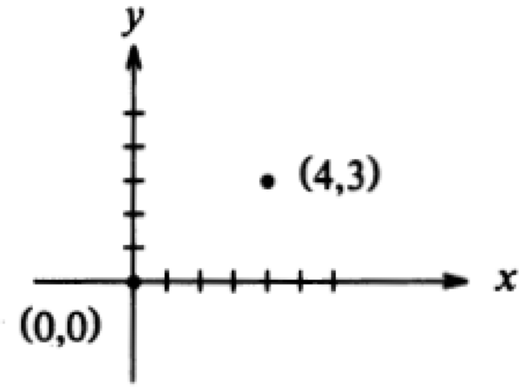
```
struct point {
     int x;
     int y;
};
```

member name



struct point

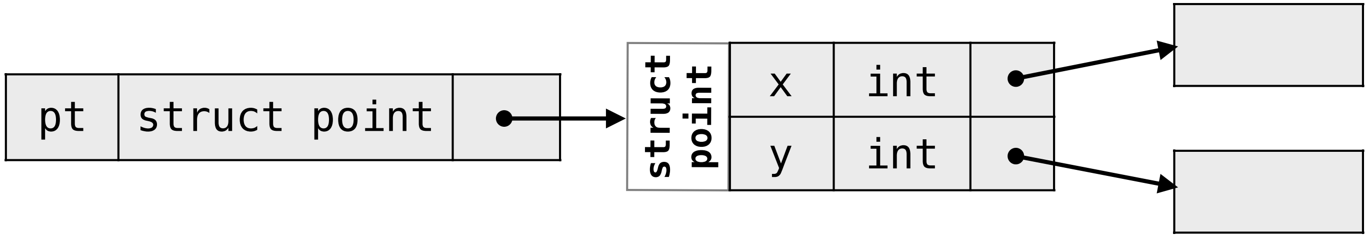
x	int	
y	int	



```
struct point {
     int x;
     int y;
} pt;
```

```
struct point pt;
```

structure name

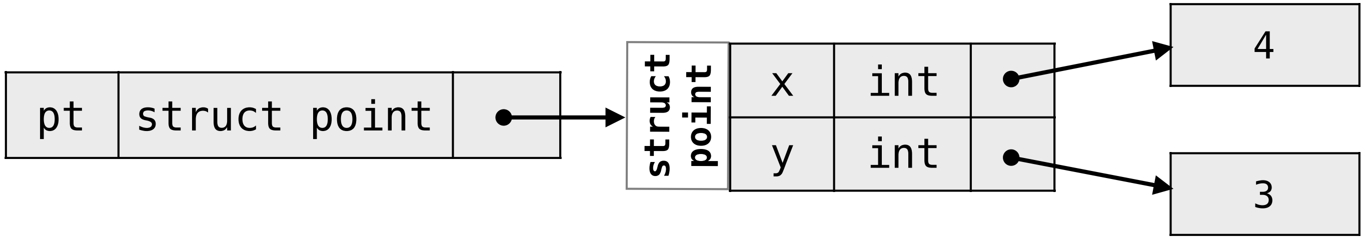


pt | struct point

struct point

x	int	
y	int	

```
struct point pt = { 4, 3 };
```



pt | struct point

struct point

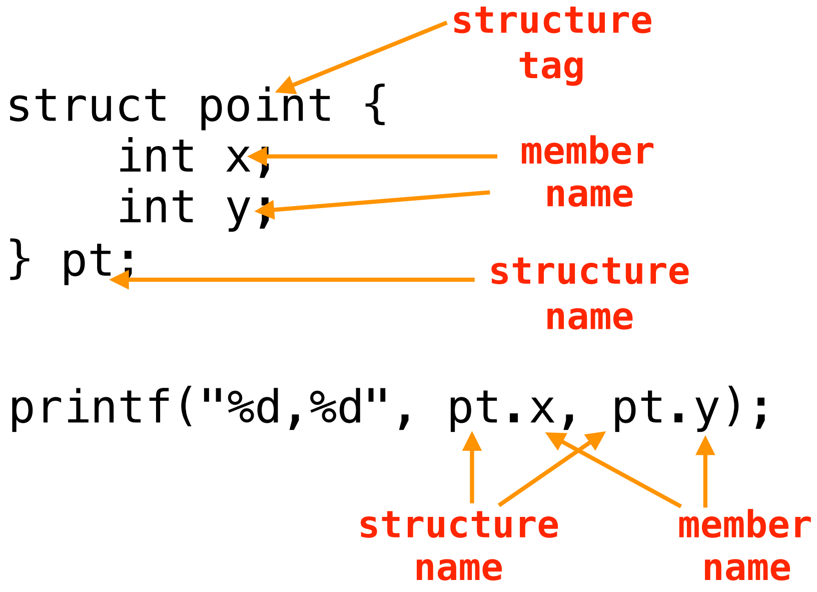
x	int	
y	int	

4

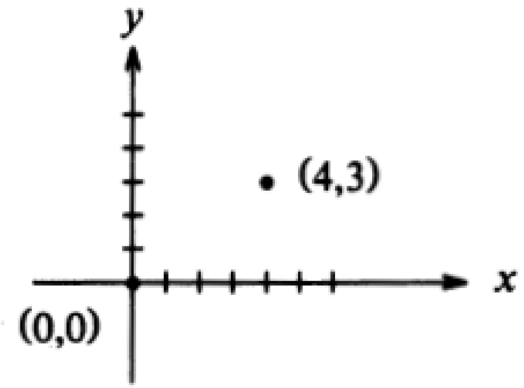
3

Structure

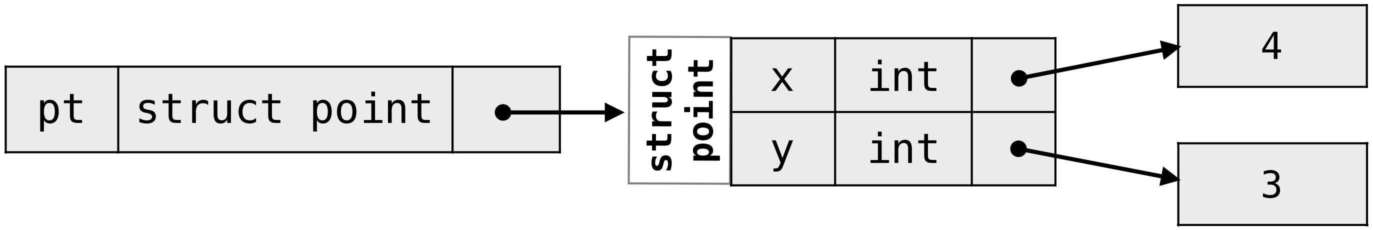
다른 타입의 변수를 모아서 하나로 묶어서 이름 붙인 구조



```
struct point {
     int x;
     int y;
} pt;

printf("%d,%d", pt.x, pt.y);
```



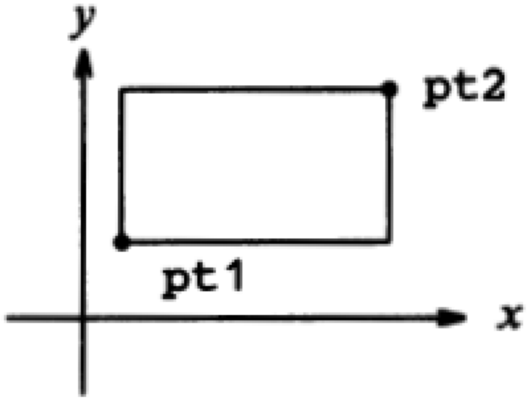
```
double dist, sqrt(double);

dist = sqrt((double)pt.x * pt.x + (double)pt.y * pt.y);
```



Nested Structure

```
struct point {
    int x;
    int y;
};
```

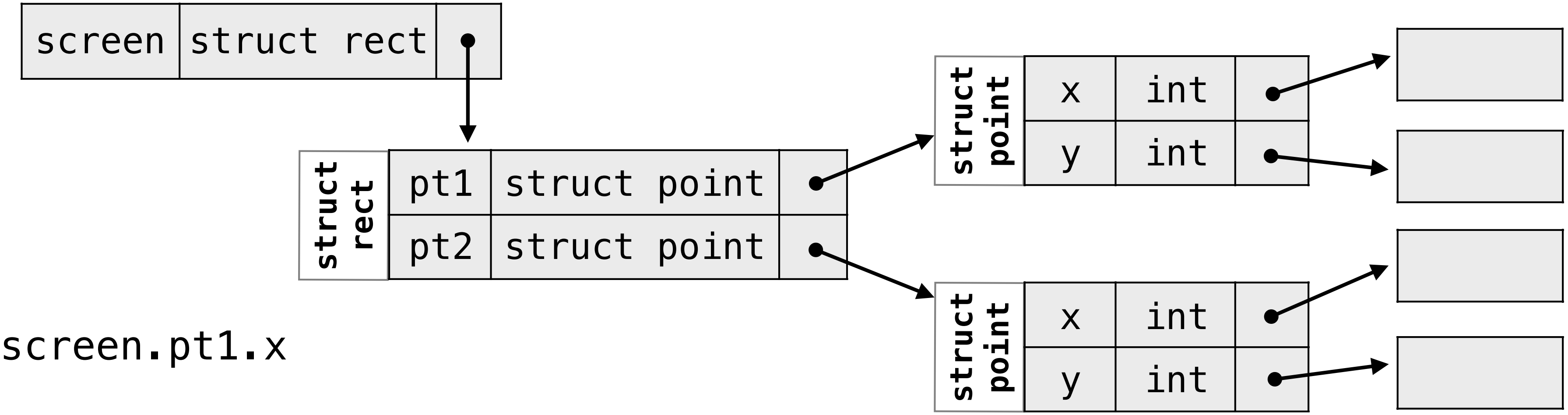
struct point	x	int	
	y	int	



```
struct rect {
    struct point pt1;
    struct point pt2;
};
```

struct rect	pt1	struct point	
	pt2	struct point	

```
struct rect screen;
```



```
screen.pt1.x
```

Operations on structure

- copy or assign as a unit
  - pass arguments to functions
  - return values from functions
- take address with &
- access its members

- 비교 불가

사례 : Functions manipulating points and rectangles

1. pass components separately
2. pass an entire structure
3. pass a pointer to it

```
struct point {
    int x;
    int y;
};
```

```
/* makepoint: make a point from x and y components */
struct point makepoint(int x, int y) {
    struct point temp;

    temp.x = x;
    temp.y = y;
    return temp;
}
```

Operations on structure

- copy or assign as a unit
  - pass arguments to functions
  - return values from functions
- take address with &
- access its members

- 비교 불가

사례 : Functions manipulating points and rectangles

1. pass components separately
2. pass an entire structure
3. pass a pointer to it

```
struct point {
    int x;
    int y;
};
```

```
struct rect {
    struct point pt1;
    struct point pt2;
};
```

```
struct rect screen;
struct point middle;
struct point makepoint(int, int);

screen.pt1 = makepoint(0, 0);
screen.pt2 = makepoint(XMAX, YMAX);
middle = makepoint((screen.pt1.x + screen.pt2.x)/2,
                   (screen.pt1.y + screen.pt2.y)/2);
```

Operations on structure

- copy or assign as a unit
  - pass arguments to functions
  - return values from functions
- take address with &
- access its members

- 비교 불가

사례 : Functions manipulating points and rectangles

1. pass components separately
2. **pass an entire structure**
3. pass a pointer to it

```
struct point {
    int x;
    int y;
};
```

```
struct rect {
    struct point pt1;
    struct point pt2;
};
```

```
/* addpoint: add two points */
struct point addpoint(struct point p1, struct point p2) {
    p1.x += p2.x;
    p1.y += p2.y;
    return p1;
}
```

Operations on structure

- copy or assign as a unit
  - pass arguments to functions
  - return values from functions
- take address with &
- access its members

- 비교 불가

사례 : Functions manipulating points and rectangles

1. pass components separately
2. **pass an entire structure**
3. pass a pointer to it

```
struct point {
    int x;
    int y;
};
```

```
struct rect {
    struct point pt1;
    struct point pt2;
};
```

```
/* ptinrect: return 1 if p in r, 0 if not */
int ptinrect(struct point p, struct rect r) {
    return p.x >= r.pt1.x && p.x < r.pt2.x
        && p.y >= r.pt1.y && p.y < r.pt2.y;
}
```

Operations on structure

- copy or assign as a unit
  - pass arguments to functions
  - return values from functions
- take address with &
- access its members

- 비교 불가

사례 : Functions manipulating points and rectangles

1. pass components separately
2. **pass an entire structure**
3. pass a pointer to it

```
struct point {
    int x;
    int y;
};
```

```
struct rect {
    struct point pt1;
    struct point pt2;
};
```

```
#define min(a, b) ((a) < (b) ? (a) : (b))
#define max(a, b) ((a) > (b) ? (a) : (b))

/* canonrect: canonicalize coordinates of rectangle */
struct rect canonrect(struct rect r) {
    struct rect temp;

    temp.pt1.x = min(r.pt1.x, r.pt2.x);
    temp.pt1.y = min(r.pt1.y, r.pt2.y);
    temp.pt2.x = max(r.pt1.x, r.pt2.x);
    temp.pt2.y = max(r.pt1.y, r.pt2.y);
    return temp;
}
```

Operations on structure

- copy or assign as a unit
  - pass arguments to functions
  - return values from functions
- take address with &
- access its members

- 비교 불가

사례 : Functions manipulating points and rectangles

1. pass components separately
2. pass an entire structure
3. **pass a pointer to it**

```
struct point {
    int x;
    int y;
};
```

```
struct rect {
    struct point pt1;
    struct point pt2;
};
```

```
struct point origin;
struct point *pp;

pp = &origin;
printf("(%d,%d)\n”, (*pp).x, (*pp).y);
printf("(%d,%d)\n”, pp->x, pp->y);
```

```
struct rect r;
struct rect *rp = &r;

r.pt1.x                (r.pt1).x
rp->pt1.x              (rp->pt1).x
```

```
struct {
    int len;
    char *str;
} *p;

++p->len               ++(p->len)

*p->str                *(p->str)
*p->str++              (*p->str)++
*p++->str
```

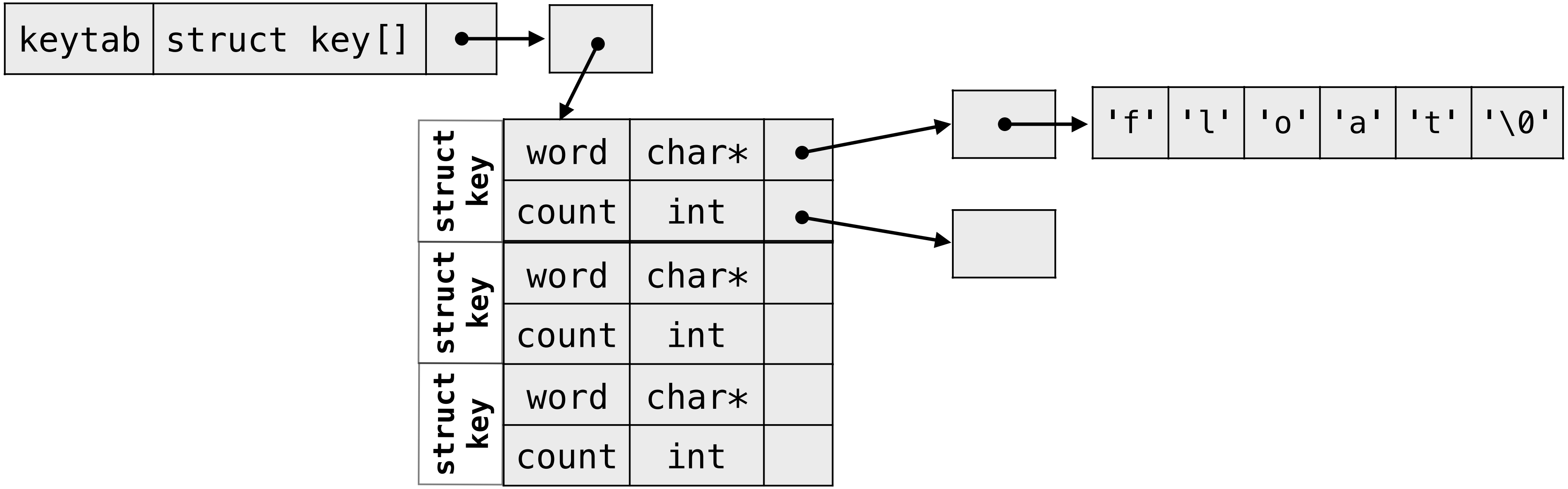
Arrays of Structures

사례 : C 프로그램에서 키워드 빈도수를 세는 프로그램

```
char *keyword[NKEYS];
int keycount[NKEYS];

char *word;
int count;
```

```
struct key {
    char *word;
    int count;
} keytab[NKEYS];
```

```
struct key {
    char *word;
    int count;
};

struct key keytab[NKEYS];
```



keytab struct key[]

struct key: word char*, count int

struct key: word char*, count int

struct key: word char*, count int

'f' 'l' 'o' 'a' 't' '\0'

Arrays of Structures

사례 : C 프로그램에서 키워드 빈도수를 세는 프로그램

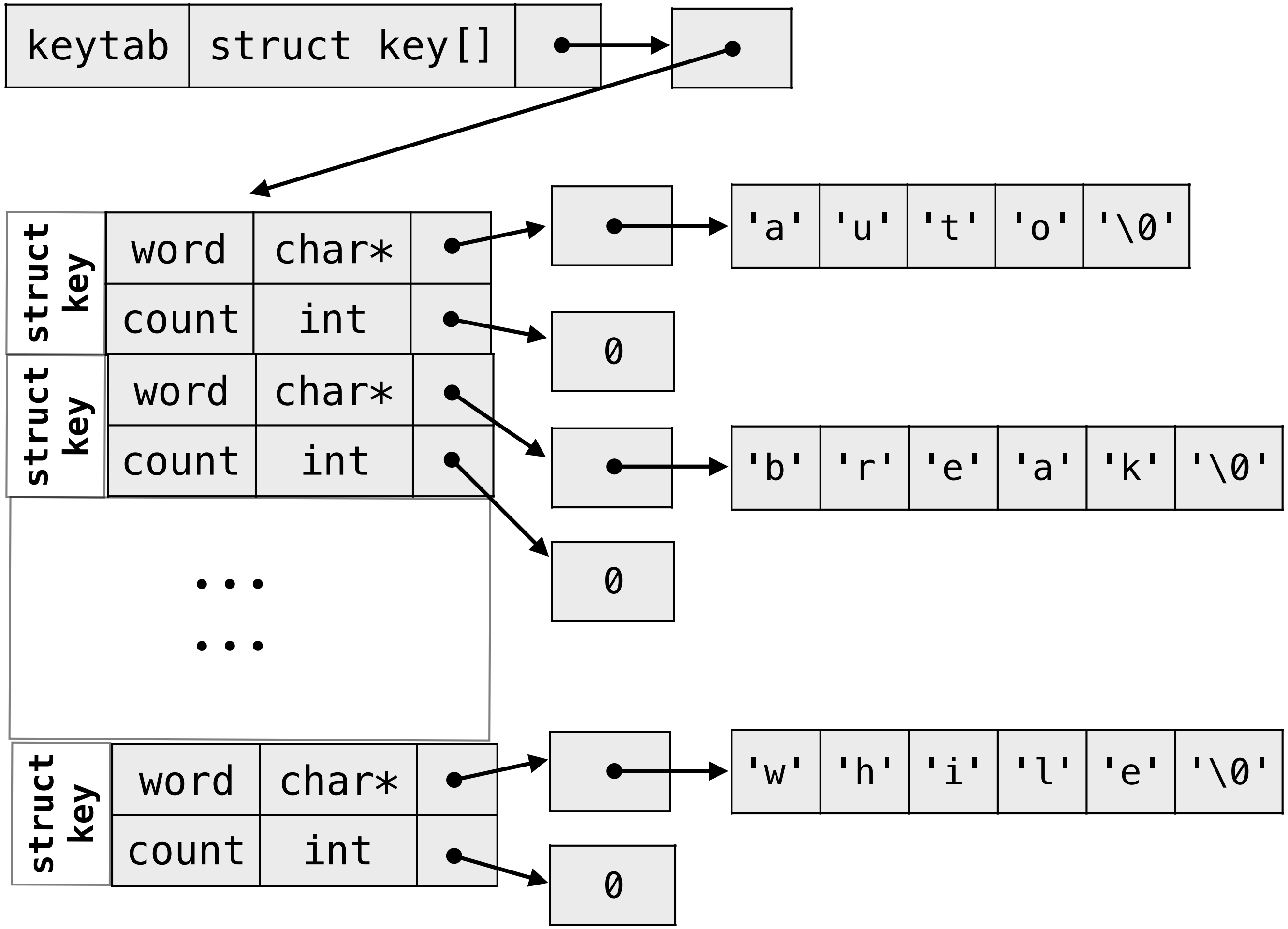
```
struct key {
    char *word;
    int count;
} keytab[] = {
    "auto", 0,
    "break", 0,
    "case", 0,
    "char", 0,
    "const", 0,
    "continue", 0,
    "default", 0,
    /* ... */
    "unsigned", 0,
    "void", 0,
    "volatile", 0,
    "while", 0
}
```

```
struct key {
    char *word;
    int count;
} keytab[] = {
    { "auto", 0 },
    { "break", 0 },
    { "case", 0 },
    { "char", 0 },
    { "const", 0 },
    { "continue", 0 },
    { "default", 0 },
    /* ... */
    { "unsigned", 0 },
    { "void", 0 },
    { "volatile", 0 },
    { "while", 0 }
}
```

Arrays of Structures

사례 : C 프로그램에서 키워드 빈도수를 세는 프로그램

```
struct key {
    char *word;
    int count;
} keytab[] = {
    "auto", 0,
    "break", 0,
    "case", 0,
    "char", 0,
    "const", 0,
    "continue", 0,
    "default", 0,
    /* ... */
    "unsigned", 0,
    "void", 0,
    "volatile", 0,
    "while", 0
}
```



Arrays of Structures

사례 : C 프로그램에서 키워드 빈도수를 세는 프로그램

```
#define MAXWORD 100

int getword(char *, int);
int binsearch(char *, struct key *, int);

/* count C keywords */
main() {
    int n;
    char word[MAXWORD];

    while (getword(word, MAXWORD) != EOF)
        if (isalpha(word[0]))
            if ((n = binsearch(word, keytab, NKEYS)) >= 0)
                keytab[n].count++;
    for (n = 0; n < NKEYS; n++)
        if (keytab[n].count > 0)
            printf("%4d %s\n", keytab[n].count, keytab[n].word);
    return 0;
}
```

```
#define NKEYS (sizeof keytab / sizeof (struct key))
```

```
#define NKEYS (sizeof keytab / sizeof keytab[0])
```

sizeof

```
printf("char = %lu\n", sizeof(char));
printf("short = %lu\n", sizeof(short));
printf("int = %lu\n", sizeof(int));
printf("long = %lu\n", sizeof(long));
printf("float = %lu\n", sizeof(float));
printf("double = %lu\n", sizeof(double));
printf("char* = %lu\n", sizeof(char*));
printf("int* = %lu\n", sizeof(int*));
```

```
struct point {
    int x;
    int y;
};

printf("struct point = %lu\n”, sizeof(struct point));
```

```
int x;
int a[10];
struct key {
    char *word;
    int count;
} keytab[100];

printf("x = %lu\n", sizeof(x));
printf("a = %lu\n", sizeof(a));
printf("struct key = %lu\n", sizeof(struct key));
printf("keytab = %lu\n", sizeof(keytab));
```

```
printf("size_t = %lu\n", sizeof(size_t));
```

Arrays of Structures

사례 : C 프로그램에서 키워드 빈도수를 세는 프로그램

```
/* binsearch: find word in tab[0]...tab[n-1] */
int binsearch(char *word, struct key tab[], int n) {
    int cond;
    int low, high, mid;

    low = 0;
    high = n - 1;
    while (low <= high) {
        mid = (low + high) / 2;
        if ((cond = strcmp(word, tab[mid].word)) < 0)
            high = mid - 1;
        else if (cond > 0)
            low = mid + 1;
        else
            return mid;
    }
    return -1;
}
```

Arrays of Structures

사례 : C 프로그램에서 키워드 빈도수를 세는 프로그램

```
/* getword: get next word or character from input */
int getword(char *word, int lim) {
    int c, getch(void);
    void ungetch(int);
    char *w = word;

    while (isspace(c = getch()))
        ;
    if (c != EOF)
        *w++ = c;
    if (!isalpha(c)) {
        *w = '\0';
        return c;
    }
    for ( ; --lim > 0; w++)
        if (!isalnum(*w = getch())) {
            ungetch(*w);
            break;
        }
    *w = '\0';
    return word[0];
}
```

Arrays of Structures

사례 : C 프로그램에서 키워드 빈도수를 세는 프로그램

```
#define BUFSIZE  100

static char buf[BUFSIZE];   /* buffer for ungetch */
static int bufp = 0;        /* bext free position in buf */

int getch(void) {    /* get a (possibly pushed back) character */
    return (bufp > 0) ? buf[--bufp] : getchar();
}

void ungetch(int c) {   /* push character back on input */
    if (bufp >= BUFSIZE)
        printf("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}
```

Pointers to Structures

사례 : C 프로그램에서 키워드 빈도수를 세는 프로그램

pointer-to-structure 버전으로 바꾸어보자!

Self-referential Structures

사례 : C 프로그램에서 키워드 빈도수를 세는 프로그램

이분검색나무 (Binary Search Tree)

“now is the time for all good men to come to the aid of their party”

마디
node

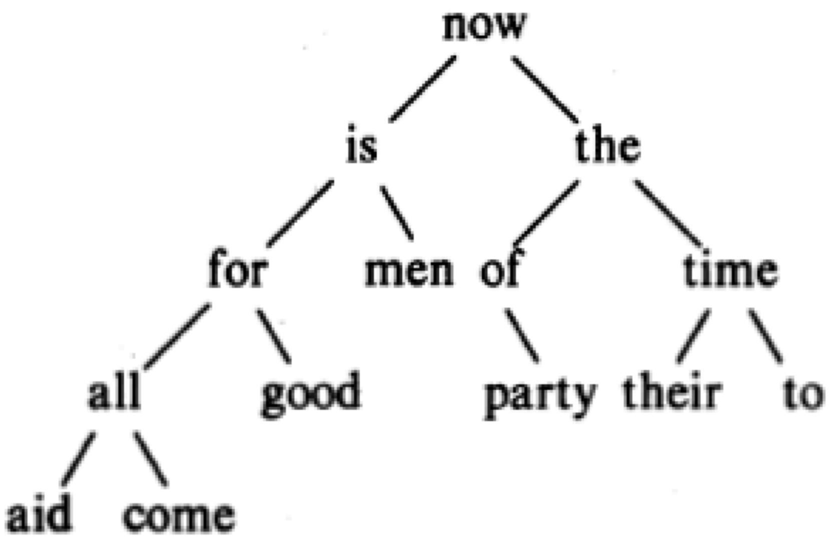
줄기
branch

왼자식마디
left child node

오른자식마디
right child node

왼아래나무
left subtree

오른아래나무
right subtree



- 노드의 왼아래나무에 있는 단어는 모두 그 노드의 단어보다 사전순으로 작고,
- 노드의 오른아래나무에 있는 단어는 모두 그 노드의 단어보다 사전순으로 크다.

Self-referential Structures

사례 : C 프로그램에서 키워드 빈도수를 세는 프로그램

이분검색나무 (Binary Search Tree)

```
struct tnode {
    char *word;
    int count;
    struct tnode *left;
    struct tnode *right;
}
```

struct tnode	word	char *	
	count	int	
	left	struct tnode *	
	right	struct tnode *	

```
struct tnode *addtree(struct tnode *, char *);
void treeprint(struct tnode *);
int getword(char *, int);

/* word frequency count */
int main() {
    struct tnode *root;
    char word[MAXWORD];

    root = NULL;
    while (getword(word, MAXWORD) != EOF)
        if (isalpha(word[0]))
            root = addtree(root, word);
    treeprint(root);
    return 0;
}
```

Self-referential Structures

사례 : C 프로그램에서 키워드 빈도수를 세는 프로그램

```
struct tnode *talloc(void);
char *strdup(char *);

/* addtree: add a node with w, at or below p */
struct tnode *addtree(struct tnode *p, char *w) {
    int cond;

    if (p == NULL) {
        p = talloc();
        p->word = strdup(w);
        p->count = 1;
        p->left = p->right = NULL;
    }
    else if ((cond = strcmp(w, p->word)) == 0)
        p->count++;
    else if (cond <0)
        p->left = addtree(p->left, w);
    else
        p->right = addtree(p->right, w);
    return p;
}
```

Self-referential Structures

사례 : C 프로그램에서 키워드 빈도수를 세는 프로그램

```c
/* treeprint: in-order print of tree p */
void treeprint(struct tnode *p) {
    if (p != NULL) {
        treeprint(p->left);
        printf("%4d %s\n", p->count, p->word);
        treeprint(p->right);
    }
}
```

```c
#include <stdlib.h>

/* talloc: make a tnode */
struct tnode *talloc(void) {
    return (struct tnode *) malloc(sizeof(struct tnode));
}

/* strdup: make a duplicate of s */
char *strdup(char *s) {
    char *p;

    p = (char *) malloc(strlen(s)+1); /* +1 for '\0' */
    if (p != NULL)
        strcpy(p, s);
    return p;
}
```