

2010-1학기 프로그래밍입문(1)

참고자료 chap 6-8.

---

## 메모리 동적 할당

박종혁

Tel: 970-6702

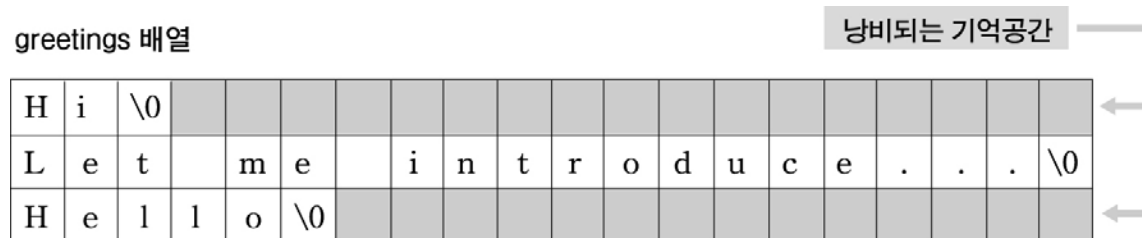
Email: [jhpark1@snut.ac.kr](mailto:jhpark1@snut.ac.kr)

출처: 뇌를 자극하는 C프로그래밍, 한빛미디어

## □ 동적할당의 필요성

- 프로그램을 작성하는 단계에서 필요한 기억공간의 크기를 결정하는 것은 정적할당이다.
  - 변수나 배열의 선언
- 프로그램의 실행 중에 입력되는 데이터에 맞게 기억공간을 확보해야 할 때는 동적할당이 필요하다.
  - 세 명의 인사말을 저장하는 배열은 낭비가 심하다.

```
char greetings[3][20]; // 가장 긴 인사말이 20바이트라고 할 때
```



## ▶ 동적 할당 함수(malloc, free)

- 메모리를 동적으로 할당하기 위해서는 함수를 호출해야 한다.

```
void *malloc(unsigned int); // memory allocation
```

- int형 변수로 사용할 기억공간을 할당 받는 경우

```
int *ip; // 할당 받은 기억공간을 가리킬 포인터변수
```

```
ip = (int *) malloc(4);
```

원하는 바이트 수를 주고 호출

- 기억공간을 할당하고 void 포인터를 리턴한다.
- int형 변수로 활용하기 위해서 형변환 한다.
- int형을 가리키는 포인터 변수에 저장한다.



- 포인터변수로 할당 받은 기억공간을 사용한다.

```
*ip = 10; // ip가 가리키는 기억공간에 10을 저장한다.  
printf(“%d\n”, *ip); // ip가 가리키는 기억공간의 값을 출력한다.
```

## ▶ 메모리를 동적으로 할당 받는 프로그램

```
#include <stdio.h>
#include <stdlib.h>           // malloc함수를 사용하기 위해서 포함시킨다.

int main()
{
    int *ip;                 // int형을 가리킬 포인터변수
    double *dp;             // double형을 가리킬 포인터변수

    ip=(int *)malloc(sizeof(int));           // 기억공간을 동적으로 할당 받아서
    dp=(double *)malloc(sizeof(double));     // 각 포인터 변수에 연결시킨다.

    *ip=10;                       // 포인터변수로 각각 할당 받은 기억공간을
    *dp=3.4;                       // 참조하여 값을 저장한다.

    printf("정수형으로 사용 : %d\n", *ip);   // 포인터변수로 저장된 값을 출력한다.
    printf("실수형으로 사용 : %lf\n", *dp);

    return 0;
}
```

### 출력 결과

```
정수형으로 사용 : 10
실수형으로 사용 : 3.400000
```

## ▶ 동적 할당되는 메모리는 반납해야 한다.

- 동적할당 메모리는 기억부류(storage class)의 힙(heap)영역에 할당된다.



- 힙(heap)영역은 프로그램이 종료될 때까지 기억공간이 유지된다. 따라서 사용이 끝난 동적할당 메모리는 명시적으로 반납한다.

```
void free(void *); // 동적할당 된 기억공간을 반환한다.
```

```
int *ip;  
ip=(int *)malloc(sizeof(int));  
*ip=20;  
free(ip); // 포인터를 전달인자로 주고 호출한다.
```

## ▶ 동적 할당되는 메모리는 확인해야 한다.

- 동적할당함수는 힙 영역에 원하는 크기의 메모리가 존재하지 않으면 **널 포인터**를 리턴한다. 따라서 이 값을 확인하여 널포인터를 참조하지 않도록 해야 한다.

```
ip=(int *)malloc(sizeof(int));
if(ip==0){                                     // 리턴된 포인터가 널 포인터인지 확인한다.
    printf("메모리가 부족합니다.\n");         // 널 포인터이면 메시지만 출력한다.
}
else{                                           // 기억공간이 할당되었으면 사용한다.
    *ip=10;
    printf("%d\n", *ip);
}
```

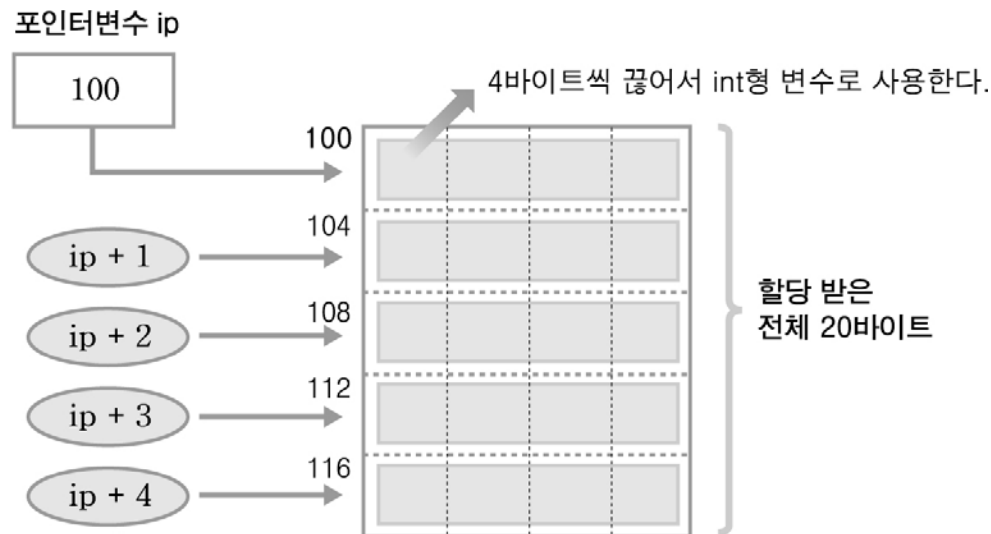
- 널 포인터는 보통 **"NULL"**이라는 이름으로 기호화 하여 사용하는데 전처리 단계에서 0으로 바뀌므로 상수값 0과 같다.

## □ 동적할당 기억공간의 활용

- 메모리의 동적할당은 많은 기억공간을 한꺼번에 할당 받아서 배열로 사용하는 것이 효율적이다.

- int형 변수 5개를 동적으로 할당 받는 경우

```
int *ip;           // 포인터변수 선언  
ip = (int *)malloc(20); // 20바이트를 한꺼번에 할당 받는다.
```



## ▶ 메모리를 동적으로 할당 받아 배열로 사용하는 프로그램

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

int main()
{
    int *ip;
    int i, sum=0;

    ip=(int *)malloc(5*sizeof(int));           // 전체 20바이트의 기억공간 할당
    if(ip==0){                                 // 메모리가 할당 되었는지 확인하여
        printf("메모리가 부족합니다!\n");      // 메모리가 부족하면 메시지를 출력하고
        return 1;                               // 프로그램을 종료한다.
    }
    printf("다섯 명의 나이를 입력하세요 : ");
    for(i=0; i<5; i++){
        scanf("%d", ip+i);                     // 데이터를 저장할 포인터를 전달한다.
        sum+=ip[i];                             // 입력된 값을 참조하여 누적한다.
    }
    printf("다섯 명의 평균나이 : %.1lf\n", sum/5.0); // 평균나이 출력
    free(ip);                                   // 할당 받은 메모리 반환

    return 0;
}
```

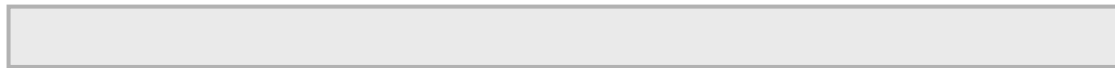
출력 결과

```
다섯 명의 나이를 입력하세요 : 21 27 24 22 35 (엔터)
다섯 명의 평균나이 : 25.8
```

## ▶ 동적 할당을 사용하여 문자열을 처리하자.

- 메모리 동적 할당을 사용하면 입력되는 문자열의 길이에 맞게 기억 공간을 할당할 수 있다.

- ① 문자열을 입력 받기 전에는 그 길이를 알 수 없으므로 우선 충분한 크기의 문자배열이 필요하다.



- ② 문자배열에 문자열을 입력 받는다.

뇌를 자극하는 C프로그래밍

- ③ 문자열의 길이를 계산하여 그 크기에 맞게 기억공간을 동적으로 할당 받는다.



- ④ 동적으로 할당 받은 기억공간에 입력 받은 문자열을 복사한다.

뇌를 자극하는 C프로그래밍



입력된 문자열의 길이에 딱 맞는 기억공간

## ▶ 세 개의 문자열을 저장하기 위한 동적 할당 프로그램

```
#include <stdio.h>
#include <string.h>           // 문자열 처리함수를 위한 헤더파일 포함
#include <stdlib.h>          // 동적 할당 함수를 위한 헤더파일 포함

int main()
{
    char temp[80];           // 임시 문자배열, 충분히 크게 확보한다.
    char *str[3];           // 동적 할당된 기억공간을 연결할 포인터배열
    int i;                  // 반복 제어변수

    for(i=0; i<3; i++){
        printf("문자열을 입력하세요 : ");
        gets(temp);         // 문자열 입력
        str[i]=(char *)malloc(strlen(temp)+1); // 입력 받은 문자열의 길이에 맞게 동적 할당한다.
        strcpy(str[i], temp); // 동적 할당된 기억공간에 문자열을 복사한다.
    }

    for(i=0; i<3; i++){
        printf("%s\n", str[i]); // 포인터배열의 요소를 참조하여 입력된 문자열 출력
    }

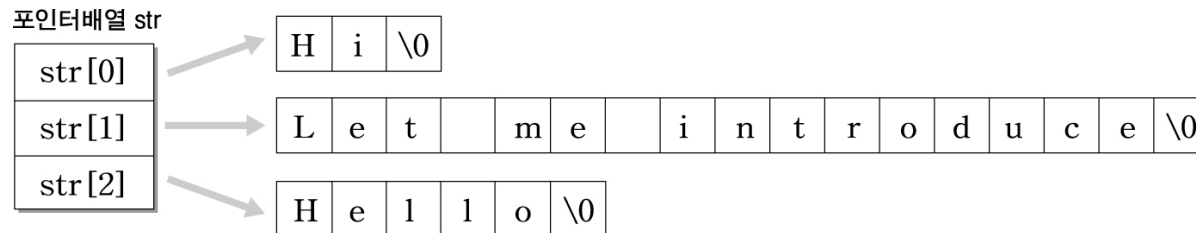
    for(i=0; i<3; i++){
        free(str[i]);       // 할당 받은 메모리를 반환한다.
    }
    return 0;
}
```

## ▶ 세 개의 문자열을 저장하기 위한 동적 할당 프로그램

### 출력 결과

```
문자열을 입력하세요 : Hi (엔터)
문자열을 입력하세요 : Let me introduce (엔터)
문자열을 입력하세요 : Hello (엔터)
Hi
Let me introduce
Hello
```

- 포인터배열은 각 동적 할당 기억공간을 연결하여 가변배열을 만든다.



- 포인터배열은 널포인터로 초기화하여 문자열을 출력 할 때 널포인터를 검사하는 방법을 쓸 수 있다.

```
char *str[100]={0};      | for(i=0; str[i]!=0; i++){
                          |     printf(“%s\n”, str[i]);
                          | }
                          |
```

## ▶ 가변배열의 문자열을 함수로 출력하자.

- 가변배열에 저장된 문자열을 출력할 때 포인터배열의 배열명을 사용한다.

```
str_prn(str); // 포인터배열의 배열명을 주고 함수 호출
```

- 포인터배열의 배열명은 (char \*)형을 가리키므로 매개변수는 이중포인터 변수를 사용한다.

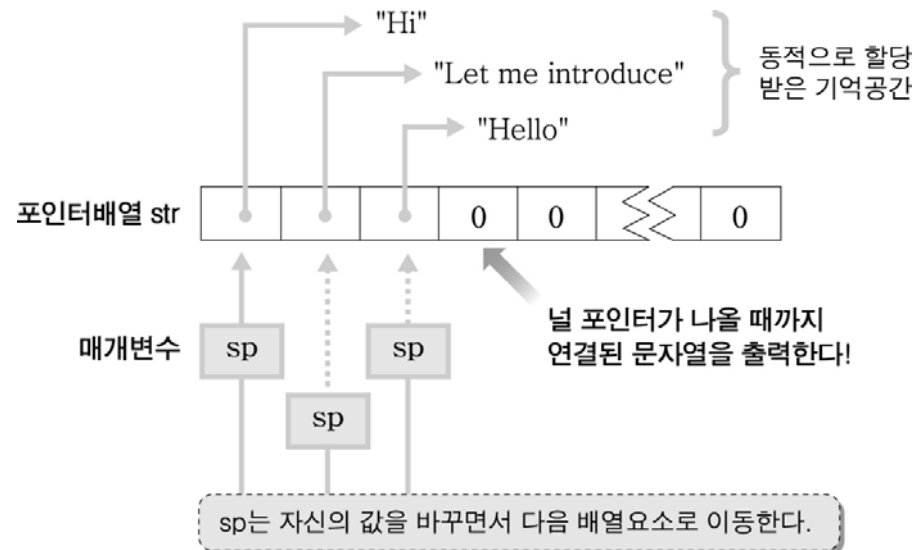
배열명 str → 

char *	char *	char *
--------	--------	--------

 void str\_prn(char \*\*sp); // 함수의 원형 선언

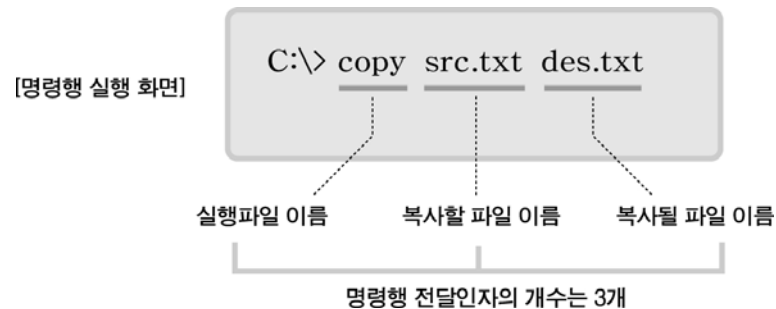
- 매개변수를 사용하여 가변배열에 저장된 문자열을 출력한다.

```
void str_prn(char **sp)
{
    while(*sp!=0){
        printf("%s\n", *sp);
        sp++;
    }
}
```

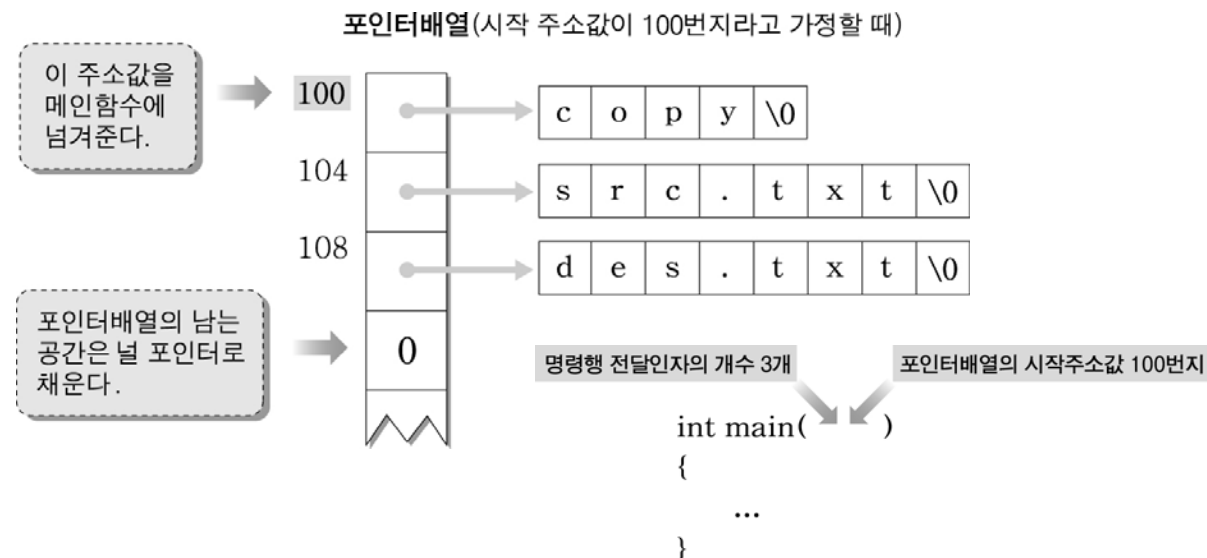


## □ 메인함수의 전달인자

- 메인함수의 전달인자는 실행될 때 명령행에서 만들어진다.



- 운영체제는 명령행의 문자열을 가변배열로 만들고 그 개수와 포인터를 메인함수에 전달한다.



## ▶ 명령행 전달인자를 출력하는 프로그램

```
#include <stdio.h>

int main(int argc, char **argv) // 명령행 전달인자를 받을 매개변수
{
    int i;

    for(i=0; i<argc; i++){ // 전달인자(문자열)의 개수만큼 반복
        printf("%s\n", argv[i]); // 전달인자(문자열)를 하나씩 출력한다.
    }

    return 0;
}
```

출력 결과

```
C:\W>mycommand first_arg second_arg (엔터)
MYCOMMAND
first_arg
second_arg

C:\W>_
```

- 포인터배열의 마지막 배열요소는 널포인터로 채워지므로 널포인터를 검사하여 출력하는 것도 가능하다.

```
while(*argv != 0){ // argv가 가리키는 포인터배열의 값이 널포인터가 아닌 동안
    printf("%s\n", *argv); // 문자열을 출력한다.
    argv++; // 포인터배열의 다음 배열요소로 이동
}
```

## ▶ 명령행 전달인자와 동적 할당을 사용한 문자열 입력 예제

- 다음과 같이 실행되는 프로그램을 만들어보자.
  1. 입력되는 문자열을 길이에 맞게 동적으로 할당 받은 기억공간에 저장한다.  
단, 문자열의 최대 길이는 널문자까지 포함하여 80바이트로 한다.
  2. 프로그램을 작성할 때는 문자열이 몇 개가 입력될지 알 수 없으나 실행할 때는 명령행 전달인자로 그 최대값을 입력해 준다.  

```
C:\W> strings 100 // 최대 100개의 문자열을 입력할 수 있다.
```
  3. 새로운 문자열을 입력할 때 엔터키만 입력하면 입력을 끝내고 그 동안 입력된 문자열을 출력한다.

```
C:\W>strings 5 (엔터)
문자열을 입력하세요 : What's up! (엔터)
문자열을 입력하세요 : Good morning~ (엔터)
문자열을 입력하세요 : Hi... *^^* (엔터)
문자열을 입력하세요 : (엔터)
What's up!
Good morning~
Hi... *^^*
```

## ▶ 명령행 전달인자와 동적 할당을 사용한 문자열 입력 예제

4. 만약 최대 입력 문자열을 모두 입력하면 메시지와 함께 그 동안 입력된 문자열을 출력하고 종료한다.

```
C:\W>strings 3
문자열을 입력하세요 : Cheer up.
문자열을 입력하세요 : Good evening.
문자열을 입력하세요 : Hello world!
문자열 입력이 최대값을 모두 채웠습니다.
Cheer up.
Good evening.
Hello world!
```

5. 문자열을 입력하는 부분은 함수로 작성한다.

## ▶ 문자열을 입력하여 저장하는 프로그램 : PART1

```
#include <stdio.h>
#include <string.h>           // strlen함수를 위한 헤더파일
#include <stdlib.h>          // malloc, atoi함수를 위한 헤더파일

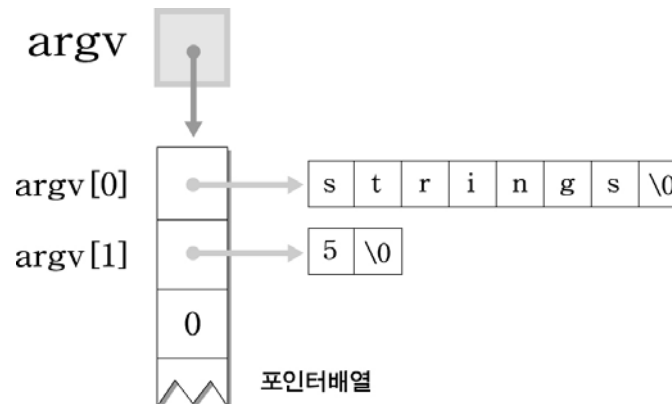
void str_prn(char **);       // 문자열 출력함수의 선언

int main(int argc, char **argv) // 명령행 전달인자를 매개변수에 받는다.
{
    char temp[80];           // 문자열 입력을 위한 임시 문자배열
    char **str;              // 포인터배열을 연결할 이중포인터변수
    int max;                 // 최대 입력 문자열 수를 저장할 변수
    int i;                   // 반복 제어변수

    max=atoi(argv[1]);     // 문자열을 수치값으로 변환한다.
    str=(char **)malloc((max+1) * sizeof(char *)); // 포인터배열도 동적 할당한다.
```

C:\W>strings 5

실행하면  
가변배열이  
만들어진다.



## ▶ 문자열을 입력하여 저장하는 프로그램 : PART1

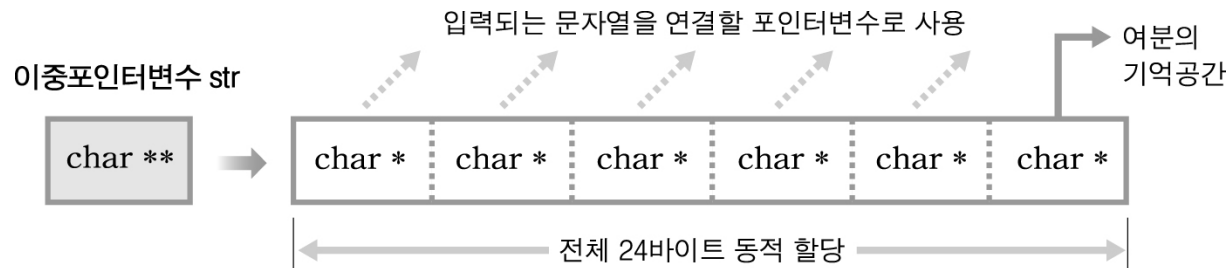
▷ `max=atoi(argv[1]);`

- `argv[1]`이 연결하고 있는 숫자형태의 문자열을 수치값으로 바꾼다.
- `atoi`함수는 문자열을 전달인자로 받아서 수치값으로 바꾸어 준다.

```
int atoi(char *); // ascii to integer, 문자열을 정수값으로 바꾼다.
```

▷ `str=(char **)malloc((max+1) * sizeof(char *));`

- `max`의 수만큼 문자열을 연결할 포인터배열을 동적으로 할당 받는다. 이 때 데이터의 끝을 표시하기 위해 여분의 기억공간을 하나 더 할당 받는다. 할당 받은 기억공간은 이중포인터변수 `str`에 연결해 둔다.



## ▶ 문자열을 입력하여 저장하는 프로그램 : PART2

```
i=0; // 반복 제어변수 초기화
while(1){ // 무한 반복
    printf("문자열을 입력하세요 : ");
    gets(temp); // 문자열 입력
    if(temp[0]=='\0') break; // 엔터만 입력되면 temp배열의 첫번째 요소는 널문자가 된다.

    str[i]=(char *)malloc(strlen(temp)+1); // 문자열을 저장할 메모리 할당하고 포인터 배열요소에 연결한다.
    strcpy(str[i], temp); // 문자열 복사
    i++; // 하나의 문자열을 입력할 때마다 하나씩 증가한다.
    if(i==max){ // 입력된 문자열의 수를 검사하여 최대값을 다 채우면 반복문 종료
        printf("문자열 입력이 최대값을 모두 채웠습니다.\n");
        break;
    }
}
str[i]=0; // 입력이 끝난 후에 포인터배열의 마지막 요소는 널포인터로 마감한다.
str_prn(str); // 입력된 문자열은 함수를 호출하여 출력한다.
i=0; // 반복 제어변수를 초기화하고
while(str[i]!=0){ // 포인터배열에 연결된 동적할당 기억공간들을 반환한다.
    free(str[i]);
    i++;
}
free(str); // 동적으로 할당 받은 포인터배열의 기억공간도 반환한다.
return 0;
}
```

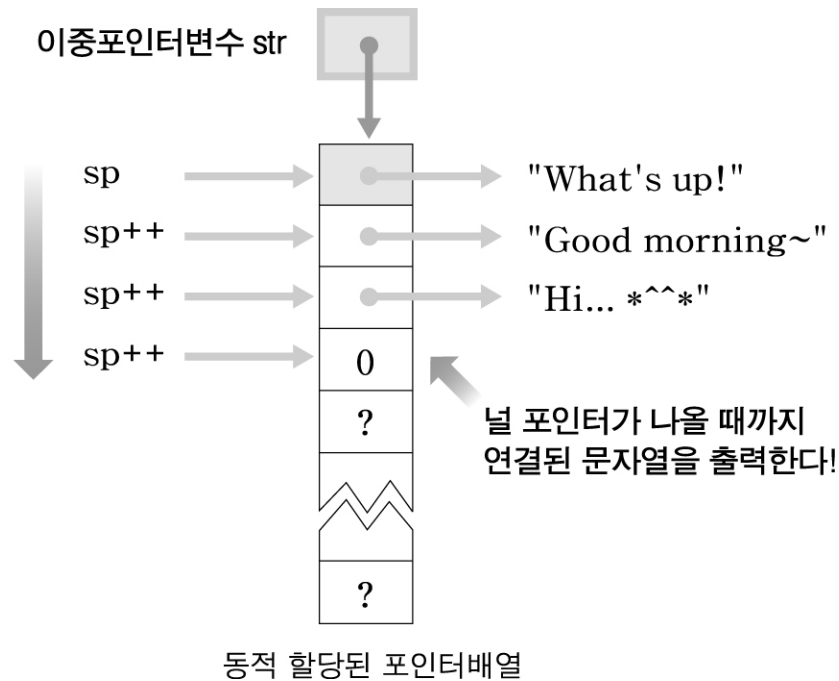
## ▶ 문자열을 입력하여 저장하는 프로그램 : PART3

```

void str_prn(char **sp)           // 매개변수는 이중포인터변수
{
    while(*sp != 0){             // 포인터배열의 값이 널 포인터가 아닐 때까지
        printf("%s\n", *sp);    // 포인터배열이 연결하고 있는 문자열 출력
        sp++;                    // 포인터배열의 배열요소를 차례로 이동한다.
    }
}

```

매개변수는 하나씩 증가되면서 다음 배열 요소를 가리킨다.



## □ 기타 동적 할당 함수

- calloc함수는 배열을 할당 받고 초기화한다.

```
void *calloc(unsigned int, unsigned int);
```

- 첫 번째 배열요소의 개수, 두 번째는 배열요소의 크기를 전달인자로 준다.
- double형 변수 5개로 사용할 배열을 할당 받는 경우

```
double *dp;
```

```
dp = (double *) calloc ( 5, sizeof(double) );
```

배열요소의 개수

double형 변수 하나의 크기

```
double *ap;
```

```
int i;
```

```
ap=(double *)calloc(5, sizeof(double));
```

```
for(i=0; i<5; i++){
```

```
    printf("%lf\n", ap[i]);
```

```
}
```

출력 결과

```
0.000000  
0.000000  
0.000000  
0.000000  
0.000000
```

// 모두 0으로  
// 초기화 된다.

## ▶ realloc 함수로 기억공간의 크기를 조절한다.

- 저장된 데이터에 변화가 생기면 할당 받은 기억공간을 realloc 함수로 조절할 수 있다.

```
void *realloc(void *, unsigned int); // 기억공간의 재할당
```

- 첫번째 전달인자는 이미 할당 받은 기억공간의 포인터를 준다.
- 두 번째는 새로 할당 받고자 하는 크기를 바이트 단위로 입력한다.

```
int *ip;
```

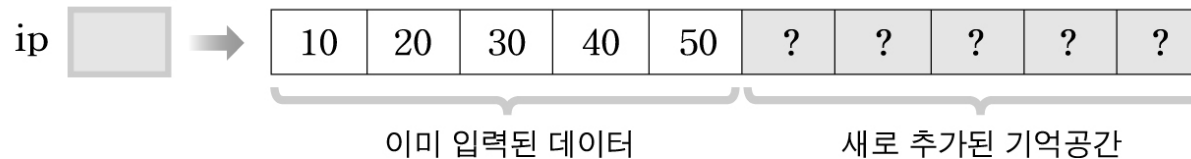
```
ip = (int *) calloc (5, sizeof(int));
```

```
... // 할당 받은 기억공간에 10, 20, 30, 40, 50을 저장한 경우
```

```
ip = (int *) realloc ( ip, 10*sizeof(int) );
```

이미 할당 받은  
기억공간의 위치

재할당 받을 전체  
기억공간의 크기



## ▶ realloc 함수를 사용한 양수값 입력 프로그램

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

int main()
{
    int *ip
    int size=5;    // 처음에 5개만 입력한다.
    int cnt=0;    // 입력된 데이터의 수를 센다.
    int num;
    int i;

    ip=(int *)calloc(size, sizeof(int)); // 일단 5개를 입력할 기억공간만 할당 받는다.
    while(1){                             // while(TRUE)
        printf("양수를 입력하세요 => ");
        scanf("%d", &num);
        if(num<=0) break;
        if(cnt<size){                     // 입력된 데이터가 5개를 넘지 않으면
            ip[cnt++]=num;                // 할당 받은 기억공간에 차례로 저장한다.
        }
        else{                              // 그렇지 않으면
            size+=5;                      // 데이터의 최대값을 5씩 증가한다.
            ip=(int *)realloc(ip, size*sizeof(int)); // 새로 증가된 수만큼 기억공간을 늘린다.
            ip[cnt++]=num;                 // 새로 할당된 기억공간에 저장한다.
        }
    }
}
```