13장 고급 프로그래밍

김명호

내용

- C 시스템의 메모리 배치
- 대형 프로그램의 구성
- 정적 외부 변수와 함수
- 추상 자료형
- 가변 인자 함수
- 미리 정의된 매크로와 가변 인자 매크로
- 자기참조 구조체
- 라이브러리
- 신호

메모리 배치

- C 프로그램 실행 메모리

메모리 배치

- C 프로그램 실행 메모리
  - 스택 : 함수의 지역 변수, 매개변수, 리턴 주소 저장
  - 힙 : malloc(), calloc()에 의해 할당되는 공간
  - 텍스트 : 실행 코드(문장) 저장
  - 데이터 : 전역 변수, 정적 변수 저장
- 실행 파일에는 텍스트와 초기화된 전역 변수와 정적 변수 저장
  - 초기화되지 않은 것에 대해서는 이름과 크기만 저장(BSS, Block Started by Symbol)

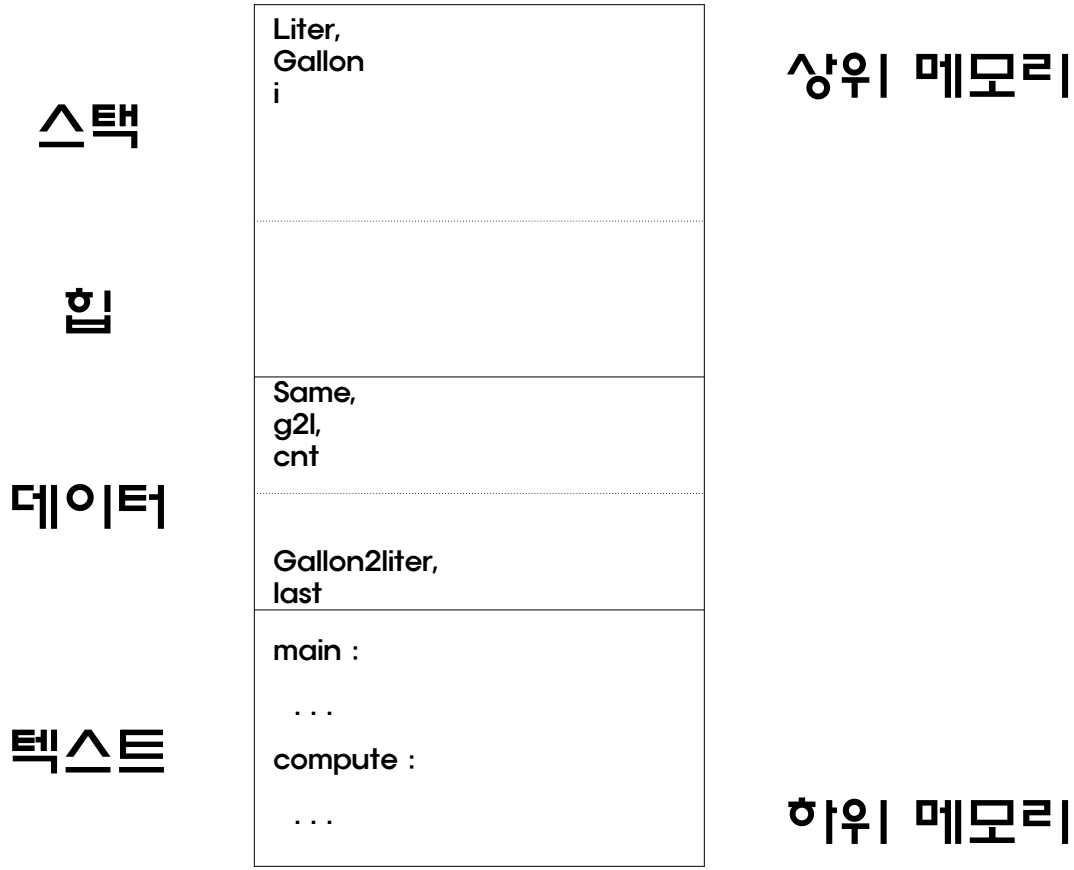
메모리 배치

프로그램 13.1

```
int   same;                          // BSS 부분에 저장
float gallon2liter = 3.7854118;      // 데이터 부분에 저장
float g2l[N][2];                     // BSS 부분에 저장
int   last = 1;                      // 데이터 부분에 저장
float compute(float);
// main()과 compute() 함수의 모든 문장은 텍스트 부분에 저장
int main(void){
   float liter, gallon;
   int   i;
   . . .
}
float compute(float input){
   static int cnt = 0;                // BSS 부분에 저장
   float  result;
   int    i;
   . . .
}
```

메모리 배치

- C 프로그램 실행 메모리



대형 프로그램 구성

- 프로그램은 여러 파일로 작성될 수 있음
- 보통 관련된 함수들을 하나의 파일에 저장함
- 한 프로그램을 이루는 파일들은 보통 같은 디렉터리에 저장됨

대형 프로그램 구성

프로그램 13.2 (main.c)

```
#include <stdio.h>
typedef struct grade{
   int   grade[3];
   char  p_f[3];
   int   sum;
   float avg;
}grade;
int grade_proc3(grade *);
int main(void){
   grade st = {{0}, {0}, -1, -1.0};
   printf("성적 입력(국어, 산수, 과학) : ");
   scanf("%d%d%d", &st.grade[0], &st.grade[1], &st.grade[2]);
   if (grade_proc3(&st))
      return 1;
   printf("국어 : %d (%c)\n", st.grade[0], st.p_f[0]);
   printf("산수 : %d (%c)\n", st.grade[1], st.p_f[1]);
   printf("과학 : %d (%c)\n", st.grade[2], st.p_f[2]);
   printf("총점 : %d\n", st.sum);
   printf("평균 : %.2f\n", st.avg);
   return 0;
}
```

대형 프로그램 구성

프로그램 13.2 (grade.c)

```
#include <stdio.h>
typedef struct grade{
   int   grade[3];
   char  p_f[3];
   int   sum;
   float avg;
}grade;
int grade_proc3(grade * stp)
{
   if (stp == NULL) {
      printf("오류 : NULL 포인터\n");
      return -1;
   }
   stp -> sum = stp -> grade[0] + stp -> grade[1] + stp -> grade[2];
   stp -> avg = stp -> sum / 3.0;
   stp -> p_f[0] = stp -> grade[0] < 60 ? 'f' : 'p';
   stp -> p_f[1] = stp -> grade[1] < 60 ? 'f' : 'p';
   stp -> p_f[2] = stp -> grade[2] < 60 ? 'f' : 'p';
   return 0;
}
```

대형 프로그램 구성

- **컴파일**
  - 프로그램을 이루는 모든 파일을 명시하면 됨

```
gcc -o grade main.c grade.c
```

- **파일 별 컴파일**

```
gcc -c main.c
gcc -c grade.c
gcc -o grade main.o grade.o
```

  - 수정된 파일만 컴파일 하면 됨

```
gcc -c main.c
gcc -o grade main.o grade.o
```

대형 프로그램 구성

- 사용자 헤더 파일
  - 프로그램을 이루는 파일들이 공통으로 사용하는 헤더파일이나, 매크로, 함수 원형, 구조체 선언 등을 모아두는 파일
  - 확장자로 h를 사용함
  - 다른 소스 파일과 같은 디렉터리에 둠
  - 대형 프로그램을 여러 명이 같이 개발할 때 유용함
  - 헤더파일을 여러 번 포함해도 되게끔 작성하는 것이 좋음

대형 프로그램 구성

프로그램 13.3 (grade.h)

```
#include <stdio.h>
typedef struct grade{
   int   grade[3];
   char  p_f[3];
   int   sum;
   float avg;
}grade;
int grade_proc3(grade *);
```

대형 프로그램 구성

프로그램 13.3 (main.c)

```
#include "grade.h"
int main(void)
{
   grade st = {{0}, {0}, -1, -1.0};

   printf("성적 입력(국어, 산수, 과학) : ");
   scanf("%d%d%d", &st.grade[0], &st.grade[1], &st.grade[2]);
   if (grade_proc3(&st))
      return 1;
   printf("국어 : %d (%c)\n", st.grade[0], st.p_f[0]);
   printf("산수 : %d (%c)\n", st.grade[1], st.p_f[1]);
   printf("과학 : %d (%c)\n", st.grade[2], st.p_f[2]);
   printf("총점 : %d\n", st.sum);
   printf("평균 : %.2f\n", st.avg);
   return 0;
}
```

대형 프로그램 구성

프로그램 13.3 (grade.c)

```
#include "grade.h"
int grade_proc3(grade * stp)
{
   if (stp == NULL) {
      printf("오류 : NULL 포인터\n");
      return -1;
   }
   stp -> sum = stp -> grade[0] + stp -> grade[1] + stp -> grade[2];
   stp -> avg = stp -> sum / 3.0;
   stp -> p_f[0] = stp -> grade[0] < 60 ? 'f' : 'p';
   stp -> p_f[1] = stp -> grade[1] < 60 ? 'f' : 'p';
   stp -> p_f[2] = stp -> grade[2] < 60 ? 'f' : 'p';
   return 0;
}
```

대형 프로그램 구성

프로그램 13.3 (grade.h)

```
#ifndef  _GRADE_H_
// 같은 헤더 파일이 두 번 이상 포함되는 것을 방지함
#define  _GRADE_H_
#include <stdio.h>
typedef struct grade{
   int   grade[3];
   char  p_f[3];
   int   sum;
   float avg;
}grade;
void grade_proc3(grade *);
#endif   // _GRADE_H_
```

외부 변수

- 함수 밖에 선언된 변수
- 모든 함수에서 참조가 가능함
- 함수 간에 정보 전달을 위해 유용함
- 다른 파일에 정의된 외부 변수는 extern을 사용하여 참조할 수 있음

외부 변수

프로그램 13.4 (main.c)

```
#include <stdio.h>
int   quotient, rem;          // 전역 변수 선언
int   divide(int, int);
int main(void)
{
   int a = 10, b = 3;
   extern int   quotient, rem;    // 전역 변수 참조 선언, 생략할 수 있음
   if (divide(a, b))
      printf("0으로 나눌 수 없습니다.\n");
   else
      printf("%d / %d : 몫은 %d이고 나머지는 %d입니다.\n",
             a, b, quotient, rem);
   return 0;
}
```

외부 변수

프로그램 13.4 (divide.c)

```
int is_zero(int);
int divide(int dividend, int divisor)
{
   extern int   quotient, rem;          // 전역 변수 참조 선언
   if (is_zero(divisor))
      return -1;
   quotient = dividend / divisor;
   rem = dividend % divisor;
   return 0;
}
int is_zero(int num)
{
   if (num)
      return 0;
   return 1;
}
```

정적 외부 변수

- 외부 변수에 static이 적용된 변수
- 정적 외부 변수는 그 변수가 선언된 파일 내에 있는 함수에서만 사용될 수 있음
- 데이터 접근을 제한하여 안전한 프로그램을 만들 수 있음

정적 외부 함수

- 함수 정의에 static이 적용된 함수
- 정적 외부 함수는 그 함수가 정의된 파일 내에 있는 함수에서만 사용될 수 있음

정적 외부 변수

프로그램 13.5 (password.c)

```
#include <stdbool.h>
static int passwd = 78;                // 정적 외부 변수, 다른 파일에서 사용 못함
static bool is_login = false;          // 정적 외부 변수, 다른 파일에서 사용 못함
static void set_login(bool ok){        // 정적 외부 함수, 다른 파일에서 호출 못함
   is_login = ok;
}
int access(int num){
   if (num != passwd) {
      set_login(false);
      return -1;
   }
   set_login(true);
   return 0;
}
int change_passwd(int new){
   if (!is_login)
      return -1;
   passwd = new;
   return 0;
}
```

정적 외부 변수

프로그램 13.5 (main.c)

```
#include <stdio.h>
int   access(int);
int   change_passwd(int);
int main(void){
   int num, new, select = 1;
   while (select){
      printf("0:종료, 1:로그인, 2:패스워드 변경\n번호를 선택하세요 :");
      scanf("%d", &select);
      switch(select) {
      . . .
      case 1:
         printf("패스워드 : ");
         scanf("%d", &num);
         if (access(num))
            printf("잘못된 패스워드입니다.\n");
         else
            printf("로그인 됐습니다.\n");
         break;
```

정적 외부 변수

프로그램 13.5 (main.c)

```
        case 2:
            printf("새로운 패스워드 : ");
            scanf("%d", &new);
            if (change_passwd(new))
                printf("패스워드를 변경할 수 없습니다.\n");
            else
                printf("패스워드가 변경됐습니다.\n");
            break;
        default:
            printf("잘못된 번호입니다.\n");
        }
    }
    return 0;
}
```

프로그램 결과

```
$ password
0:종료, 1:로그인, 2:패스워드 변경
번호를 선택하세요 :1
패스워드 : 78
로그인 됐습니다.
0:종료, 1:로그인, 2:패스워드 변경
번호를 선택하세요 :2
새로운 패스워드 : 90
패스워드가 변경됐습니다.
0:종료, 1:로그인, 2:패스워드 변경
번호를 선택하세요 :0
프로그램을 종료합니다.
$ password
0:종료, 1:로그인, 2:패스워드 변경
번호를 선택하세요 :1
패스워드 : 90
잘못된 패스워드입니다.
0:종료, 1:로그인, 2:패스워드 변경
번호를 선택하세요 :2
새로운 패스워드 : 90
패스워드를 변경할 수 없습니다.
0:종료, 1:로그인, 2:패스워드 변경
번호를 선택하세요 :0
프로그램을 종료합니다.
```

추상 자료형

- 처리해야 할 데이터 집합과 그 연산을 정의한 명세
- 데이터는 추상 자료형에서 정의된 연산자에 의해서만 다루어 지기 때문에 데이터를 보호할 수 있음
- 구현에 대해서는 고려하지 않음

프로그램 13.1

```
#include <stdio.h>
#define N 10
int same;
float gallon2liter = 3.7854118;
float g2l[N][2];
int last = 1;
float compute(float);
```

프로그램 13.1

```
int main(void){
   float liter, gallon;
   int i;
   while (last){
      printf("겔론 : ");
      scanf("%f", &gallon);
      liter = compute(gallon);
      printf("%.2f 겔론은 %.2f 리터입니다.\n", gallon, liter);
   }
   printf("%d 번 변환 했습니다.\n", N);
   for (i = 0; i < N; i++)
      printf("%.2f 겔론 : %.2f 리터\n", g2l[i][0], g2l[i][1]);
   return 0;
}
```

프로그램 13.1

```
float compute(float input){
   static int cnt = 0;
   float result;
   int i;
   for (i = 0; i < cnt; i++)
      if (g2l[i][0] == input) {
         same++;
         return g2l[i][1];
      }
   result = input * gallon2liter;
   g2l[cnt][0] = input;
   g2l[cnt][1] = result;
   cnt++;
   if (cnt == N)  last = 0;
   return result;
}
```

프로그램 13.1

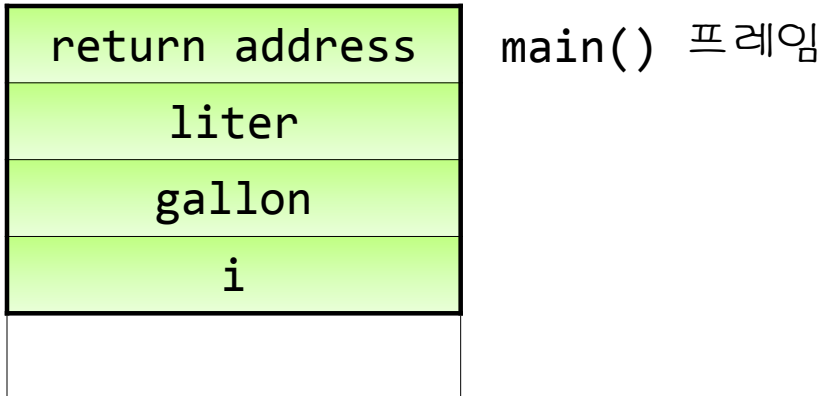
```
int main(void){
   float liter, gallon;
   int i;
   while (last){
      . . .
      liter = compute(gallon);
      . . .
   }
   . . .
}
float compute(float input){
   static int cnt = 0;
   float result;
   int i;
   . . .
}
```

메모리 스택

- 함수 호출에 의해 생성되는 함수 프레임은 메모리의 스택 부분에 저장됨
  - 함수 프레임 : 지역 변수, 매개 변수, 리턴 주소로 구성됨
  - 저장된 함수 프레임은 함수가 종료되면 삭제됨
  - 가장 나중에 호출된 함수가 가장 빨리 종료되기 때문에 메모리 스택에서 가장 먼저 삭제되는 것은 가장 최근에 쌓인 함수 프레임임(후입선출)

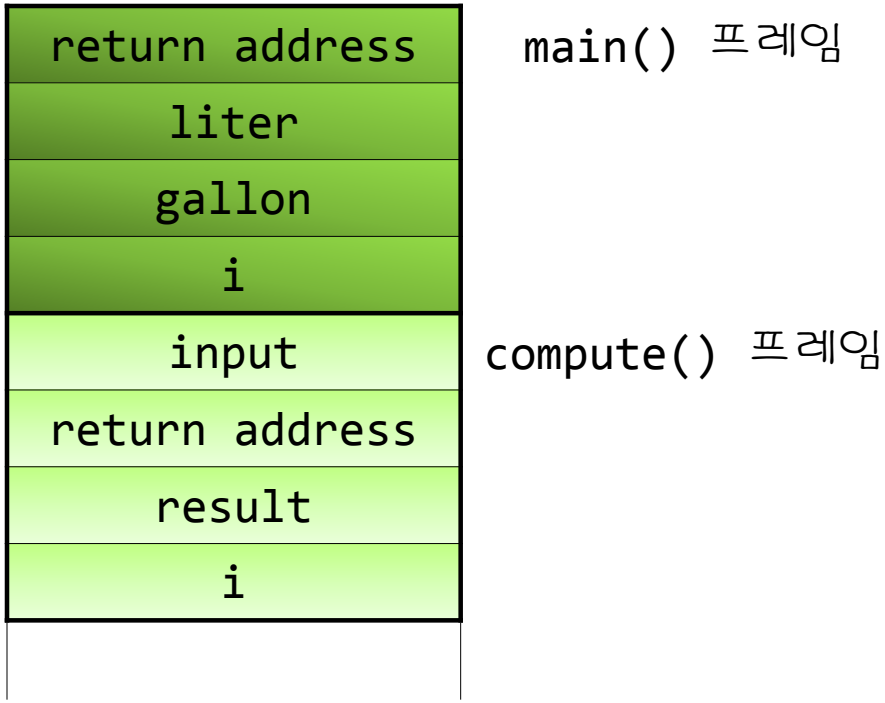
추상 자료형

- **함수 호출 예 (프로그램 13.1)**
  - main() 함수 호출



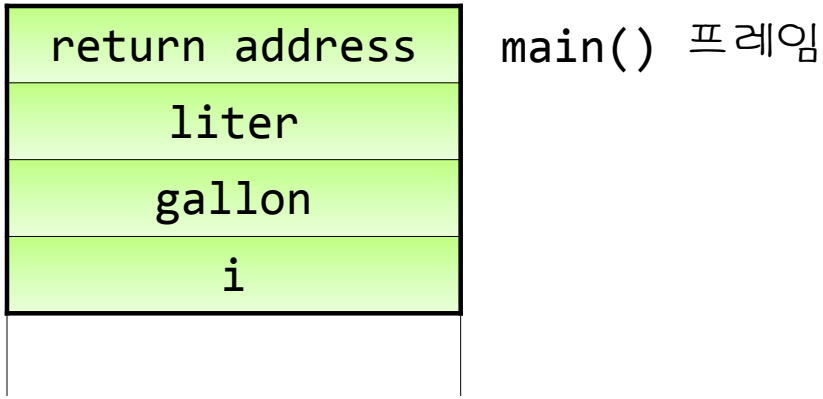
추상 자료형

- **함수 호출 예 (프로그램 13.1)**
  - compute() 함수 호출



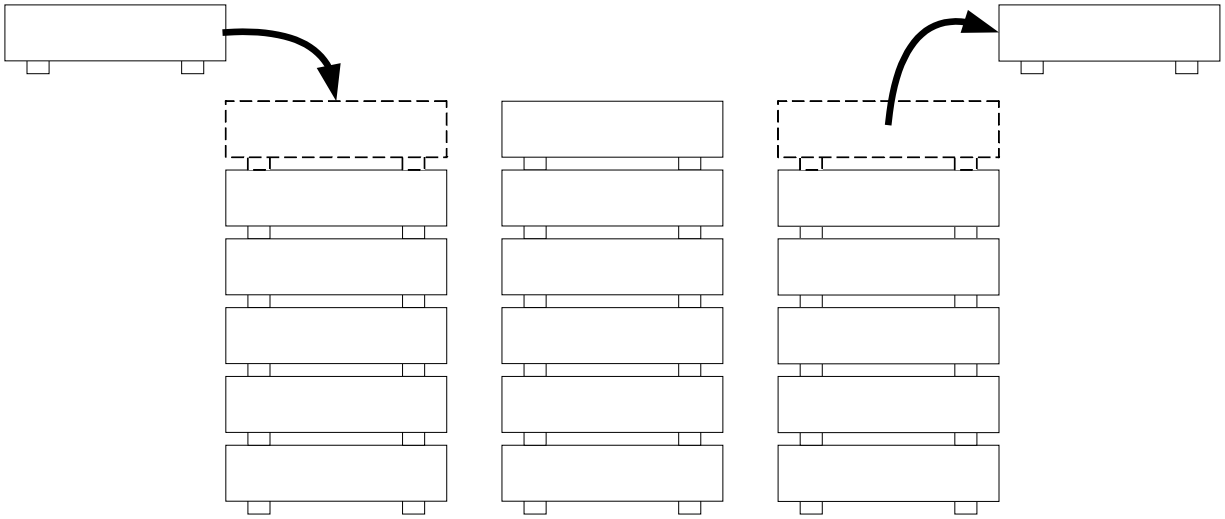
추상 자료형

- **함수 호출 예 (프로그램 13.1)**
  - compute() 함수 return



추상 자료형

- **추상 자료형 스택(stack)**
  - 데이터를 선형으로 저장하는 자료구조로 나중에 저장한 것을 먼저 사용함 (후입선출)



삽입 삭제

추상 자료형

- 처리해야 할 데이터 집합과 그 연산을 정의한 명세
- 데이터는 추상 자료형에서 정의된 연산자에 의해서만 다루어 지기 때문에 데이터를 보호할 수 있음
- 구현에 대해서는 고려하지 않음

추상 자료형

- 추상 자료형 스택(stack)
  - 연산자
    - push : 스택에 데이터 삽입
    - pop : 스택으로부터 데이터 삭제
    - top : 스택의 톱에 있는 데이터를 삭제하지 않고 확인
    - empty : 스택이 비어 있는지 검사
    - full : 스택이 꽉 찼는지 검사
    - reset : 스택 초기화

추상 자료형 스택 구현

- 데이터를 선형으로 저장하는 방법
  - 배열
  - 링크드 리스트
- 문자를 저장하는 스택을 배열로 구현

```
#define MAX 100
typedef struct stack {
    char s[MAX];
    int top;
} stack;
```

추상 자료형 스택 구현

- **스택 연산자 구현 예**
  - push 연산자

함수 13.1

```
void push(char c, stack *stk)
{
    stk -> top++;
    stk -> s[stk -> top] = c;
}
```

추상 자료형 스택 구현

프로그램 13.6 (stack.h)

```
#include <stdio.h>
#include <stdbool.h>
#define   MAX_LEN   100
#define   EMPTY      -1
#define   FULL        (MAX_LEN - 1)
typedef struct stack {
   char   s[MAX_LEN];
   int    top;
} stack;
void reset(stack *stk);
void push(char c, stack *stk);
char pop(stack *stk);
char top(const stack *stk);
bool empty(const stack *stk);
bool full(const stack *stk);
```

추상 자료형 스택 구현

프로그램 13.6 (stack.c)

```
#include "stack.h"
void reset(stack *stk){
   stk -> top = EMPTY;
}
void push(char c, stack *stk){
   stk -> top++;
   stk -> s[stk -> top] = c;
}
char pop(stack *stk){
   return (stk -> s[stk -> top--]);
}
char top(const stack *stk){
   return (stk -> s[stk -> top]);
}
bool empty(const stack *stk){
   return (stk -> top == EMPTY);
}
bool full(const stack *stk){
   return (stk -> top == FULL);
}
```

추상 자료형 스택 구현

프로그램 13.6 (main.c)

```
#include "stack.h"
int main(void){
   char   str[] = "Stack Test!";
   int    i;
   stack  s;
   reset(&s);          // 스택 초기화
   printf("문자열 : %s\n", str);
   //  스택에 문자열 push
   for (i = 0; str[i] != '\0'; ++i)
      if (!full(&s))
         push(str[i], &s);
   printf("역 문자열 : ");
   // 스택에서 문자열 pop
   while (!empty(&s))
      putchar(pop(&s));
   putchar('\n');
   return 0;
}
```

프로그램 결과

```
문자열 : Stack Test!
역 문자열 : !tseT kcatS
```

가변 인자 함수

- 함수의 매개변수 개수가 정해져 있지 않는 함수
- 매개변수에서 …(점 세 개)으로 표시
- printf()

```
printf("재미있는 C!\n");
printf("이름 : %s.", "김진혁");
printf("%d * %d =  %d\n", 123, 43, 123 * 43);
```

– 함수 원형

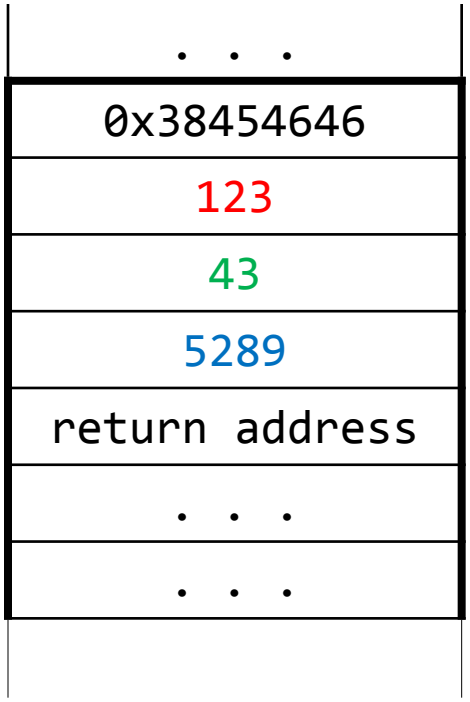
```
int printf(const char *cntrl_string, ...);
```

가변 인자 함수

- **스택**

```
printf("%d * %d =  %d\n", 123, 43, 123 * 43);
```

– printf() 호출

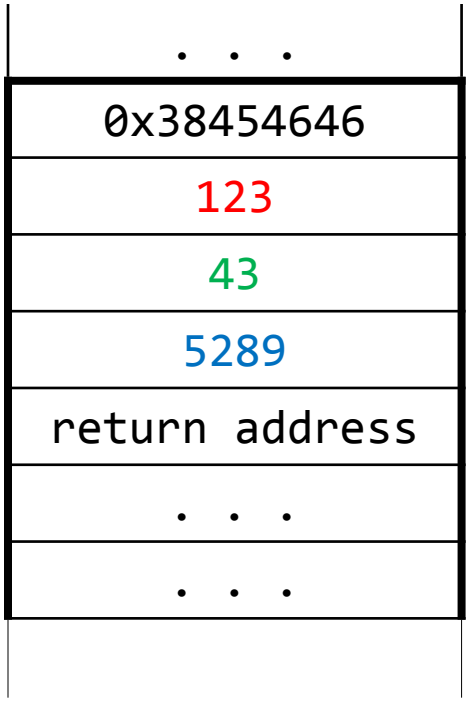


가변 인자 함수

- **스택**

```
printf("%d * %d =  %d\n", 123, 43, 123 * 43);
```

– printf() 호출



char *	4 byte
int	4 byte
int	4 byte
int	4 byte

가변 인자 함수

- <stdarg.h>에 정의된 매크로를 사용하여 만듦

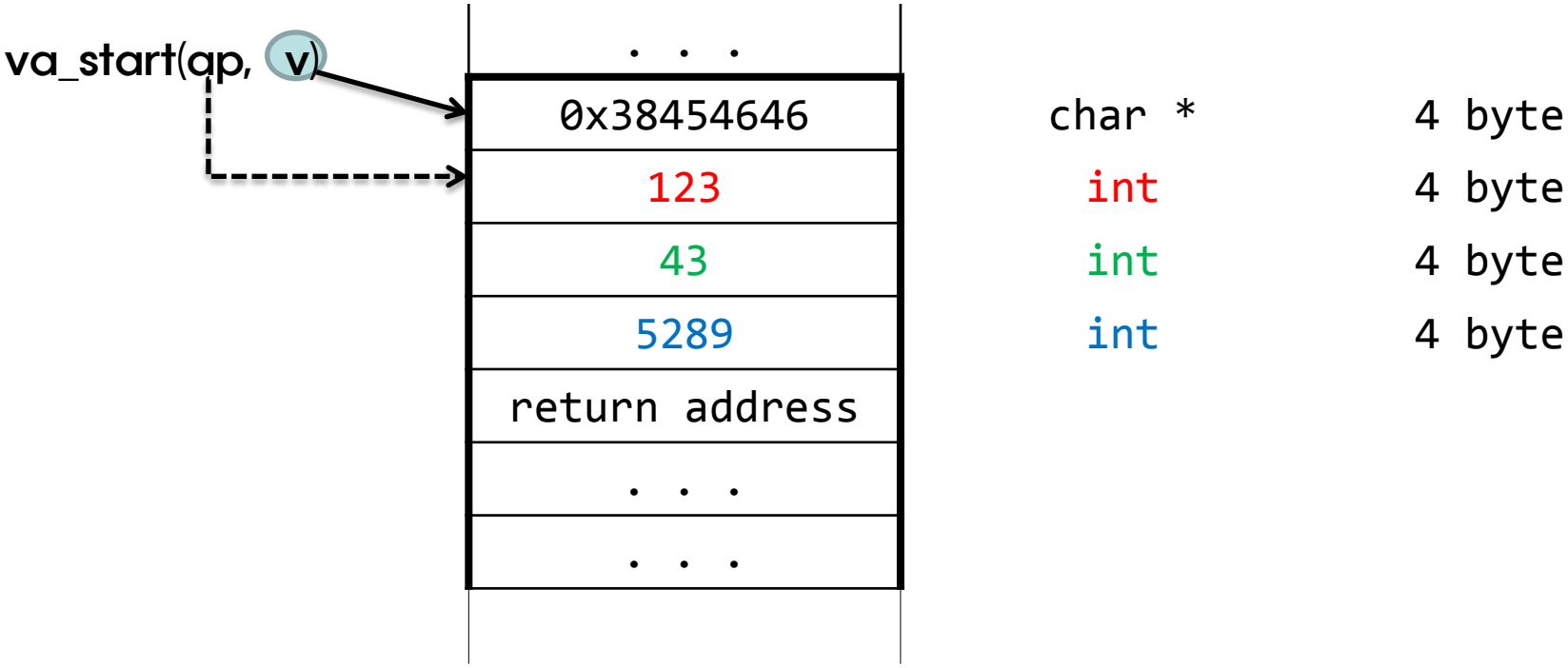
매크로	설명
va_start(ap, v)	ap가 가변 인자의 첫 번째 인자를 포인트 하게 한다. v는 가변 인자 바로 직전의 인자 이다.
va_arg(ap, type)	ap가 포인트 하는 곳의 값을 type 형으로 읽고, ap는 다음 인자를 포인트 하게 한다.
va_copy(dest, src)	src 포인터를 dest로 복사한다. C99에서 추가되었다.
va_end(ap)	va_start()로 초기화된 ap 포인터를 제거한다. va_start()를 사용한 함수에서 사용해야 한다.

가변 인자 함수

- **스택**

```
printf("%d * %d =  %d\n", 123, 43, 123 * 43);
```

– printf() 호출

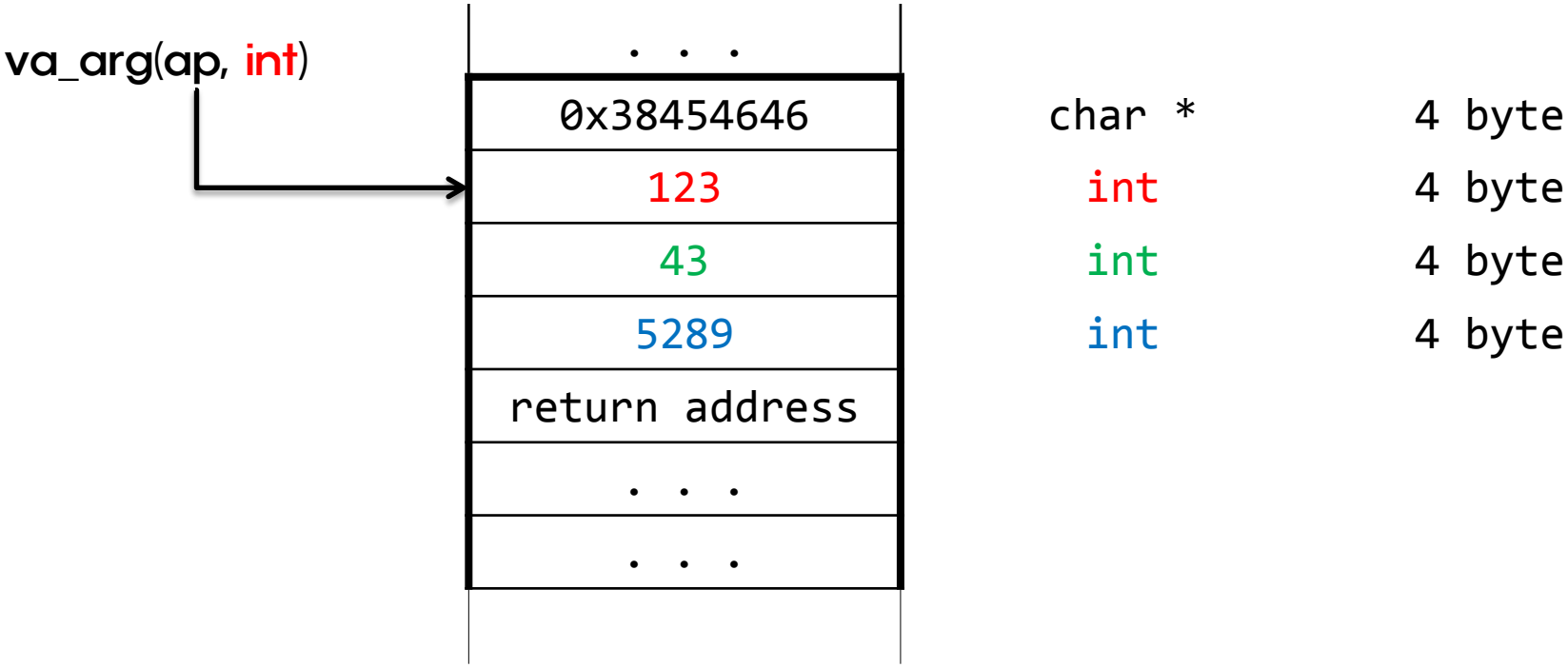


가변 인자 함수

- **스택**

```
printf("%d * %d =  %d\n", 123, 43, 123 * 43);
```

– printf() 호출

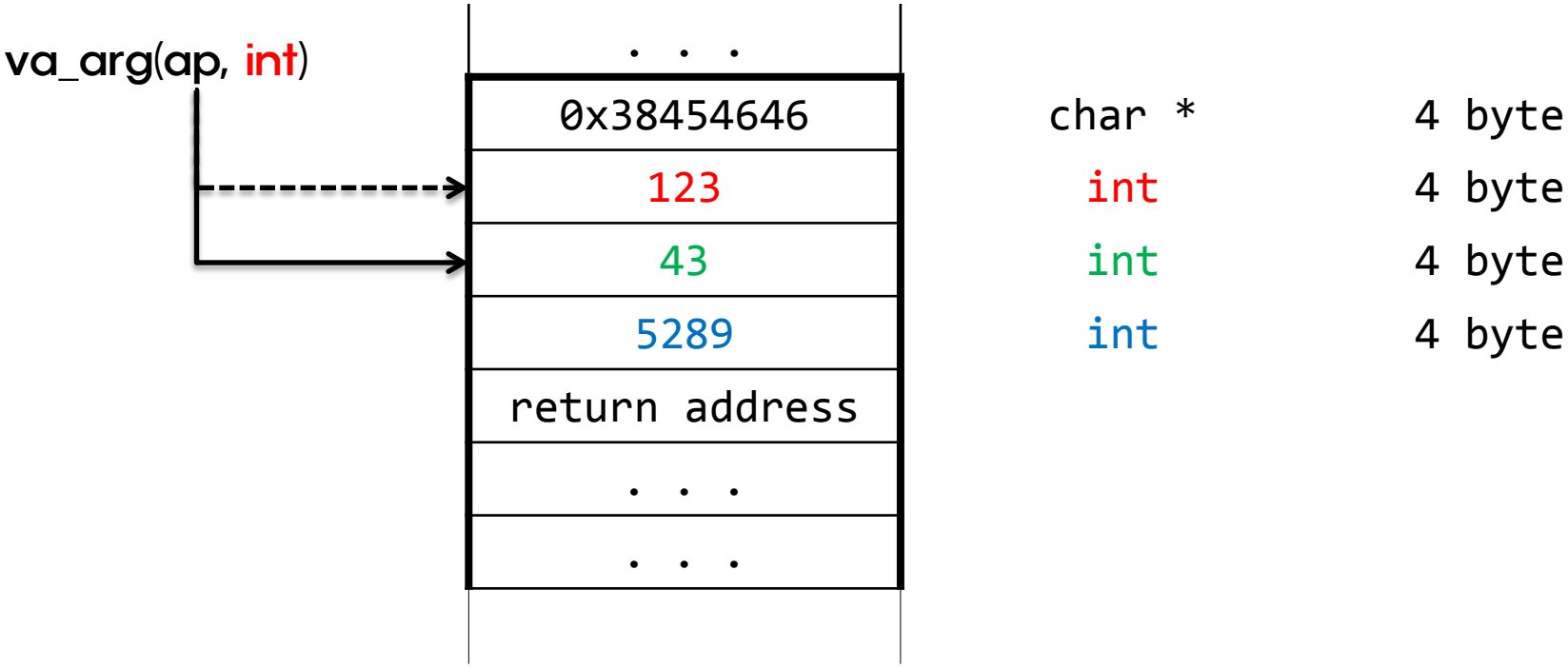


가변 인자 함수

- **스택**

```
printf("%d * %d =  %d\n", 123, 43, 123 * 43);
```

– printf() 호출



가변 인자 함수

프로그램 13.7

```
#include  <stdio.h>
#include  <stdarg.h>
#define END 0
#define INT 1
#define DOUBLE 2

double  va_sum(int, ...);

int main(void)
{
   printf("Sum = %.3f\n", va_sum(INT, 3, DOUBLE, 3.0, END));
   printf("Sum = %.3f\n", va_sum(DOUBLE, 20.0, INT, 90, DOUBLE, 0.5, END));
   return 0 ;
}
```

가변 인자 함수

프로그램 13.7

```
double va_sum(int type, ...)
{
   double  sum = 0.0;
   va_list  ap;
   va_start(ap, type);
   while(type != 0){
      if (type == INT)
         sum += va_arg(ap, int);
      else
         sum += va_arg(ap, double);
      type = va_arg(ap, int);
   }
   va_end(ap);
   return sum;
}
```

프로그램 결과

```
Sum = 6.000
Sum = 110.500
```

미리 정의된 매크로

- 7개의 미리 정의된 매크로가 있음

매크로	값
__DATE__	현재 날짜를 포함하는 문자열
__FILE__	파일 이름을 포함하는 문자열
__LINE__	현재 라인 번호를 나타내는 정수
__STDC__	표준을 따르는 경우 1, 아니면 0
__TIME__	현재 시간을 포함하는 문자열
__STDC_HOSTED__	호스트 구현이면 1, 아니면 0
__STDC_VERSION__	정수 상수 199901L

- 미리 정의된 식별자(C99) __func__ : 현재 수행중인 함수 이름을 갖는 문자 배열

가변 인자 매크로

- C99부터 매크로도 가변 인자를 가질 수 있음
- 가변 인자를 점 세개(…)로 표시
- 매크로 정의에서 가변 매개변수는 __VA_ARGS__로 명시
- 예제

```
#define ERRPRINTF(...) \
  printf("오류 : (" __FILE__ " 파일) "__VA_ARGS__)
```

가변 인자 매크로

프로그램 13.8

```
#include <stdio.h>
#define ERRPRINTF(...) printf("오류 : (" __FILE__ " 파일) "__VA_ARGS__)
int main(void){
   float divisor, dividend;

   printf("******** 나누기 프로그램 ********\n");
   printf("피제수를 입력하세요 : ");
   scanf("%f", &dividend);
   printf("제수를 입력하세요 : ");
   scanf("%f", &divisor);

   if (divisor == 0.0)
      ERRPRINTF("제수가 %.3f 입니다.\n", divisor);
   else
      printf("%.3f / %.3f = %.3f\n", dividend, divisor,
              dividend / divisor);
   return 0;
}
```

프로그램 결과

```
******** 나누기 프로그램 ********
피제수를 입력하세요 : 89
제수를 입력하세요 : 0
오류 : (divide.c 파일) 제수가 0입니다.
```

가변 인자 매크로

- **컴파일러 메시지 형태**

```
divide.c: In function `main':
divide.c:14: error: 'divisor' undeclared (first use in this
  function)
```

  - 오류가 있는 파일명, 함수명, 행번호 표시

- **가변 인자 매크로로 만든 오류 출력 매크로**

```
#define ERRPRINTF2(...) \
printf(__FILE__ ": '%s' 함수:\n", __func__),\
printf(__FILE__ ":%d:오류:", __LINE__),\
printf(__VA_ARGS__)
```

  - 여러 문장을 사용하기 위해 콤마 연산자 사용

가변 인자 매크로

- 매크로 정의에서 콤마 연산자 대신 문장의 끝을 나타내는 세미콜론(;)을 사용하면 문제가 발생함

```
#define ERRPRINTF2(...) \
printf(__FILE__ ": '%s' 함수:\n", __func__);\
printf(__FILE__ ":%d:오류:", __LINE__);\
printf(__VA_ARGS__)
```

```
ERRPRINTF2("제수가 0입니다.\n");
```

⇓

```
printf("divide.c" ": '%s' 함수:\n", __func__);
printf("divide.c" ":%d:오류:", 17);
printf("제수가 0입니다.\n", divisor);
```

가변 인자 매크로

- **문제 발생 예제**

```
if (divisor == 0.0)
   ERRPRINTF2("제수가 0입니다.\n");
else
   printf("%.3f / %.3f = %.3f\n", dividend, divisor, dividend /
  divisor);
```

⇓

```
if (divisor == 0.0)
   printf("divide.c" ": '%s' 함수:\n", __func__);
   printf("divide.c" ":%d:오류:", 17);
   printf("제수가 0입니다.\n", divisor);
else
   printf("%.3f / %.3f = %.3f\n", dividend, divisor, dividend /
  divisor);
```

가변 인자 매크로

- 매크로가 여러 문장으로 정의될 때에는 do-while 구문을 사용하는 것이 좋음

```
#define ERRPRINTF2(...) \
    do {\
        printf(__FILE__ ": '%s' 함수:\n", __func__);\
        printf(__FILE__ ":%d:오류:", __LINE__);\
        printf(__VA_ARGS__); \
    } while (0)
```

  - while 의 조건으로 0을 사용해야 함, 0이외의 값을 사용하면 무한 루프에 빠짐

가변 인자 매크로

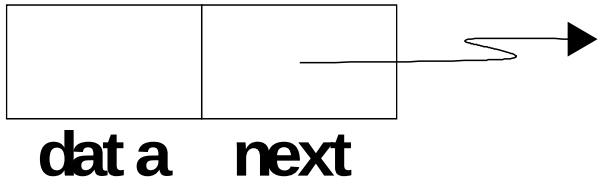
- do-while 문은 한 문장으로 취급되기 때문에 오류가 발생하지 않음

```
if (divisor == 0.0)
   do {
      printf("divide.c" ": '%s' 함수:\n", __func__);
      printf("divide.c" ":%d:오류:", 17);
      printf("제수가 0입니다.\n", divisor);
   } while (0);
else
   printf("%.3f / %.3f = %.3f\n", dividend, divisor, dividend /
  divisor);
```

자기 참조 구조체

- 자신과 같은 구조체 형을 포인트하는 멤버를 갖는 구조체
- 예

```
struct list {
    int             data;
    struct list    *next;
} a;
```



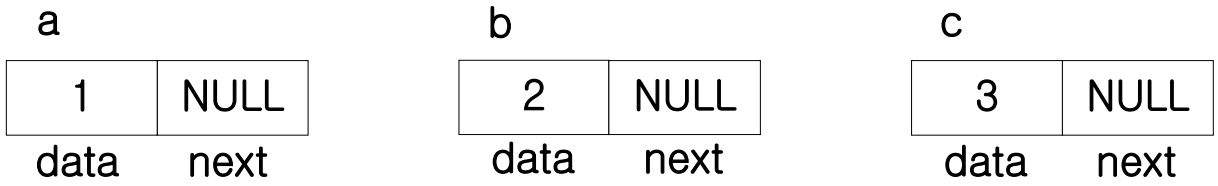
자기 참조 구조체

프로그램 13.9

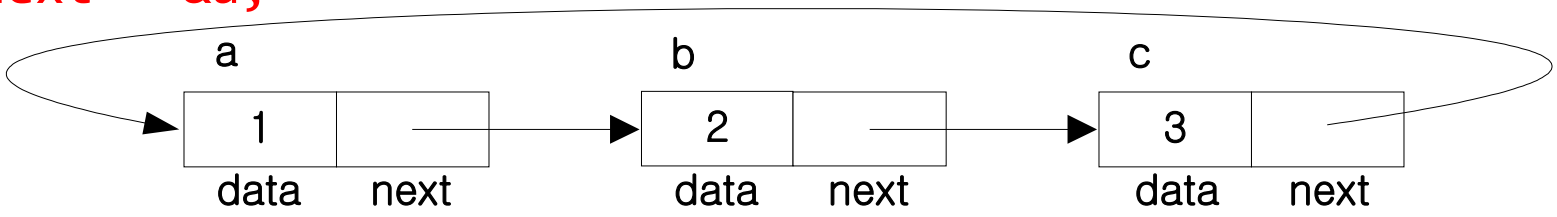
```
#include <stdio.h>
struct list {
   int            data;
   struct list   *next;
};
int main(void){
   struct list  a = {1, NULL}, b = {2, NULL}, c = {3, NULL};
   a.next = &b;
   b.next = &c;
   c.next = &a;
   printf("a : %d, b : %d, c : %d\n", a.data, b.data, c.data);
   printf("a : %d, b(a.next->data) : %d, c(b.next->data) : %d\n",
                 a.data, a.next->data, b.next->data);
   printf("a : %d, b(c.next->next->data) : %d, \
                 c(a.next->next->data) : %d\n",
                 a.data, c.next->next->data, a.next->next->data);
   return 0;
}
```

자기 참조 구조체

```
struct list a = {1, NULL}, b = {2, NULL}, c = {3, NULL};
```




```
a.next = &b;
b.next = &c;
c.next = &a;
```




```
b.next -> data ?
a.next -> next -> data ?
```

프로그램 결과

```
a : 1, b : 2, c : 3
a : 1, b(a.next->data) : 2, c(b.next->data) : 3
a : 1, b(c.next->next->data) : 2, c(a.next->next->data) : 3
```

동적 메모리 할당과 자기 참조 구조체

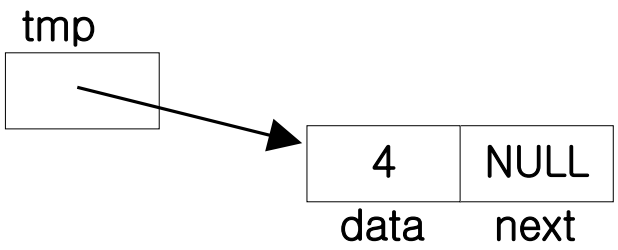
- 자기 참조 구조체는 동적 메모리 할당 기법과 함께 사용됨
- 동적으로 할당받은 구조체는 기존 구조체가 포인트하여 유지함
- 동적 메모리 할당은 malloc()과 calloc()으로 함
- 예

```
tmp = (struct list *)malloc(sizeof(struct list));
```

동적 메모리 할당과 자기 참조 구조체

- 동적으로 할당받은 메모리 공간은 포인터를 통해 구조체를 다루는 것과 똑같이 다루면 됨
- 예

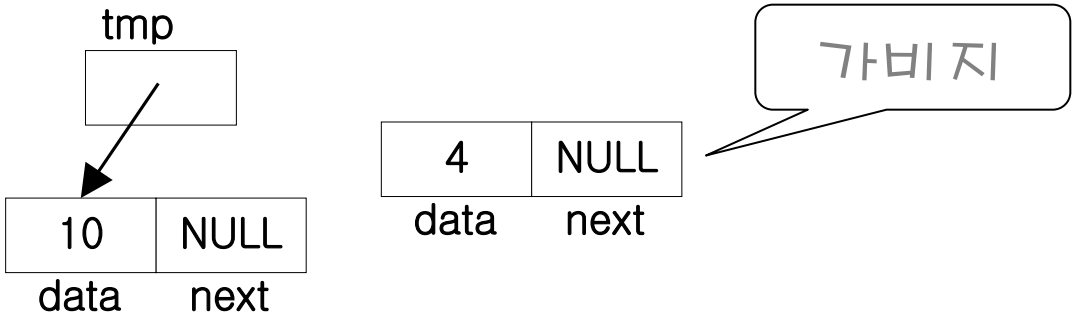
```
tmp = (struct list *)malloc(sizeof(struct list));
tmp -> data = 4;
tmp -> next = NULL;
```



동적 메모리 할당과 자기 참조 구조체

- 동적으로 할당받은 메모리 공간은 어느 방법으로든 접근할 수 있게 만들어야 함
  - 가비지 : 접근할 수 없는 동적으로 할당받은 메모리 공간
- 예

```
tmp = (struct list *)malloc(sizeof(struct list));
tmp -> data = 10;
tmp -> next = NULL;
```



동적 메모리 할당과 자기 참조 구조체

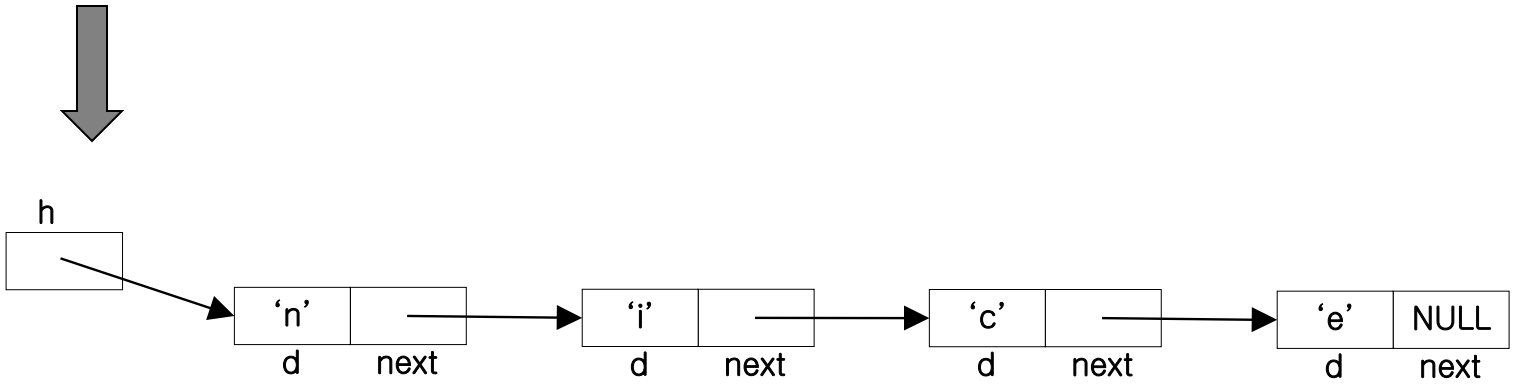
- 가비지가 발생하지 않게 하기 위해서는 이전 포인터를 잘 관리해야 함
- 예

```
tmp -> next = (struct list *)malloc(sizeof(struct list));
tmp -> next -> data = 10;
tmp -> next -> next = NULL;
```

동적 메모리 할당과 자기 참조 구조체

- 자기 참조 구조체를 다루는 프로그램은 재귀 함수를 많이 사용함
- 예제 프로그램
  - 문자열을 링크드 리스트로 변환
  - 예

"nice"



동적 메모리 할당과 자기 참조 구조체

프로그램 13.10

```
#include <stdio.h>
#include <stdlib.h>
#define   N   10
struct linked_list {
   char                d;
   struct linked_list  *next;
};
typedef  struct linked_list   ELEMENT;
typedef  ELEMENT             *LINK;

LINK string_to_list(char s[]){
   LINK  head;
   if (s[0] == '\0')
      return NULL;
   else {
      head = malloc(sizeof(ELEMENT));
      head -> d = s[0];
      head -> next = string_to_list(s + 1);
      return head;
   }
}
```

동적 메모리 할당과 자기 참조 구조체

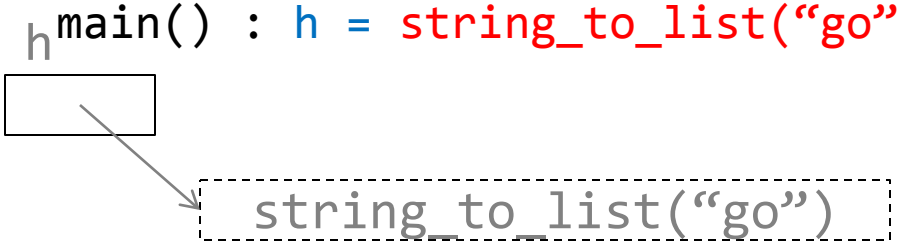
프로그램 13.10

```
void print_list(LINK head){
   if (head == NULL)
      printf("NULL\n");
   else {
      printf("%c --> ", head -> d);
      print_list(head -> next);
   }
}
int main(void){
   char   input[N];
   LINK  h;
   printf("문자열 입력: ");
   scanf("%s", input);
   h = string_to_list(input);
   printf("변환 리스트 결과 : \n");
   print_list(h);
   return 0;
}
```

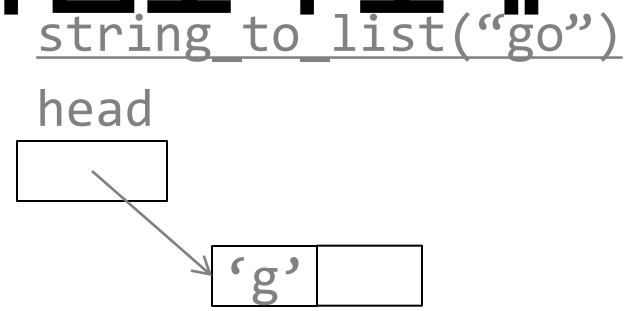
동적 메모리 할당과 자기 참조 구조체

프로그램 13.10

```
LINK string_to_list(char s[]){
   LINK  head;
   if (s[0] == '\0')
      return NULL;
   else {
      head = malloc(sizeof(ELEMENT));
      head -> d = s[0];
      head -> next = string_to_list(s + 1);
      return head;
   }
}
main() : h = string_to_list("go");
```

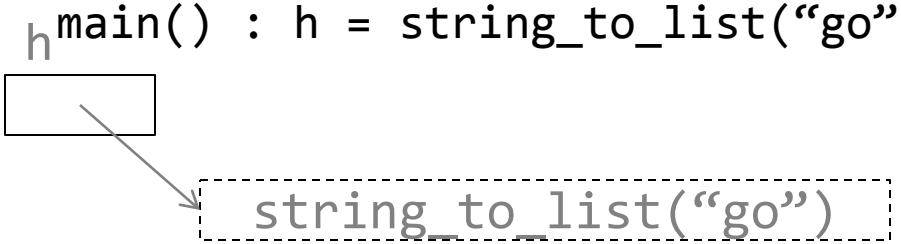


동적 메모리 할당과 자기 참조 구조체



프로그램 13.10

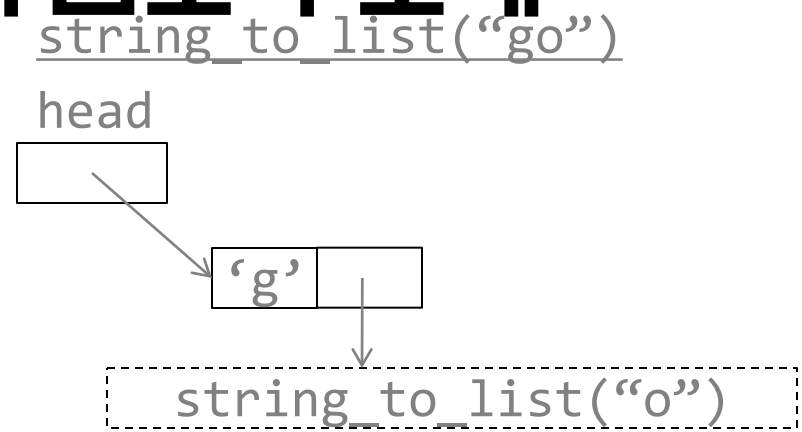
```
LINK string_to_list(char s[]){
    LINK  head;
    if (s[0] == '\0')
        return NULL;
    else {
        head = malloc(sizeof(ELEMENT));
        head -> d = s[0];
        head -> next = string_to_list(s + 1);
        return head;
    }
}
main() : h = string_to_list("go");
```



동적 메모리 할당과 자기 참조 구조체

프로그램 13.10

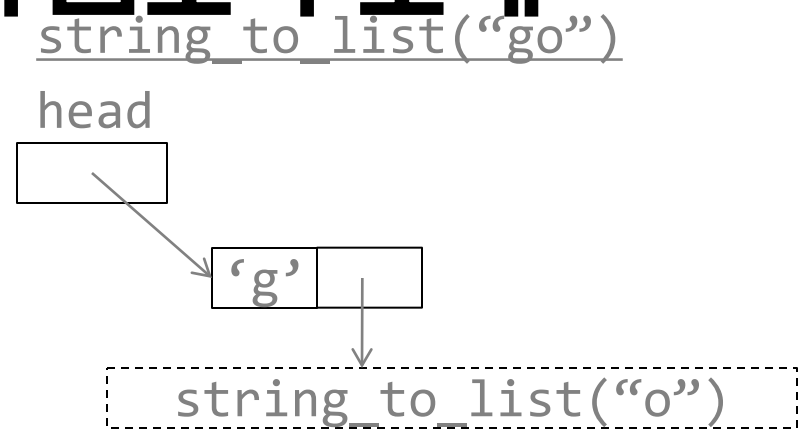
```
LINK string_to_list(char s[]){
    LINK  head;
    if (s[0] == '\0')
        return NULL;
    else {
        head = malloc(sizeof(ELEMENT));
        head -> d = s[0];
        head -> next = string_to_list(s + 1);
        return head;
    }
}
main() : h = string_to_list("go");
```

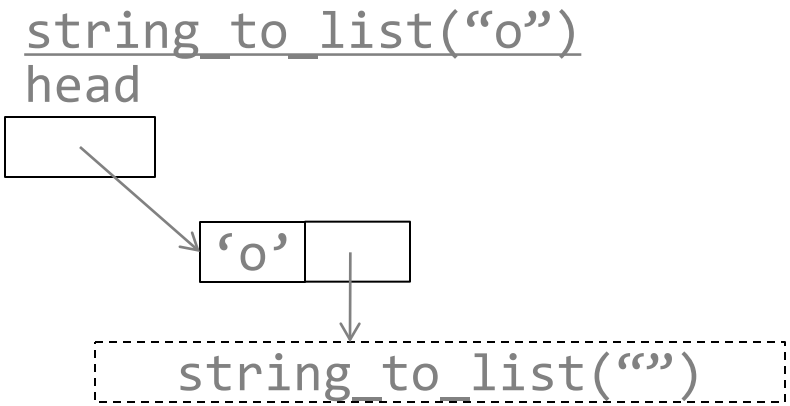


동적 메모리 할당과 자기 참조 구조체

프로그램 13.10

```
LINK string_to_list(char s[]){
    LINK  head;
    if (s[0] == '\0')
        return NULL;
    else {
        head = malloc(sizeof(ELEMENT));
        head -> d = s[0];
        head -> next = string_to_list(s + 1);
        return head;
    }
}
main() : h = string_to_list("go");
```

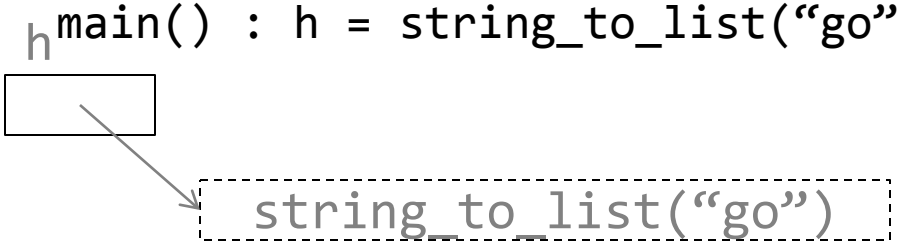


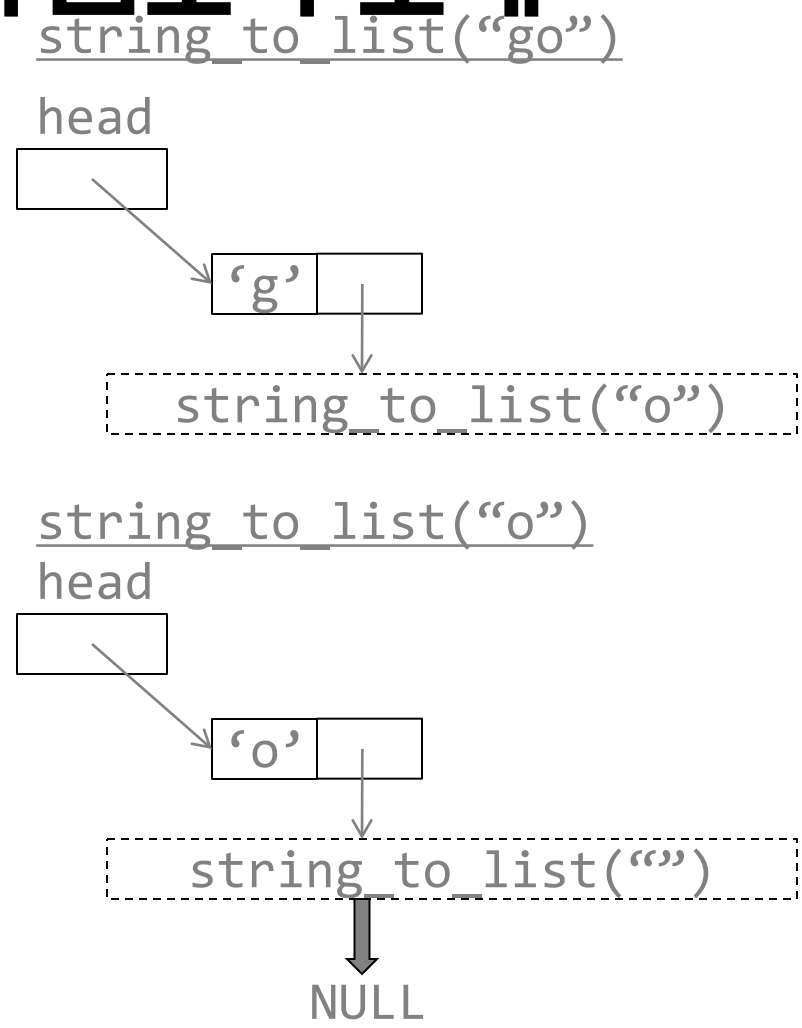


동적 메모리 할당과 자기 참조 구조체

프로그램 13.10

```
LINK string_to_list(char s[]){
    LINK  head;
    if (s[0] == '\0')
        return NULL;
    else {
        head = malloc(sizeof(ELEMENT));
        head -> d = s[0];
        head -> next = string_to_list(s + 1);
        return head;
    }
}
main() : h = string_to_list("go");
```

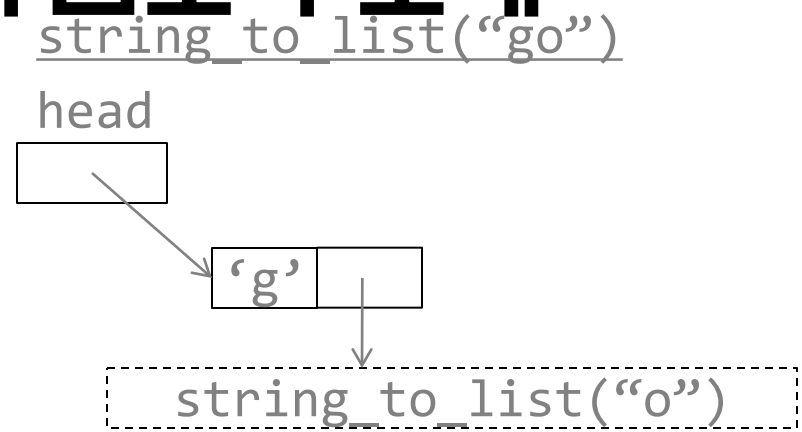


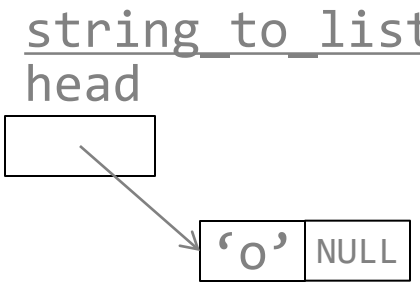


동적 메모리 할당과 자기 참조 구조체

프로그램 13.10

```
LINK string_to_list(char s[]){
    LINK  head;
    if (s[0] == '\0')
        return NULL;
    else {
        head = malloc(sizeof(ELEMENT));
        head -> d = s[0];
        head -> next = string_to_list(s + 1);
        return head;
    }
}
main() : h = string_to_list("go");
```

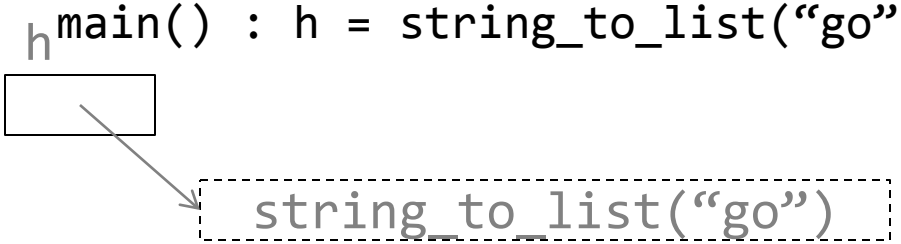


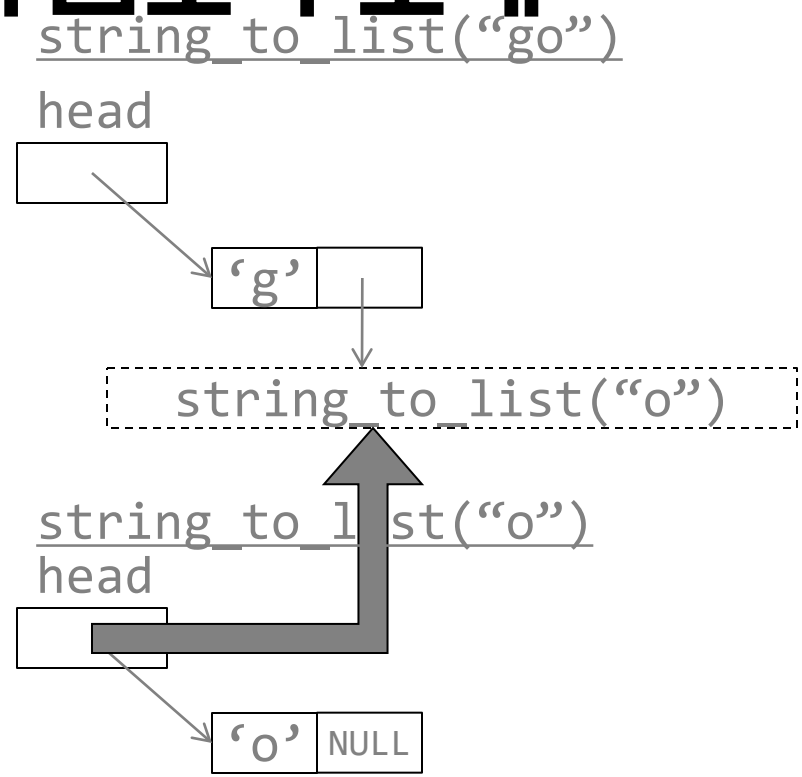


동적 메모리 할당과 자기 참조 구조체

프로그램 13.10

```
LINK string_to_list(char s[]){
    LINK  head;
    if (s[0] == '\0')
        return NULL;
    else {
        head = malloc(sizeof(ELEMENT));
        head -> d = s[0];
        head -> next = string_to_list(s + 1);
        return head;
    }
}
main() : h = string_to_list("go");
```

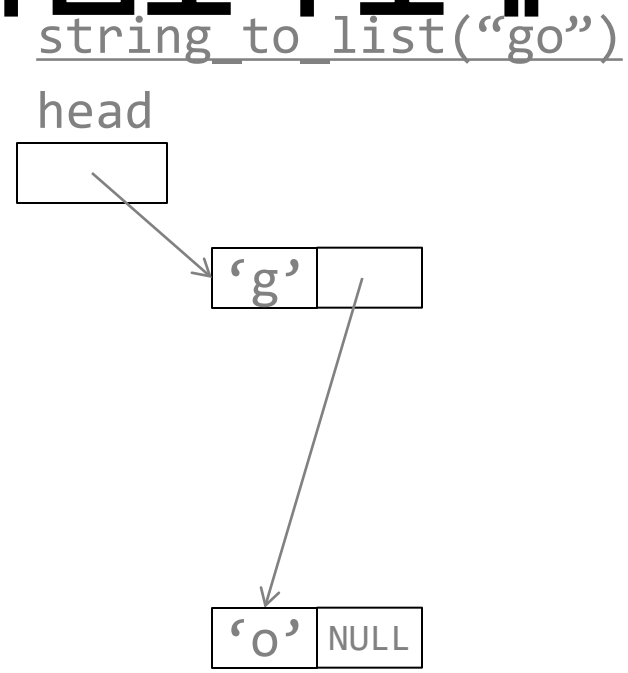




동적 메모리 할당과 자기 참조 구조체

프로그램 13.10

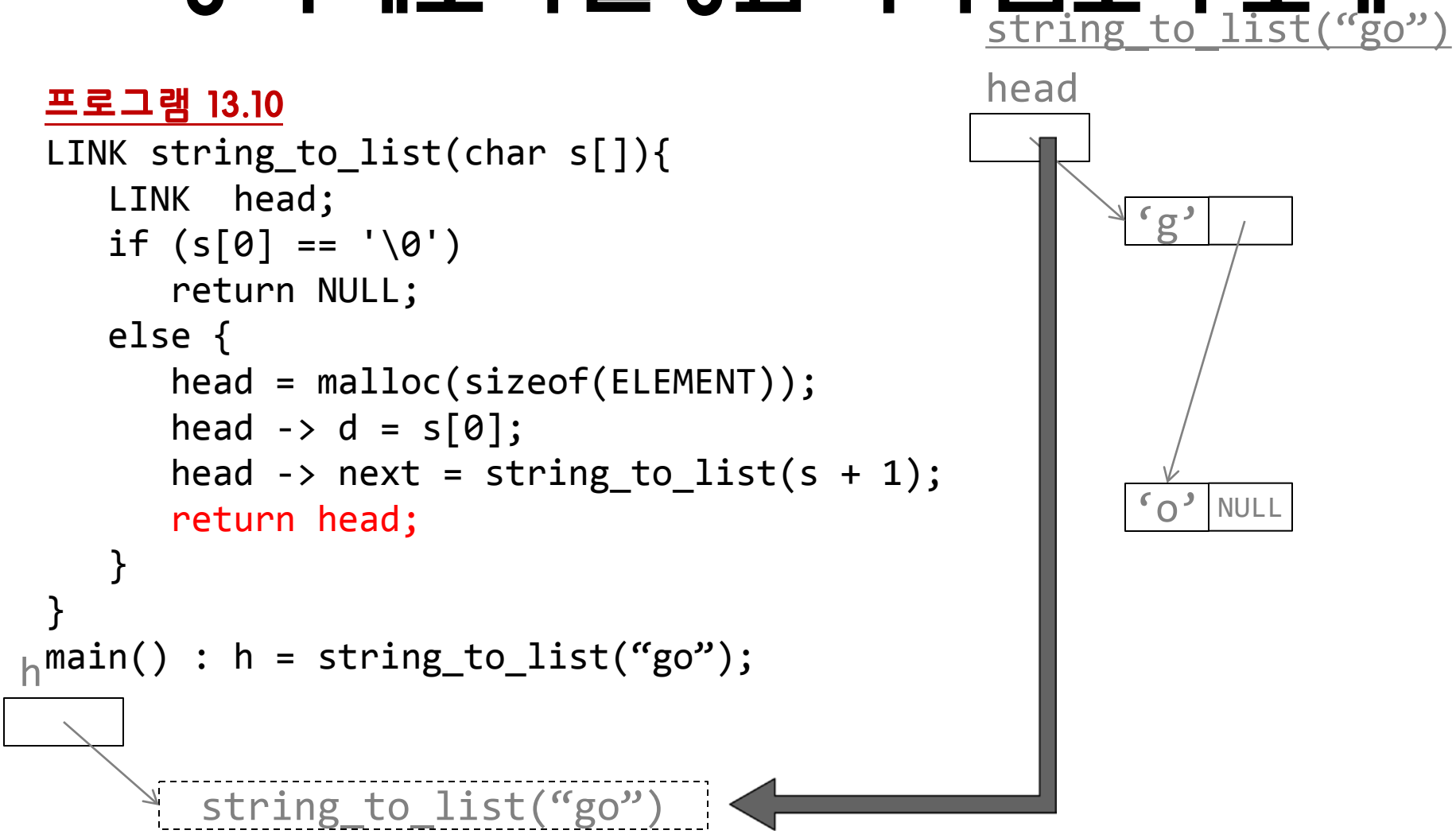
```
LINK string_to_list(char s[]){
    LINK  head;
    if (s[0] == '\0')
        return NULL;
    else {
        head = malloc(sizeof(ELEMENT));
        head -> d = s[0];
        head -> next = string_to_list(s + 1);
        return head;
    }
}
main() : h = string_to_list("go");
```



동적 메모리 할당과 자기 참조 구조체

프로그램 13.10

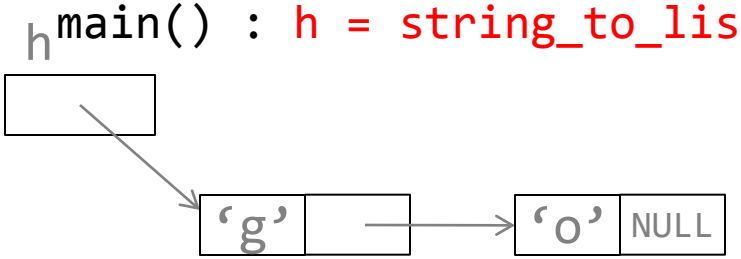
```
LINK string_to_list(char s[]){
    LINK  head;
    if (s[0] == '\0')
        return NULL;
    else {
        head = malloc(sizeof(ELEMENT));
        head -> d = s[0];
        head -> next = string_to_list(s + 1);
        return head;
    }
}
main() : h = string_to_list("go");
```



동적 메모리 할당과 자기 참조 구조체

프로그램 13.10

```
LINK string_to_list(char s[]){
   LINK  head;
   if (s[0] == '\0')
      return NULL;
   else {
      head = malloc(sizeof(ELEMENT));
      head -> d = s[0];
      head -> next = string_to_list(s + 1);
      return head;
   }
}
main() : h = string_to_list("go");
```



프로그램 결과

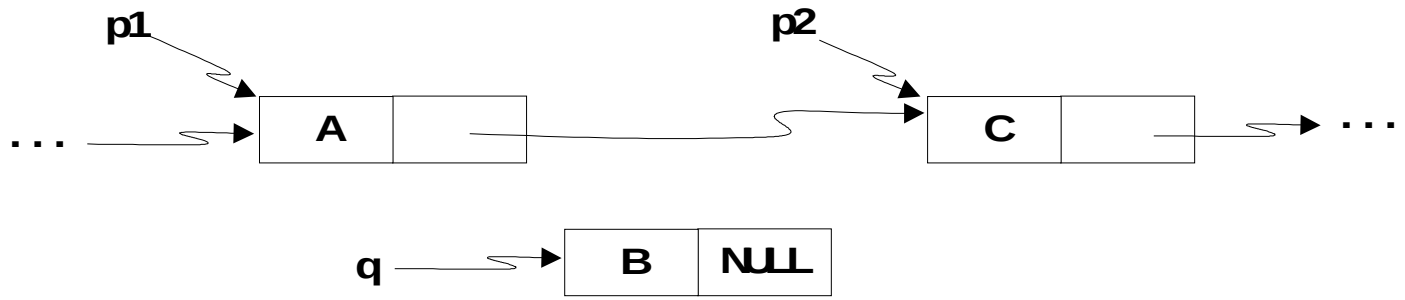
```
문자열 입력 : nice
변환 리스트 결과 :
n --> i --> c --> e --> NULL
```

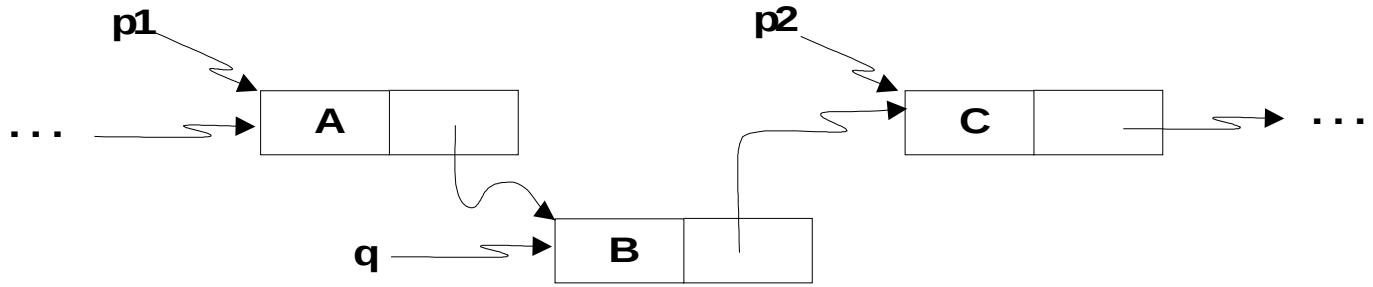
자기참조 구조체를 사용할 때 주의 사항

- 포인터를 다룰 때에는 함수를 만들어서 할 것

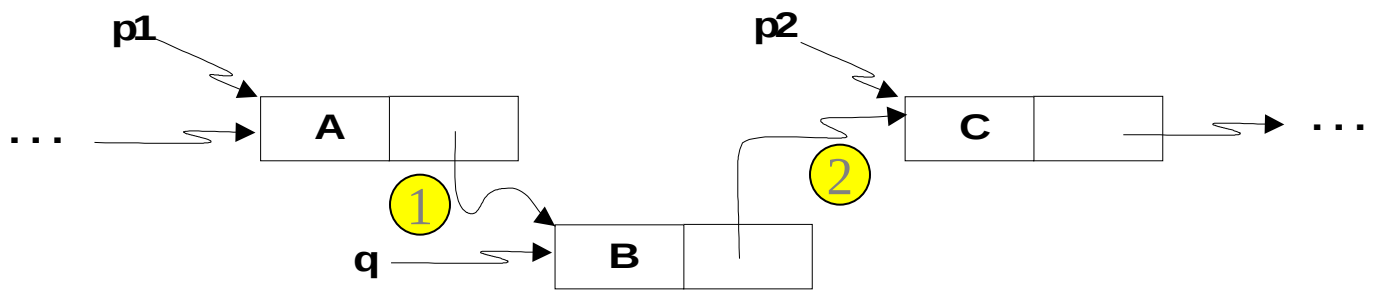
  InitNode(), InsertNode(), DeleteNode()

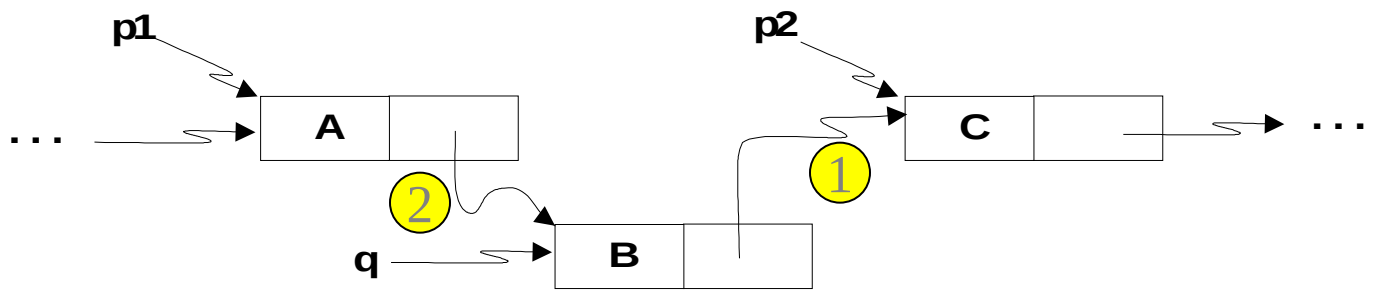
- 포인터를 다루는 함수를 만들기 전에 그림을 먼저 그릴 것





자기참조 구조체를 사용할 때 주의 사항





자기참조 구조체를 사용할 때 주의 사항

1. 포인터를 선언할 때 초기화 할 것

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

2. 포인터를 사용하기 전에 그 값을 확인할 것

```
if (p != NULL){
  . . .  // p 사용
}
else {
  . . .  // 오류처리
}
```

자기참조 구조체를 사용할 때 주의 사항

3. 할당받은 메모리 만큼만 값을 다루어야 함

```
p = (int *)malloc(N * sizeof(int));
while (isTrue()){
    p[i] = x;
    i++;
}
```

자기참조 구조체를 사용할 때 주의 사항

3. 할당받은 메모리 만큼만 값을 다루어야 함

```
p = (int *)malloc(N * sizeof(int));
while (isTrue() && (i < N)){
    p[i] = x;
    i++;
}
```

자기참조 구조체를 사용할 때 주의 사항

4. 포인터를 다루는 부분은 독립 함수로 만들 것

   InitNode(), InsertNode(), DeleteNode(), 등

5. 동적 메모리 할당을 받기 위해 malloc()이나 calloc() 문장을 만들 때 적절한 곳에 free() 문장도 같이 만들 것

6. free()를 사용하여 메모리를 반납한 후 포인터에 NULL 값을 배정할 것

```
free(p)
p = NULL;
```

7. 포인터를 다루기 전에 그림을 그려 볼 것

자기참조 구조체를 사용할 때 주의 사항

* 자신이 없으면 다른 대안을 찾아볼 것
  - 가변길이 배열

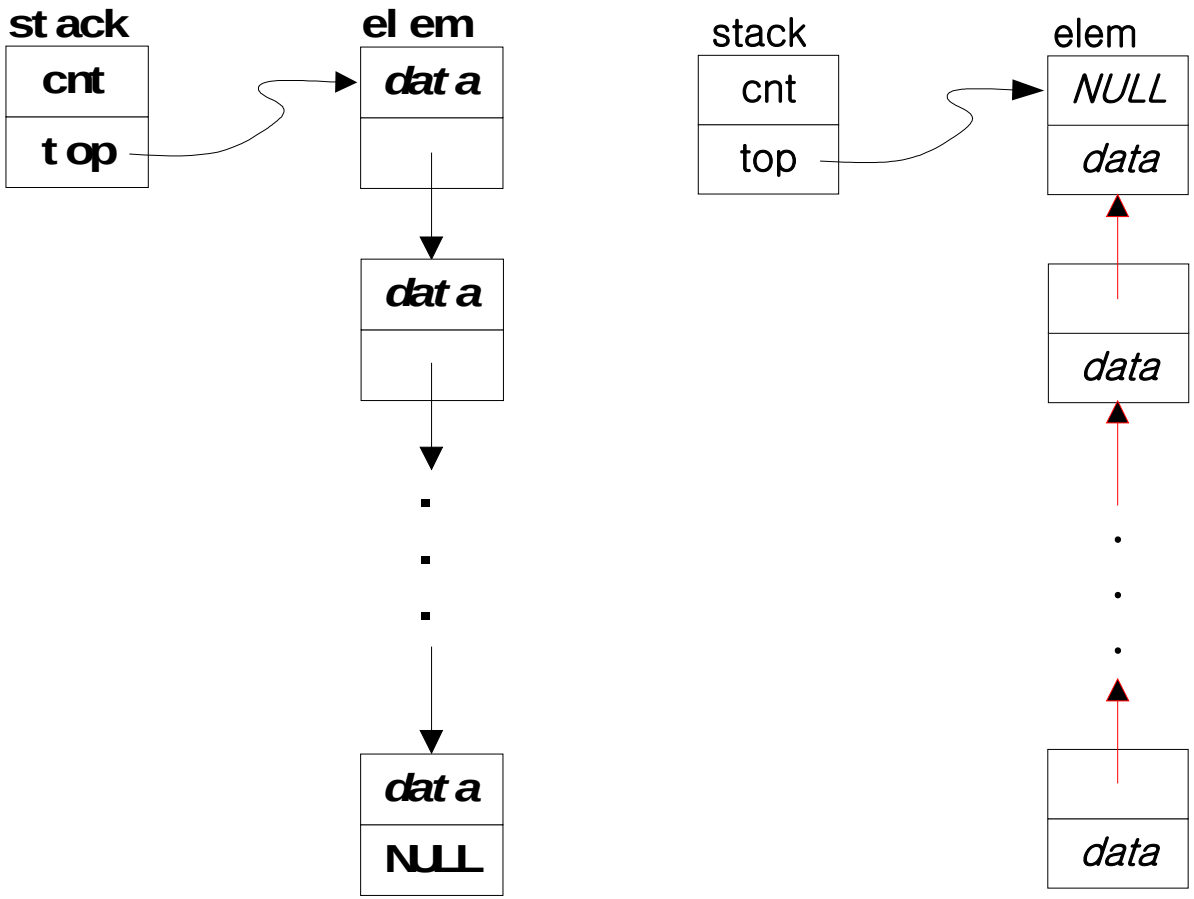
```
n = 10;
int a[n];
```

  - 플렉시블 배열 멤버(뒤에 설명)

```
struct s{
    int num;
    int a[];
}
struct s d1 = malloc(sizeof(struct s) + 40);
```

스택 – 선형 연결리스트로 구현

- 선형 연결 리스트로 구현한 스택



스택 구현 – stack.h

```
#include <stdio.h>
#include <stdlib.h>
#define  EMPTY    0
#define  FULL     10000
typedef  char   data;
typedef  enum {false, true}   boolean;
struct elem {              // element on the stack
   data        d;
   struct elem *next;
};
```

스택 구현 – stack.h

```
typedef   struct elem  elem;
struct stack {
   int   cnt;          // a count of the elements
   elem  *top;         // ptr to the top element
};
typedef   struct stack  stack;
void      initialize(stack *stk);
void      push(data d, stack *stk);
data      pop(stack *stk);
data      top(stack *stk);
boolean   empty(const stack *stk);
boolean   full(const stack *stk);
```

스택 구현 - 함수

```
#include "stack.h"
void initialize(stack *stk){
    stk -> cnt = 0;
    stk -> top = NULL;
}
void push(data d, stack *stk){
    elem  *p;
    p = malloc(sizeof(elem));
    p -> d = d;
    p -> next = stk -> top;
    stk -> top = p;
    stk -> cnt++;
}
```

스택 구현 - 함수

```
data pop(stack *stk){
    data  d;
    elem  *p;
    d = stk -> top -> d;
    p = stk -> top;
    stk -> top = stk -> top -> next;
    stk -> cnt --;
    free(p);
    return d;
}
```

스택 구현 - 함수

```
data top(stack *stk){
    return (stk -> top -> d);
}
boolean empty(const stack *stk){
    return ((boolean) (stk -> cnt == EMPTY));
}
boolean full(const stack *stk){
    return ((boolean) (stk -> cnt == FULL));
}
```

스택 - 검사 프로그램

```
#include "stack.h"
int main(void){
   char   str[ ] = "My name is joanna Kelly!";
   int    i;        stack  s;
   initialize(&s);            // initialize the stack
   printf(" In the string: %s\n", str);
   for (i = 0; str[i] != '\0'; ++i)
      if (!full(&s))
        push(str[i], &s);     // push a char on the stack
   printf("From the stack: ");
   while (!empty(&s))
      putchar(pop(&s));       // pop a char off the stack
   putchar('\n');
   return 0;
}
```

스택 - 검사 프로그램

- **출력**

```
In the string : My name is Jonna Kelly!

From the stack : !ylleK annoJ si eman yM
```

플렉시블 배열 멤버

- 두 개 이상의 멤버를 갖는 구조체 형 정의에서 크기가 명시되지 않은 마지막 배열 멤버
- C99
- 예

```
struct subject {
    int     num;
    int     grade[];
};
```

플렉시블 배열 멤버

- 플렉시블 구조체의 크기는 플렉시블 배열 멤버를 제외한 크기임
  - 플렉시블 구조체 변수를 선언하면 플렉시블 배열 멤버는 메모리 할당을 받지 않음

```
struct subject {
    int     num;
    int     grade[];
};
sizeof(struct subject) : 4
struct subject science;
science.num = 10;        // OK
science.grade[0] = 95;   // 오류
```

플렉시블 배열 멤버

- 사용하기 전에 플렉시블 배열 멤버를 위한 메모리 공간을 할당 받아야 함
- 사용 예

```
struct subject *science_p;
science_p =
    (struct subject *) malloc(sizeof(struct subject) + 40);
// grade[]를 위해 40 바이트 할당 → int 원소 10개
// 40 대신 sizeof(int) * 10 이 더 좋음
science_p -> num = 20;
science_p -> grade[4] = 85;  // OK
```

플렉시블 배열 멤버

프로그램 13.11

```
typedef struct subject {
   int    num;
   int    grade[];
} subject;
int main(void){
   int n, i, sum;
   subject *math_p;
   subject *c_p;
   printf("수학 과목 수강생 수 : ");
   scanf("%d", &n);
   math_p = (subject *) malloc(sizeof(subject) + sizeof(int) * n);
   math_p -> num = n;
   printf("수학 성적 %d 개를 입력하세요.\n", math_p -> num);
   for (i = 0; i < math_p -> num; i++){
      printf("%d 번째 성적 : ", i);
      scanf("%d", math_p -> grade + i);
   }
```

플렉시블 배열 멤버

프로그램 13.11

```
printf("C 과목 수강생 수 : ");
scanf("%d", &n);
c_p = (subject *) malloc(sizeof(subject) + sizeof(int) * n);
c_p -> num = n;
printf("C 성적 %d 개 입력하세요.\n", c_p -> num);
for (i = 0; i < c_p -> num; i++){
   printf("%d 번째 성적 : ", i);
   scanf("%d", c_p -> grade + i);
}

printf("\n수학 성적\n");
for (i = sum = 0; i < math_p -> num; i++){
   printf("%d번 학생 성적 : %d\n", i,  math_p -> grade[i]);
   sum += math_p -> grade[i];
}
```

플렉시블 배열 멤버

프로그램 13.11

```
    printf("총점 : %d, 평균 : %.2f\n", sum, sum/(float)math_p -> num);
    printf("\nC 성적\n");
    for (i = sum = 0; i < c_p -> num; i++){
        printf("%d번 학생 성적 : %d\n", i, c_p -> grade[i]);
        sum += c_p -> grade[i];
    }
    printf("총점 : %d, 평균 : %.2f\n", sum, sum/(float)c_p -> num);

    return 0;
}
```

프로그램 결과

```
수학 과목 수강생 수 : 3
수학 성적 3 개를 입력하세요.
0 번째 성적 : 78
1 번째 성적 : 88
2 번째 성적 : 92
C 과목 수강생 수 : 5
C 성적 5 개 입력하세요.
0 번째 성적 : 87
1 번째 성적 : 90
2 번째 성적 : 95
3 번째 성적 : 71
4 번째 성적 : 90
```

```
수학 성적
0번 학생 성적 : 78
1번 학생 성적 : 88
2번 학생 성적 : 92
총점 : 258, 평균 : 86.00

C 성적
0번 학생 성적 : 87
1번 학생 성적 : 90
2번 학생 성적 : 95
3번 학생 성적 : 71
4번 학생 성적 : 90
총점 : 433, 평균 : 86.60
```

라이브러리

- 라이브러리 목록은 ar 명령어로 볼 수 있음

  ```
  ar t /lib/libc.a
  ```

  - libc.a : 표준 C 라이브러리
  - /lib : 디폴트 라이브러리 디렉터리
- 프로그래머는 자신의 라이브러리를 만들 수 있음

라이브러리

- 라이브러리 만들기

함수 13.2(iszero.c)

```
int is_zero(int num)
{
    if (num)
        return 0;
    return 1;
}
```

1. .o 파일 만들기

```
$ gcc -c iszero.c
```

2. 라이브러리 만들기

```
$ ar ruv iszero.a iszero.o
$ ranlib iszero.a
```

라이브러리

프로그램 13.12 (main.c)

```
#include <stdio.h>
int   quotient, rem;       // 전역 변수 선언
int   divide(int, int);

int main(void){
   int a = 10, b = 3;
   if (divide(a, b))
      printf("0으로 나눌 수 없습니다.\n");
   else
      printf("%d / %d : 몫은 %d이고 나머지는 %d입니다.\n",
             a, b, quotient, rem);
   return 0;
}
```

라이브러리

프로그램 13.12 (divide.c)

```
int divide(int dividend, int divisor){
   extern int   quotient, rem;     // 전역 변수 참조 선언
   if (is_zero(divisor))
      return -1;
   quotient = dividend / divisor;
   rem = dividend % divisor;
   return 0;
}
```

라이브러리

- 사용자 라이브러리를 사용할 경우 컴파일할 때 사용자 라이브러리를 명시하면 됨

```
$ gcc -o divide main.c divide.c iszero.a
```

- main.c와 divide.c에서 호출하는 함수가 main.c와 divide.c에 정의되어 있지 않으면 iszero.a 라이브러리를 먼저 검색하고 없으면 표준 라이브러리에서 검색함

신호

- 프로세스 : 실행 중인 프로그램
- 신호는 프로세스에게 외부 환경의 변화나 오류를 전달할 때 사용
- 일반적으로 신호는 비정상적인 사건에 의해 생성되고 프로그램에게 전달되어 프로그램을 종료하게 함
  - 0으로 나누기
  - 잘못된 메모리 참조
  - control-c

신호

- <signal.h>에 신호와 관련된 매크로와 함수 정의
- 많이 다루어지는 신호관련 매크로

```
#define   SIGINT     2      // 인터럽트
#define   SIGILL     4      // 비정상 연산
#define   SIGFPE     8      // 부동 소수점 예외
#define   SIGKILL    9      // KILL
#define   SIGSEGV   11      // 세그먼트 위반
#define   SIGALRM   14      // 알람 클락
```

신호

- **신호를 다루는 함수 signal()**

```
void  (*signal(int sig, void (*func)(int)))(int);
```

  - sig : 신호 종류
  - func : 신호 처리기
  - sig와 func() 함수를 연결하는 함수
  - sig 신호가 발생하면 func()가 실행됨
- **예**

```
signal(2, func);
```

  - 2번 신호(control-c)가 발생하면 프로그램이 종료되는 것이 아니라 func() 함수가 호출됨
  - 보통 숫자 대신 매크로를 사용함

```
signal(SIGINT, func);
```

신호

프로그램 13.13

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

void cntrl_c_handler(int sig){
   char answer[4];
   printf("%d번 신호가 발생했습니다.\n", sig);
   printf("계속 하시겠습니까? (y/n) ");
   scanf("%s", answer);
   if ((answer[0] == 'n') || (answer[0] == 'N'))
      exit(1);
}
```

신호

프로그램 13.13

```
int main(void){
   int   i = 0;
   signal(SIGINT, cntrl_c_handler);   // SIGINT : control-c
   while (1) {
      printf("%4d", rand() % 1000);
      if ((++i % 5) == 0) {
         i = 0;
         putchar('\n');
      }
   }
   return 0;
}
```

프로그램 결과

```
  . . . .
 919 949 558 933 245
 846 518 203 752 403
 792 608 494 676 908 <control-c>
2 번 신호가 발생했습니다.
계속 하시겠습니까? (y/n) y
    . . . .
 521 192 309 458 219
 278  64 898 339 282
 772 272 167 924 291
 761 751  31 874 340
 319 423 962 902  49
 440 730 518 810 154 <control-c>
2 번 신호가 발생했습니다.
계속 하시겠습니까? (y/n) n
$
```

신호

- 신호 처리기로 사용할 수 있는 매크로

```
#define   SIG_DFL   ((void (*)(int)) 0)
  // 신호가 발생했을 때 디폴트 행동을 취하게 함
#define   SIG_IGN   ((void (*)(int)) 1)
  // 신호가 발생했을 때 무시하게 함
```

- 예

```
signal(SIGINT, SIG_IGN);
```

– 인터럽트가 발생하면 무시함

```
signal(SIGINT, SIG_DFL);
```

– 인터럽트가 발생하면 디폴트 동작을 다시 하게 함
– 인터럽트를 위한 디폴트 동작은 프로세스 종료 임

신호

- 신호 중 9번 신호(SIGKILL)는 사용자가 임의로 다룰 수 없음
  - signal() 함수로 SIGKILL 신호와 사용자 신호처리기를 연관시켜도 SIGKILL 신호가 발생하면 그 프로세스는 무조건 종료함