
C 프로그래밍 및 실습

12. 동적 메모리 할당

세종대학교

목차

- 1) 동적 할당 개요
- 2) 동적 메모리 사용 절차
- 3) 동적 메모리 사용 예제
- 4) 기타 동적 메모리 할당 함수

1) 동적 할당 개요

▪ 정적 메모리 할당

- 지금까지 기억 장소 확보를 위해 사용한 방식은 **변수 선언**
- 변수는 모두 코드 작성 단계에서 정해지고, 할당되는 메모리 크기는 프로그램을 실행할 때 마다 일정

<code>char ch;</code>	⇒ 1 바이트 (일반 변수)
<code>int score[10];</code>	⇒ 40 바이트 (배열 변수)
<code>struct student {int id; float avg;} st1;</code>	⇒ 8 바이트 (구조체 변수)
<code>int *pi;</code>	⇒ 4 바이트 (포인터 변수)

프로그램 **실행 전에** 변수의 메모리 할당 크기가 정해지고 실행 도중 메모리 할당 크기가 변하지 않는 방식 → **정적 할당**

1) 동적 할당 개요

▪ 정적 할당 방식의 문제점

- 필요한 메모리 크기를 알 수 없는 경우, 충분히 큰 크기를 가정해서 선언해야 함
- 프로그램 실행 전, 메모리를 얼마나 필요로 할 것인지 정확하게 알 수 없는 경우, 메모리를 효율적으로 사용하기 어려움
- 어떤 회사의 웹 사이트에서 가입하는 회원의 ID를 저장해야 하는 배열을 선언해야 한다면???
 - ✓ 몇 명의 사람들이 회원가입을 할 지 정확하게 알 수 없음
→ 충분히 많은 사람들이 회원가입을 할 것으로 가정
 - ✓ 예) `int id[50000];`
 - ✓ 회원 수가 1,000 명밖에 안되는 경우 → 메모리 공간 장비
 - ✓ 회원 수가 5만명을 넘는 경우 → 프로그램에서 배열 크기 조정 후 재 컴파일 필요

1) 동적 할당 개요

- 동적 (dynamic) 메모리 할당

- 프로그램을 **실행하는 동안** 결정되는 크기에 따라 메모리를 할당하는 방식
- 메모리를 효율적으로 사용할 수 있음

- 앞 웹사이트 상황에서, 크기가 5만인 메모리를 할당 받아 사용하다,

- ✓ 회원수가 5만명을 넘어가면 → 10만명의 회원 정보를 저장할 수 있는 메모리를 새로이 할당 받아 사용
- ✓ 회원이 1,000명으로 줄어들면 → 1,000명 정도의 회원 정보를 저장할 수 있는 메모리 새로이 할당 받아 사용
- ✓ 기존 크기가 5만인 메모리는 시스템에 반납

목차

- 1) 동적 할당 개요
- 2) 동적 메모리 사용 절차
- 3) 동적 메모리 사용 예제
- 4) 기타 동적 메모리 할당 함수

2) 동적 메모리 사용 절차

■ 메모리 사용 절차

- 메모리 사용을 위해서는 메모리를 **할당**하고 **해제**하는 과정 필요
 - ✓ **할당**: 필요한 메모리를 시스템으로부터 받기
 - ✓ **사용**: 할당된 메모리 사용
 - ✓ **해제**: 사용이 끝난 메모리를 시스템에 반납
- 정적 할당 vs. 동적 할당
 - ✓ **정적 할당**: 이 과정이 프로그램과 함수의 실행 및 종료에 따라 자동으로 수행됨
 - ✓ **동적 할당**: 프로그래머가 필요한 과정을 코드에 명시적으로 작성해 주어야 함

2) 동적 메모리 사용 절차

• 동적 할당 기본 예제

- 동적으로 메모리를 할당하고 해제하는 라이브러리 함수는 `<stdlib.h>` 헤더 파일에 선언

```
#include <stdlib.h>    // 동적 메모리 관련 함수 사용을
                        // 위한 헤더

int main() {
    int *p = NULL;      // 동적 메모리 접근을 위한 포인터 변수
    p = (int *) malloc( 5*sizeof(int) ); // 동적 메모리 할당

    p[0] = 1;           // 동적 메모리 사용: 배열 형태
    *(p+2) = 3;         // 동적 메모리 사용: 포인터 형태

    free(p);            // 동적 메모리 해제
    return 0;
}
```


2) 동적 메모리 사용 절차

- 동적 메모리 할당: malloc() 함수
 - 요청된 크기의 연속된 메모리 할당(memory allocation)

함수 원형	void * malloc(unsigned int size);
함수 인자	✓ 할당 받을 메모리 크기(바이트 단위) ✓ 보통 sizeof() 연산자 활용
기능 및 반환 값	✓ 할당 받은 메모리의 시작 주소를 반환 ✓ 할당에 실패하면 → NULL 반환 ✓ 반환 값의 유형: void형 포인터 (void *)

- ✓ **void ***: 특정 자료형을 나타내지 않는 주소를 의미(13.6절에서 학습)
- ✓ 주의!! **void** (반환 값이 없음을 의미)와 혼동 하지 말자

2) 동적 메모리 사용 절차

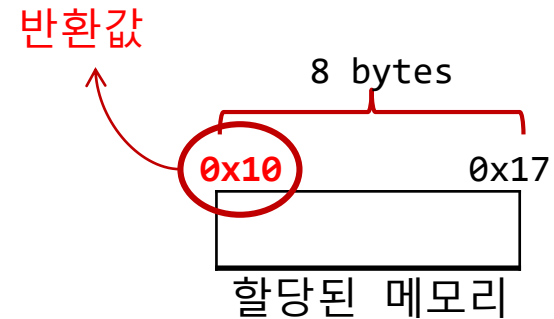
- malloc() 함수 사용 예

- 정수 2개를 저장할 공간 할당

```
malloc( 2*sizeof(int) );
```



```
malloc( 8 ) ; ⇒ 8 바이트 할당
```

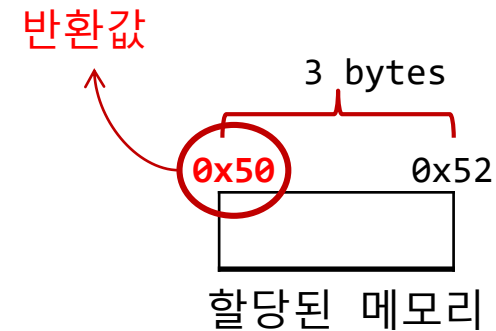


- 문자 3개를 저장할 공간 할당

```
malloc( 3*sizeof(char) );
```



```
malloc( 3 ) ; ⇒ 3 바이트 할당
```



2) 동적 메모리 사용 절차

• 동적 메모리 연결

- 변수 이름을 붙일 수 없기 때문에, **포인터 변수**를 이용하여 간접적으로 참조

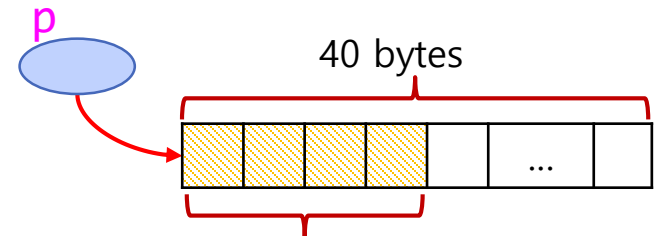
```
자료형 * 포인터_변수명 = NULL;  
포인터_변수명 = (자료형 *) malloc(메모리_크기);
```

사용하고자 하는 자료형으로
명시적 형변환

할당된 메모리를 어떤 자료형으로
사용할 지 모르므로 (void *)형 반환

• 예시)

```
int *p = NULL;  
p = (int *) malloc( 10*sizeof(int) );
```



int 형으로의 강제 형변환
→ 4bytes씩 끊어서 사용

2) 동적 메모리 사용 절차

- 동적 메모리 사용

- 할당된 메모리를 포인터 변수에 연결한 후에는 9장(포인터)에서 배운대로 사용
- 예시) 크기가 10인 배열 처럼 사용

```
int *p = (int *) malloc( 10*sizeof(int) );  
                                // 변수 선언과 동시에  
                                // 동적 할당된 주소 대입  
  
p[0] = 1;                      // *p = 1; 과 동일  
*(p+2) = 3;                    // p[2] = 3; 과 동일
```

2) 동적 메모리 사용 절차

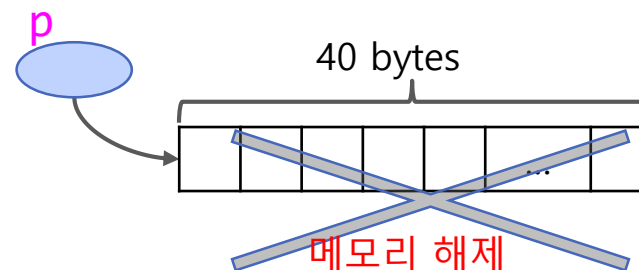
- 동적 메모리 해제: `free()` 함수
 - 지정된 메모리 공간을 해제(`free`)

함수 원형	<code>void free(void * ptr);</code>
함수 인자	해제할 메모리 공간을 가리키는 포인터 변수
기능	<code>ptr</code> 이 가리키는 메모리 해제

» 주의!! `ptr` 변수 자체를 해제시키는 것 아님

- 예시

```
int *p = NULL;  
p = (int *) malloc( 10*sizeof(int) );  
...  
free( p ); // p가 가리키는 영역 해제
```



2) 동적 메모리 사용 절차

- [예제 12.1] 다음 지시대로 동적 메모리 할당 및 해제하는 코드를 작성해보자.
 1. 정수 한 개를 저장할 수 있는 메모리 공간 할당하고 해제하기
 2. float형 실수 한 개를 저장할 수 있는 메모리 공간 할당하고 해제하기
 3. double형 실수 15개를 저장할 수 있는 메모리 공간 할당하고 해제하기
 4. 다음과 같은 구조체 한 개를 저장할 수 있는 메모리 공간 할당하고 해제하기

```
struct student{  
    int id;    char name[8];    double grade;  
};
```

2) 동적 메모리 사용 절차

- 동적 메모리를 안전하게 사용하기 위한 주의사항
 - malloc()의 반환 값을 검사하여 **메모리 할당의 성공 여부 확인**
 - ✓ NULL 포인터 반환하는 경우: 요청한 메모리 크기만큼 연속된 메모리 공간을 확보하지 못할 때
 - 비슷하게, 해제하려는 메모리 주소가 NULL인지 검사

```
int *p = NULL;
p = (int *) malloc( 10*sizeof(int) );

if( p == NULL ) {                               // 메모리 할당 실패하면,
    printf("Not enough memory!");                // 오류 상황 알리고,
    return -1;                                    // 함수 종료
}
...
if( p != NULL )                                  // p가 NULL이 아닌 경우에만
    free(p);                                       // free() 함수 호출
```

2) 동적 메모리 사용 절차

■ 메모리 할당 방식 비교

	정적 메모리 할당	동적 메모리 할당
필요한 메모리 크기 결정시점	프로그램 작성 단계	프로그램 실행 중
메모리 할당 및 해제 시점	시스템이 자동으로 할당 및 해제	개발자가 명시적으로 할당 및 해제 함수 호출
프로그램 실행 중 메모리 크기 변경 여부	불가	가능
메모리 누수(leak) 가능성	없음	있음
메모리 사용의 효율성	비효율적	효율적

목차

- 1) 동적 할당 개요
- 2) 동적 메모리 사용 절차
- 3) 동적 메모리 사용 예제
- 4) 기타 동적 메모리 할당 함수

3) 동적 메모리 할당 프로그램 예제

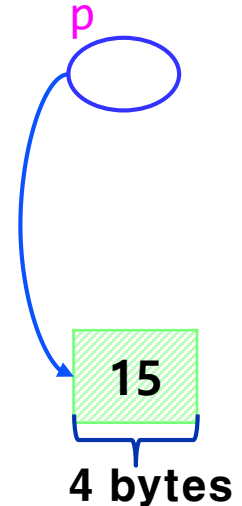
1) 동적으로 메모리를 할당 받아 사용하는 기본 프로그램

- 하나의 정수를 저장하는 메모리를 동적 할당 받아 사용
✓ 실용성은 없음

```
#include <stdio.h>
#include <stdlib.h> // 동적 할당 함수가 선언된 헤더 파일

int main() {
    int *p = NULL;           // 포인터 변수 선언
    p = (int *)malloc(sizeof(int)); // 동적 할당
    if (p == NULL) {         // 동적 할당 오류 검사
        printf("Not enough memory!");
        return -1;
    }

    *p = 15;                 // 동적 할당 메모리에 값 저장
    printf("동적 메모리 할당 (정수형): %d", *p);
    free(p);
    return 0;
}
```



3) 동적 메모리 할당 프로그램 예제

2) 일차원 배열을 동적 할당 받아 사용하는 프로그램

- 여러 개의 정수를 저장하는 메모리를 동적 할당 받아 사용
- 문제: 한 학생의 과목별 점수를 입력 받아 평균 점수를 구한 후 출력하는 프로그램을 작성하시오.
 - ① 프로그램 실행 중에 사용자로부터 과목 수 n 을 직접 입력 받음
 - ② n 개의 정수형 값을 저장할 수 있는 메모리를 동적으로 할당 받아, 할당 받은 메모리의 시작 주소를 `score`에 저장
 - ③ 할당 받은 메모리를 `score[0]~score[n-1]`까지 일반 배열 원소처럼 참조하여 사용
 - ④ 할당 받은 메모리 해제

3) 동적 메모리 할당 프로그램 예제

```
int n, i, sum = 0;
int *score = NULL;

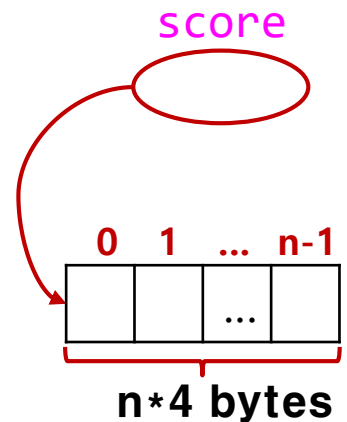
scanf("%d", &n);           // ① 과목 수 입력받기

score = (int *) malloc( n*sizeof(int) ); // ② 배열 동적 할당
if (score == NULL) {       // 동적 할당 오류 검사
    ...
}

for (i=0; i<n; i++) {
    scanf("%d", &score[i]); // ③ 과목별 점수 입력받기
    sum += score[i];         // 누적 합 계산
}

printf("%.1f", (double)sum/n); // 평균 계산

free(score);                // ④ 동적 할당 메모리 해제
return 0;
```



3) 동적 메모리 할당 프로그램 예제

3) 동적 메모리 할당을 사용한 문자열 처리 프로그램

- 길이가 다른 여러 개의 문자열을 효율적으로 저장하기 위해
- 다음과 같은 방식으로 동적 메모리 할당 이용
 - ① 문자열을 입력 받기 위한 충분한 크기의 문자 배열을 선언 (정적 할당)
 - ② ①의 문자 배열에 문자열을 입력 받음
 - ③ ②에서 입력된 문자열의 길이를 계산하여 그 크기에 맞게 메모리를 동적으로 할당 받음 (널 문자 포함)
 - ④ 할당 받은 메모리 공간에 ①의 문자 배열의 문자열을 복사함

3) 동적 메모리 할당 프로그램 예제

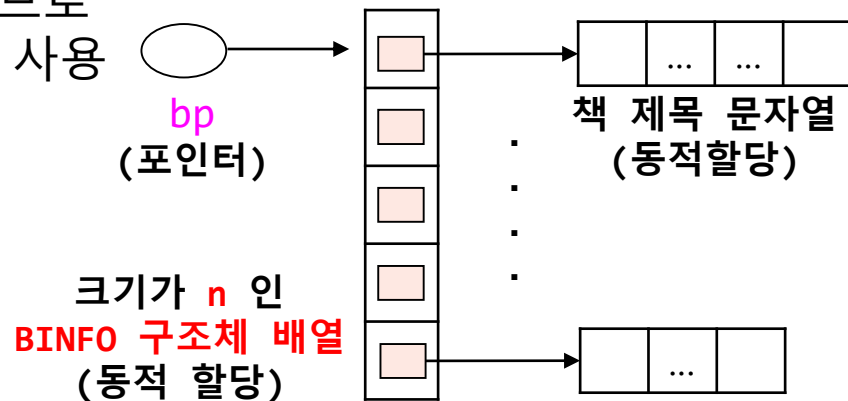
- 문제: 사용자로부터 **책의 개수 n**을 입력 받아, n개의 책 제목을 저장하는 프로그램을 작성하시오.

- ✓ 사용할 메모리 구조

- ✓ 책 하나의 정보를 저장하는 구조체

```
typedef struct book_title {  
    char *title; // 책 제목을 가리키는 포인터  
    ...        // 기타 책 정보(필요 시)  
} BINFO;
```

- ✓ 책의 개수를 모르므로
동적 구조체 배열 사용



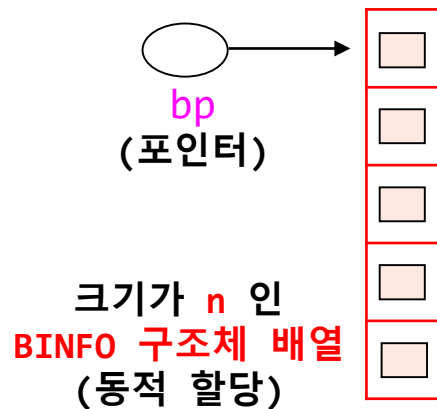
3) 동적 메모리 할당 프로그램 예제

```
BINFO *bp = NULL;
int n, i, len;
char temp[100];           // ① 문자열을 입력 받기 위한 문자 배열을 선언

scanf("%d", &n);           // 책 개수 입력
getchar();                 // 개행 문자 버리기

bp = (BINFO *)malloc( n*sizeof(BINFO) ); // 책 배열 동적 할당
if (bp == NULL) { ... }    // 동적 할당 오류 검사 (생략)

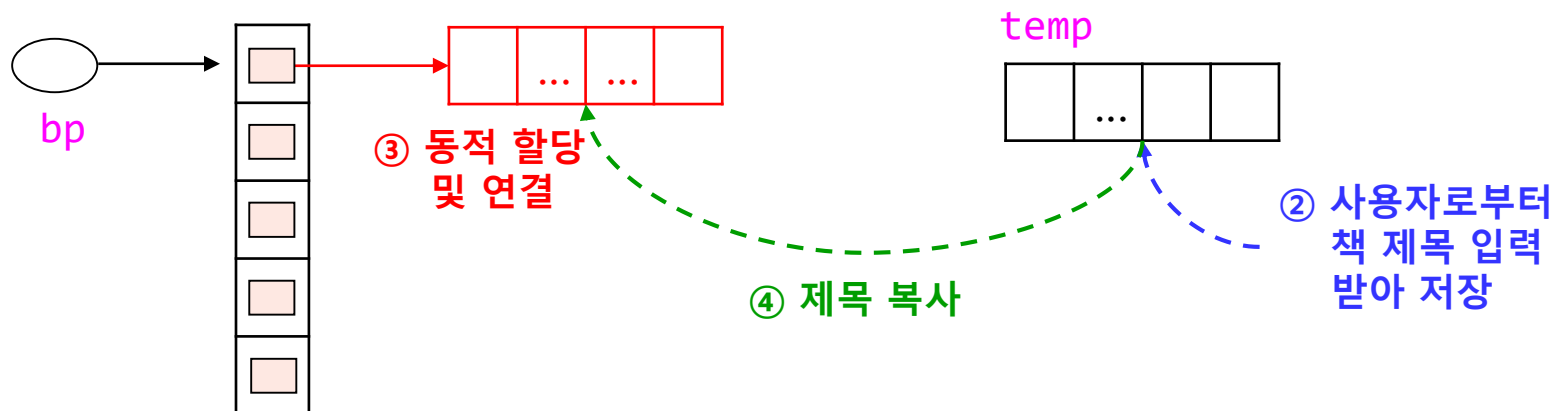
(다음 슬라이드에 계속)
```



3) 동적 메모리 할당 프로그램 예제

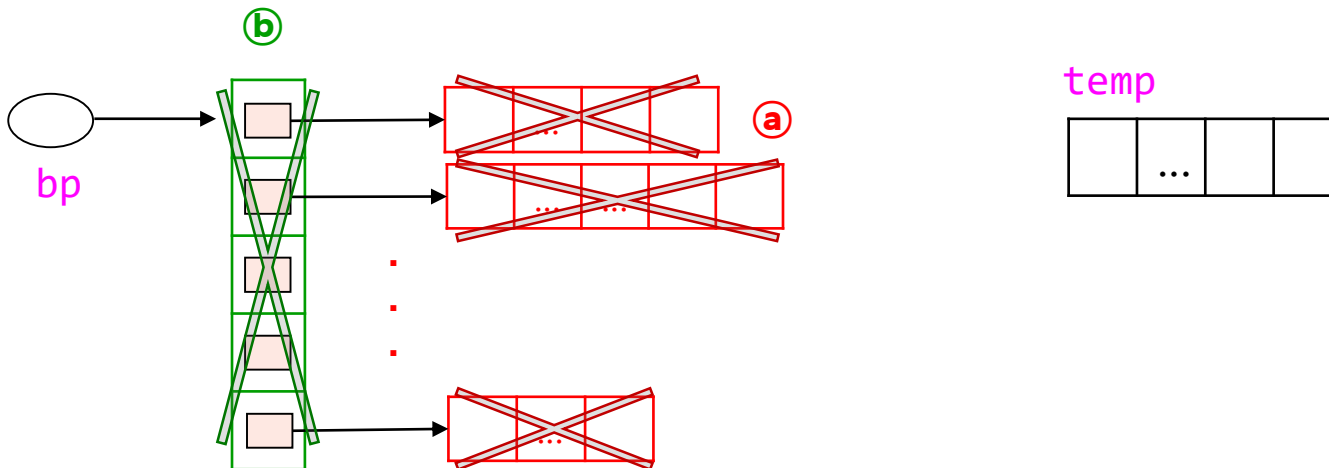
```
for (i=0; i<n; i++) {  
    gets(temp);           // ② 문자 배열 temp에 문자열을 입력 받음  
  
    len = strlen(temp); // ③ temp에 입력된 문자열 길이를 계산하고,  
    bp[i].title = (char*) malloc( (len+1)*sizeof(char) );  
                        // ③ 그 크기만큼 메모리를 동적으로 할당 받음 (널 문자 포함)  
    if (bp[i].title == NULL) { ... } // 동적 할당 오류 검사 (생략)  
  
    strcpy(bp[i].title, temp); // ④ 할당 받은 메모리에 제목 복사  
}
```

(다음 슬라이드에 계속)



3) 동적 메모리 할당 프로그램 예제

```
for (i=0; i<n; i++)  
    printf("%s\n", bp[i].title);           // 책 제목 출력  
  
for (i=0; i<n; i++)  
    free(bp[i].title);                     // ㉠ 책 제목(문자 배열)에 대한 메모리 해제  
  
free(bp);                                  // ㉡ 책 배열(구조체 배열)에 대한 메모리 해제  
  
return 0;
```



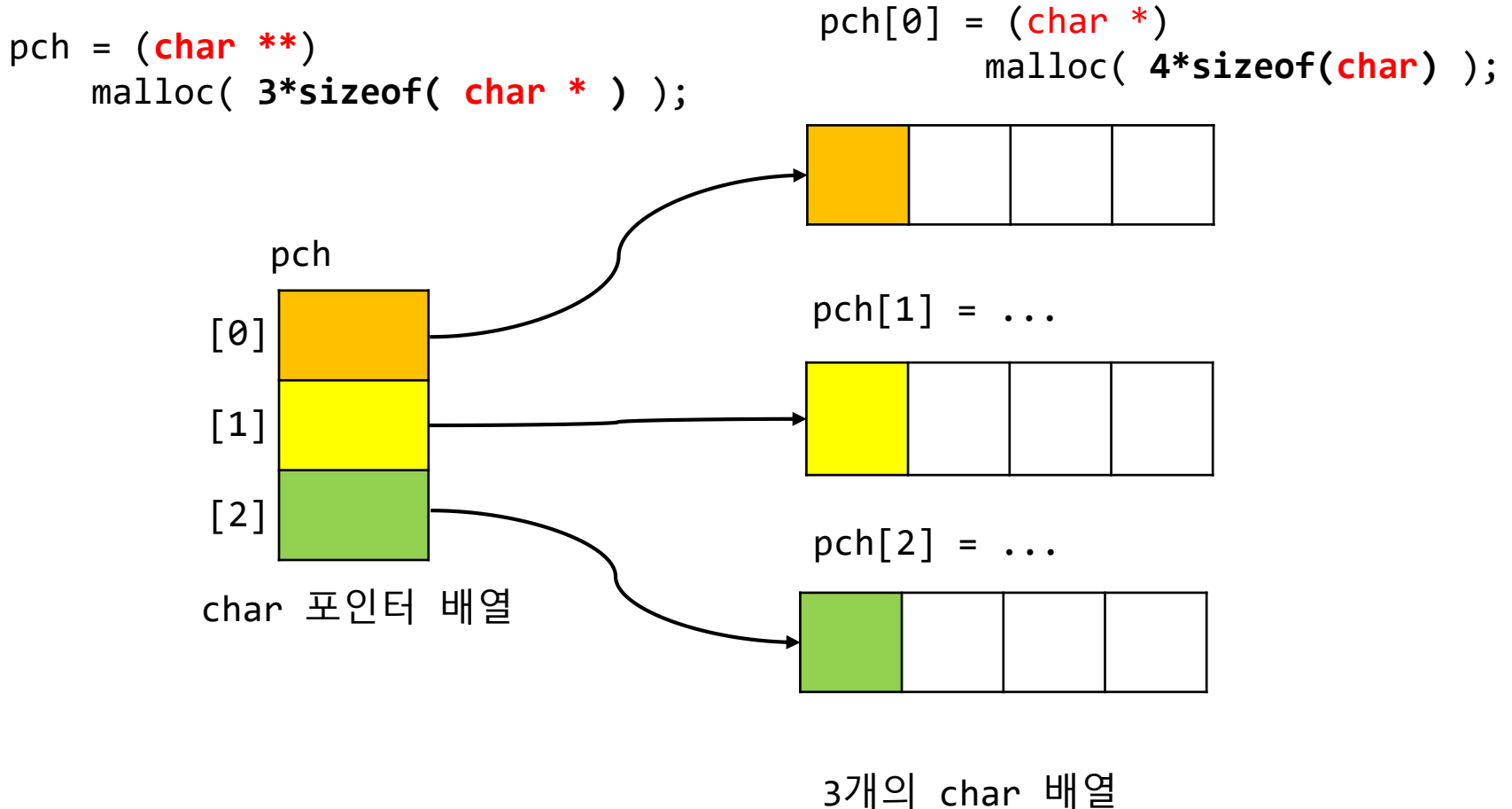
3) 동적 메모리 할당 프로그램 예제

4) 동적 메모리 할당을 사용하여 2차원 배열 할당하는 프로그램

- 문제: `char array[3][4]`라는 2차원 배열을 동적으로 할당하기
- 착각하기 쉬운 잘못된 방법
 - ✓ `char *pch = (char *) malloc(3*4*sizeof(char));`
➔ 사실 상 `malloc(12);` 호출
 - ✓ 2차원 배열 할당이 아니라, 크기가 12인 1차원 배열 할당
- 해결방법: 포인터 배열 사용(다음 슬라이드 참조)
 - ✓ 다른 다차원 도 동일한 원리 적용: 2차원 정수 배열, 3차원 배열 등

3) 동적 메모리 할당 프로그램 예제

- 해결방법: 포인터 배열 사용! (메모리 그림 참조)



3) 동적 메모리 할당 프로그램 예제

```
int i;
char **pch;                                // 이중 포인터 선언
pch = (char **)malloc( 3*sizeof(char *) );  // 포인터 배열 할당

for (i=0; i<3; i++)
    pch[i] = (char *)malloc( 4*sizeof(char) ); // 1차원 배열 할당

strcpy(pch[0], "aaa");
strcpy(pch[1], "bbb");
pch[2][0] = '\\0';                          // 2차원 배열처럼 사용 (예제 9.10 참조)

for (i=0; i<3; i++)
    puts(pch[i]);

for (i=0; i<3; i++)
    free(pch[i]);                            // 1차원 배열에 대한 메모리 해제

free(pch);                                  // 포인터 배열에 대한 메모리 해제

return 0;
```

3) 동적 메모리 할당 프로그램 예제

5) 동적 메모리 할당을 사용한 문자열 처리 프로그램

- 길이가 다른 여러 개의 문자열을 효율적으로 저장하기 위해
- 다음과 같은 방식으로 동적 메모리 할당 이용
 - ① 문자열을 입력 받기 위한 충분한 크기의 문자 배열을 선언 (정적 할당)
 - ② ①의 문자 배열에 문자열을 입력 받음
 - ③ ②에서 입력된 문자열의 길이를 계산하여 그 크기에 맞게 메모리를 동적으로 할당 받음 (널 문자 포함)
 - ④ 할당 받은 메모리 공간에 ①의 문자 배열의 문자열을 복사함

3) 동적 메모리 할당 프로그램 예제

```
int i, N;
char *pch, temp[100];    // 포인터 선언
scanf("%s", temp );      // gets( temp );
N=strlen( temp );
pch = (char *)malloc( (N+1)*sizeof(char) ); // 1차원 배열 할당
if (pch == NULL) { ... } // 동적 할당 오류 검사 (생략)
strcpy(pch, temp);        // ④ 할당 받은 메모리에 문자열 복사
printf("%s", temp );
free(pch);                // 포인터 배열에 대한 메모리 해제

return 0;
```

추가

동적할당 받은 경우 단어 교환은, 주소 교환만 가능

```
char **pa, *t;
```

```
pa = (char **)malloc(2 * sizeof(char *));
```

```
pa[0] = (char *)malloc(3 * sizeof(char));
```

```
pa[1] = (char *)malloc(4 * sizeof(char));
```

```
t=pa[0];
```

```
pa[0]=pa[1];
```

```
pa[1]=t;
```

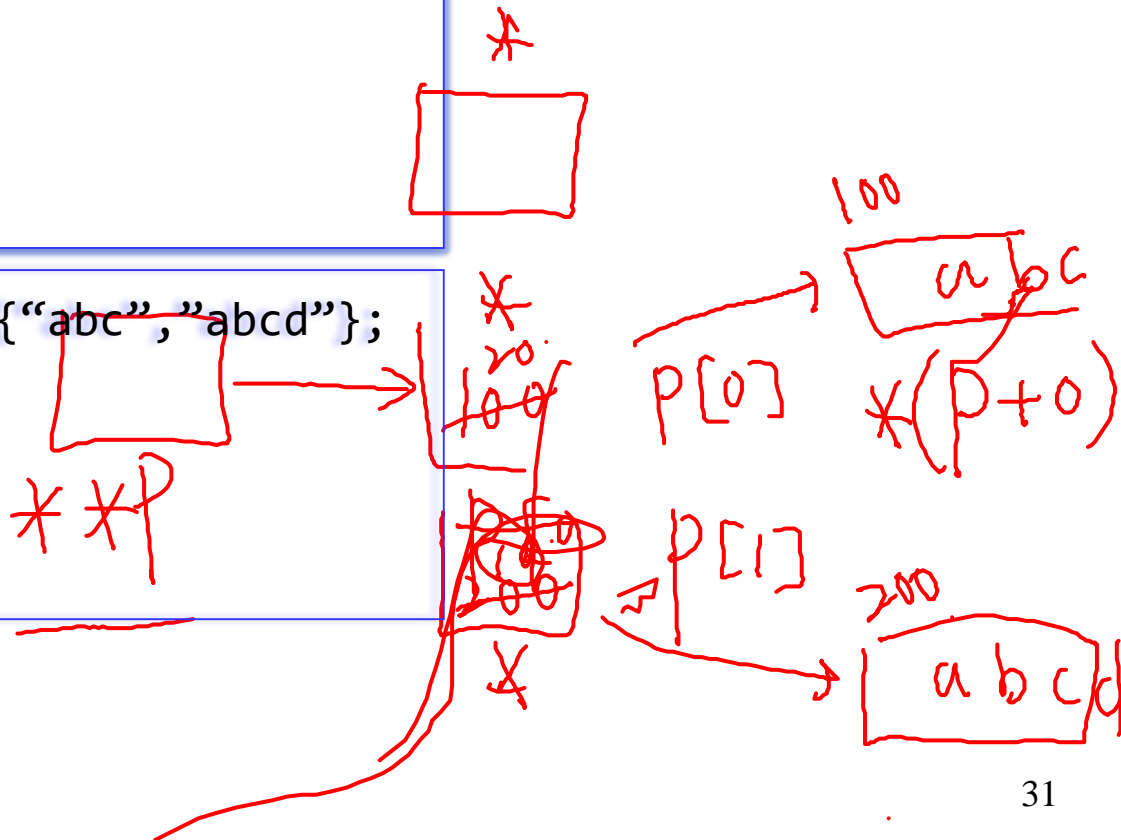
realloc도 가능

```
char tmp[100], ar[10][100]={"abc", "abcd"};
```

```
strcpy(t, ar[0]);
```

```
strcpy(ar[0], ar[1]);
```

```
strcpy(ar[1], tmp);
```



목차

- 1) 동적 할당 개요
- 2) 동적 메모리 사용 절차
- 3) 동적 메모리 사용 예제
- 4) 기타 동적 메모리 할당 함수

4) 기타 동적 메모리 할당 함수

▪ calloc() 함수

함수 원형	void * calloc(unsigned int num, unsigned int size);	
함수 인자	num	동적으로 할당 받을 원소의 개수
	size	원소 한 개의 크기 (바이트 단위)
반환 값	✓ (num*size) 바이트 수 만큼 할당하고, 할당된 메모리를 모두 0으로 초기화 후 시작 위치 반환 ✓ 실패하면 → NULL 반환	

(사용 예)

```
int *p = NULL;
```

```
p = (int *) calloc( 5, sizeof(int) );    ⇨ 크기 5인 int형 배열 할당
```

```
⇨ malloc( 5*sizeof(int) ); 와 유사
```

4) 기타 동적 메모리 할당 함수

▪ realloc() 함수

함수 원형	void * realloc(void *ptr, unsigned int size);	
함수 인자	ptr	확장 크기를 변경할 메모리의 시작 주소
	new_size	변경 후 메모리의 전체 크기 (바이트 단위)
반환 값	✓ ptr이 가리키는 메모리의 크기를 new_size 바이트 크기로 조정하고, 조정된 메모리의 시작 위치 반환 ✓ 실패하면 → NULL 반환	

(사용 예)

```
char *p = (char *) malloc( 5 );    ⇨ 5 바이트 만큼 공간 할당  
p = (char *) realloc( p, 10 );  ⇨ 10 바이트로 공간 크기 변경
```

✓ 주의!! 기존 위치에서 새로운 크기를 확보할 수 없으면,
기존 위치의 공간을 해제하고 **새로운 위치에 공간** 할당

안내

- 시험장 안내
- 동적할당은 필요한 크기만 만들기
- 동적할당 컴파일 오류, M+1 안 하는 경우
- 동적할당 컴파일 오류, 배열 밖에 사용하는 경우

```
char x[5][20] = { "aaa", "bbbb", "ccc" }, (*px)[20];
char **pa;
px = x;
pa = (char **)malloc(5 * sizeof(char *));
pa[0] = x[0]; pa[1] = x[1]; pa[2] = x[2];
printf("%s\n", px++);           printf("%s\n", px);
printf("%s\n", *pa++);          printf("%s\n", *pa);
aaa
bbbb
aaa
bbbb
```

안내

```
int x[5]={1,2,3};  
sum( x )
```

```
int sum(int x[ ]);  
int sum(int *x);
```

```
int x[5][10]={1,2,3};  
sum( x )
```

```
int sum(int x[ ][10]);  
int sum(int (*x)[10]);  
int sum(int **x); // 틀림
```

```
int **pa;  
pa =(int **) malloc(5*sizeof(int *));  
sum( pa )
```

```
int sum(int **pa);
```