한빛아카데미 Hanbit Academy, Inc.

C 언어 프로그래밍

Part 02. C 언어 기본 학습

Chapter 09. 저장소 분류

목차

학습목표

- 객체의 가시 범위와 생명 주기의 개념을 이해한다.
- 저장소 분류의 개념을 이해한다.
- 자동 저장소 분류, 전역 저장소 분류, 정적 저장소 분류의 개념과 특징을 살펴본다
- 전역, 정적, 지역 객체의 초기화 과정을 살펴본다.

01

객체의 가시 범위와 생명 주기

01. 객체의 가시 범위와 생명 주기

I. 객체의 가시 범위

[코드 9-1] 함수 안에서 사용한 객체의 사용

```
01  #include <stdio.h>
02
03  void func( )
04  {
05     int b = 3;
06     int brr[1] = { 4 };
07     printf("%d", a);
08     printf("%d", arr[0]);
09  }
10
11  int main( )
12  {
13        int a = 1;
14        int arr[1] = { 2 };
15        func();
16        printf("%d", b);
17        printf("%d", brr[0]);
18  }
```

오류

오류

01. 객체의 가시 범위와 생명 주기

I. 객체의 가시 범위

- 지역 객체의 가시 범위

[코드 9-2] 변수가 모든 함수에서 사용 가능하다면 1

```
01  #include <stdio.h>
02
03  int not(int arg)
04  {
05      int a = !arg;
06      return a;
07  }
08
09  int main()
10  {
11      int a = 0;
12      int b = not(a);
13
14      printf("%d, %d", a, b);
15  }
```

```
0, 1
```

01. 객체의 가시 범위와 생명 주기

I. 객체의 가시 범위

[코드 9-3] 변수가 모든 함수에서 사용 가능하다면 2

```
01  #include <stdio.h>
02
03  void f( )
04  {
05      int a = 1;
06  }
07
08  void g( )
09  {
10      int a = 2;
11  }
12
13  int main( )
14  {
15      f();
16      g();
17      printf("%d", a);
18  }
```

01. 객체의 가시 범위와 생명 주기

I. 객체의 가시 범위

- 전역 객체의 가시 범위

[코드 9-3] 전역 객체(변수, 배열)

```
01  #include <stdio.h>
02
03  int g = 1;
04  int garr[1] = { 1 };
05
06  void func( )
07  {
08     g++;
09     garr[0]++;
10  }
11
12  int main( )
13  {
14     printf("Before: %d, %d\r\n", g, garr[0]);
15     func();
16     printf("After: %d, %d", g, garr[0]);
17  }
```

```
Before: 1, 1
After: 2, 2
```

01. 객체의 가시 범위와 생명 주기

I. 객체의 가시 범위

- **전역 객체의 가시 범위**

표 9-1 지역 객체와 전역 객체의 가시 범위

	지역 객체	전역 객체
선언 위치	함수 내부	함수 외부
가시 범위	해당 블록	프로그램 전체

01. 객체의 가시 범위와 생명 주기

II. 객체의 생명 주기

- **객체의 생성과 소멸의 개념**
  - 객체의 생성 : 메모리의 영역을 마련하여 이름을 붙여주는 것
  - 객체의 소멸 : 객체와 연결된 메모리 영역에서 이름이 해제되는 것
  - 객체의 생명 주기 : 메모리 영역이 객체와 연결되는 순간부터 연결이 끊길 때까지 기간

01. 객체의 가시 범위와 생명 주기

II. 객체의 생명 주기

[코드 9-3] 객체의 생명 주기 – 메모리 재활용

```
01  #include <stdio.h>
02
03  void f( )
04  {
05     int a[1] = { 7 };
06     printf("%d\r\n", a[0]);
07  }
08
09  void g( )
10  {
11     int b[1];          // 초기화 안 됨
12     printf("%d\r\n", b[0]);
13  }
14
15  int main( )
16  {
17     f();
18     g();
19  }
```

```
7
7
```

01. 객체의 가시 범위와 생명 주기

II. 객체의 생명 주기

- 함수 호출에 따른 스택 프레임의 변화

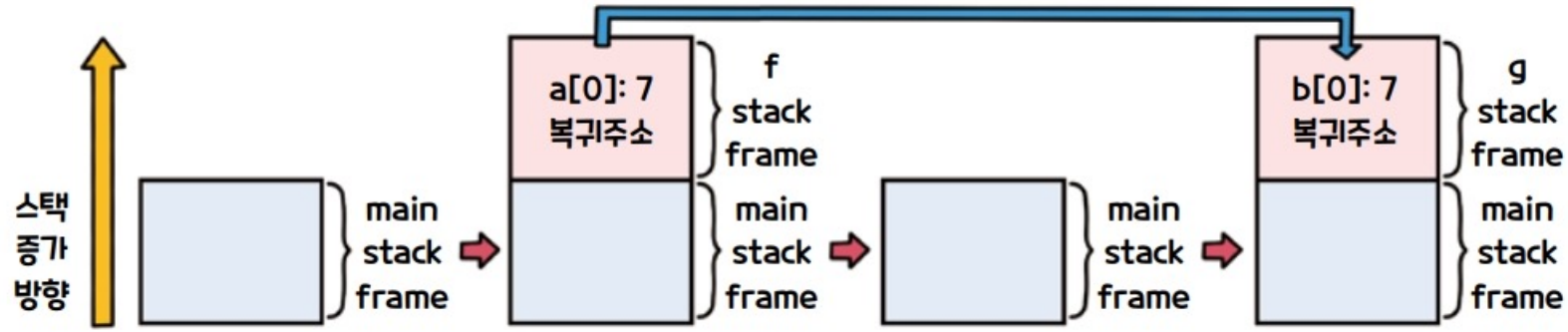

그림 9-1 함수 호출과 스택 프레임

01. 객체의 가시 범위와 생명 주기

확인문제1

1. 다음 빈칸에 들어갈 단어를 채우시오.

객체가 사용될 수 있는 범위를 [ ] 범위라고 한다. 지역 객체의 [ ] 범위는 객체가 선언된 함수이고, [ ] 객체의 [ ] 범위는 프로그램 전체이다.

2. 다음 빈칸에 들어갈 단어를 채우시오.

객체의 생성부터 소멸까지 전 기간을 [ ]라고 한다. 지역 객체의 [ ]는 함수의 호출부터 반환까지이며, [ ] 객체의 [ ]는 프로그램 시작부터 종료까지이다.

01. 객체의 가시 범위와 생명 주기

LAB 9-1 알파벳 배열에 A-Z 채우기

다음 실행 결과를 참고하여 입력받은 수를 보관하면서 이전에 보관된 수는 출력하는 프로그램을 작성해봅시다. 단, 최초 보관된 수는 0이며, 0을 보관할 경우 프로그램을 종료합니다.

```
보관할 수를 입력하세요 > 3
이전 보관된 수: 0
보관할 수를 입력하세요 > 7
이전 보관된 수: 3
보관할 수를 입력하세요 > 0
이전 보관된 수: 7
```

1. 보관된 수는 전역 변수 g_Number로 만든다.
2. 수를 보관하고 이전 보관된 수를 반환하는 함수 SaveNumber를 작성한다.
3. do~while문을 이용하여 입력된 수가 0이 아닌 경우 입력을 반복한다.

01. 객체의 가시 범위와 생명 주기

LAB 9-1 정답

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int g_Number = 0;
05
06  int SaveNumber(int n)
07  {
08      int r = g_Number;
09      g_Number = n;
10      return r;
11  }
12
13  int main( )
14  {
15      int n;
16
```

01. 객체의 가시 범위와 생명 주기

LAB 9-1 정답

```
17      do
18      {
19          printf("보관할 수를 입력하세요 > ");
20          scanf("%d", &n);
21
22          printf("이전 보관된 수: %d\r\n", SaveNumber(n));
23      }
24      while(n);
25  }
```

02

자동 저장소 분류

02. 자동 저장소 분류

I. 자동 저장소 분류의 개념

- **자동(auto) 저장소 분류**
  - 함수의 매개변수나 지역 객체를 위한 분류
  - 자동 저장 분류의 가시 범위는 함수 안의 블록으로 좁혀짐
  - 복합문은 복합문을 포함할 수 있다는 점을 주의해야 함

02. 자동 저장소 분류

I. 자동 저장소 분류의 개념

[코드 9-6] 가시 범위의 중첩

```
01  void func(int arg)
02  {
03     arg = 1;
04     int a = 2;
05
06     {
07        int b = 3;
08     }
09
10     b = 4;   <- 오류
11  }
12
13  int main( )
14  {
15     func(0);
16  }
```

02. 자동 저장소 분류

II. 가시 범위의 특징

[코드 9-7] 가시 범위와 우선권

```
01  #include <stdio.h>
02
03  int main( )
04  {
05     int a = 1;        // 첫 번째 a
06
07     {
08        int a = 2;     // 두 번째 a
09        printf("%d\r\n", a);
10     }
11
12     printf("%d", a);
13  }
```

```
2
1
```

02. 자동 저장소 분류

확인문제2

같은 이름을 가진 서로 다른 객체의 가시 범위가 포함 관계에 있을 경우 작은 가시 범위의 객체가 우선권을 갖는 이유를 서술하시오.

02. 자동 저장소 분류

LAB 9-2 **같은 이름이지만 서로 다른 제어 변수 사용하기**

같은 이름이지만 서로 다른 제어 변수를 사용하는 이중 for문으로 구구단을 출력하는 프로그램을 작성해봅시다.

두 개의 for문은 같은 이름의 제어 변수 i를 사용하지만 가시 범위가 다르기 때문에 가시 범위가 중복되는 곳이 아니라면 개별적으로 사용할 수 있다.

02. 자동 저장소 분류

LAB 7-2 정답

```
01  #include <stdio.h>
02
03  int main()
04  {
05      int r;
06
07      for(int i = 2; i < 10; i++)
08      {
09          r = i;
10          for(int i = 1; i < 10; i++)
11          {
12              printf("%d * %d = %d\r\n", r, i, r * i);
13          }
14          printf("\r\n");
15      }
16  }
```

03

전역 저장소 분류

03. 전역 저장소 분류

I. 전역 저장소 분류의 개념

- **전역(extern) 저장소 분류**
  - 가시 범위가 프로그램 전체이고, 생명 주기는 프로그램의 시작부터 끝인 객체들의 분류

03. 전역 저장소 분류

I. 전역 저장소 분류의 개념

- 전역(extern) 저장소 분류

[코드 9-8] 전역 저장소 분류

```
01  void func( )
02  {
03      g = 2;      <-- 오류
04  }
05
06  int g;
07
08  int main( )
09  {
10      g = 1;          // 문제 없음
11      func();
12  }
```

03. 전역 저장소 분류

I. 전역 저장소 분류의 개념

- 전역 객체의 선언 위치

[코드 9-9] 다른 소스 파일의 전역 변수 선언 Function.c

```
01  int g = 1;
```

Main.c

```
01  #include <stdio.h>
02
03  int main( )
04  {
05      printf("%d", g);
06  }
```

오류

03. 전역 저장소 분류

II. extern 지정자

[코드 9-10] extern을 이용한 전역 객체의 순수 선언 Function.c

```
01  int g = 1;
```

Main.c

```
01  #include <stdio.h>
02
03  extern int g;
04
05  int main( )
06  {
07     printf("%d", g);      // 문제 해결
08  }
```

03. 전역 저장소 분류

II. extern 지정자

- 주의 사항 1. 링크 오류 발생

[코드 9-11] 전역 객체의 순수 선언만 할 경우

```
01  #include <stdio.h>
02
03  extern int h;
04
05  int main( )
06  {
07      printf("%d", h);
08  }
```

오류

03. 전역 저장소 분류

II. extern 지정자

- 주의 사항 2. 순수하게 선언만 할 때는 초기화가 불가능

[코드 9-13] extern과 초기화 Function.c

```
01  int g = 1;
```

Main.c

```
01  #include <stdio.h>
02
03  extern int g = 2;
04
05  int main( )
06  {
07      printf("%d", g);
08  }
```

오류

03. 전역 저장소 분류

확인문제3

1. 지역 객체 대비 전역 객체의 단점 두 가지를 설명하시오.

2. 다음 빈칸에 공통으로 들어갈 단어를 채우시오.

전역 변수의 순수 선언이란 변수를 [ ] 하지 않고, 이미 [ ] 된 변수의 이름을 알리는 역할을 한다.

03. 전역 저장소 분류

LAB 9-3 입출금 처리 프로그램

다음 출력처럼 입출금 처리 프로그램을 작성해봅시다. 단, 입출금액이 양수이면 입금, 음수이면 출금으로 입출금 여부를 출력합니다. 단, 입출금액이 0인 경우 프로그램을 종료합니다.

```
입출금액을 입력하세요 > 3000
입금: 3000원, 잔고: 3000원
입출금액을 입력하세요 > 7000
입금: 7000원, 잔고: 10000원
입출금액을 입력하세요 > -2000
출금: 2000원, 잔고: 8000원
입출금액을 입력하세요 > 0
```

1. 잔고는 전역 변수 g_Balance로 만든다.
2. 입금과 출금을 담당할 함수 Deposit과 Withdraw를 각각 정의한다.
3. 입력 값의 양수, 음수, 0 여부를 판별하여 각각 입출금 및 종료 처리를 한다.

03. 전역 저장소 분류

LAB 9-3 정답

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int g_Balance = 0;
05
06  void Deposit(int amount)
07  {
08      g_Balance += amount;
09      printf("입금: %d원, 잔고: %d원\r\n", amount, g_Balance);
10   }
11
12  void Withdraw(int amount)
13  {
14      g_Balance -= amount;
15      printf("출금: %d원, 잔고: %d원\r\n", amount, g_Balance);
16  }
17
```

03. 전역 저장소 분류

LAB 9-3 정답

```
18  int main( )
19  {
20      int amount;
21
22      while(1)
23      {
24          printf("입출금액을 입력하세요 > ");
25          scanf("%d", &amount);
26
27          if(amount > 0)
28              Deposit(amount);
29          else if(amount < 0)
30              Withdraw(-amount);
31          else
32              break;
33      }
34  }
```

04

정적 저장소 분류

04. 정적 저장소 분류

I. 정적 저장소 분류의 개념

- **정적(static) 저장소 분류의 특징**
  - 생명 주기는 전역 저장소 분류와 같지만 가시 범위는 유동적임

II. static 지정자

- 정적 개체를 지정하기 위함
- 선언된 위치에 따라 정적 지역 객체 혹은 정적 전역 객체로 나뉨

표 9-2 객체의 가시 범위와 생명 주기

	지역 객체	전역 객체	정적 객체	
			정적 지역 객체	정적 전역 객체
가시 범위	함수 블록	프로그램 전체	함수 블록	소스 파일
생명 주기	함수 호출~반환	프로그램 시작~종료	프로그램 시작~종료	프로그램 시작~종료

04. 정적 저장소 분류

III. 정적 지역 객체

- static을 지정하여 지역 객체로 선언한 것
- 프로그램 실행 내내 살아있으면서 함수가 호출될 때마다 재활용됨

[코드 9-14] 정적 지역 변수

```
01  #include <stdio.h>
02
03  void func( )
04  {
05     static int s = 0;
06     s++;
07     printf("%d\r\n", s);
08  }
09
10  int main( )
11  {
12     func();   // 1 출력
13     func();   // 2 출력
14  }
```

```
1
2
```

04. 정적 저장소 분류

III. 정적 지역 객체

[코드 9-14] 정적 지역 변수

```
01  #include <stdio.h>
02
03  void func( )
04  {
05     static int s = 0;
06     s++;
07
08     {
09        static int s = 1;
10        s++;
11        printf("%d ", s);
12     }
13
14     printf("%d\r\n", s);
15  }
16
```

```
17  int main()
18  {
19     func();
20     func();
21  }
```

```
2 1
3 2
```

04. 정적 저장소 분류

IV. 정적 전역 객체

- static을 지정하여 전역 객체를 선언한 것
- 일반 전역 객체처럼 extern 지정자를 사용할 수 없음

[코드 9-16] 정적 전역 변수 **Function.c**

```
01  static int sg = 1;
```

Main.c

```
01  #include <stdio.h>
02
03  static int sg = 2;
04
05  int main( )
06  {
07     printf("%d", sg);
08  }
```

```
2
```

04. 정적 저장소 분류

IV. 정적 전역 객체

[코드 9-17] 전역 변수와 정적 전역 변수의 우선순위

Function.c

```
01  int sg = 1;
```

Main.c

```
01  #include <stdio.h>
02
03  static int sg = 2;
04
05  int main( )
06  {
07     printf("%d", sg);
08  }
```

```
2
```

04. 정적 저장소 분류

확인문제4

1. 다음 빈칸에 공통으로 들어갈 단어를 채우시오.

   지역 객체에 [ ]을 붙이면 정적 지역 객체가 되고, 전역 객체에 [ ]을 붙이면 정적 전역 객체가 된다. 따라서 [ ]이 지정된 객체는 정적 저장소 분류에 속한다.

2. 전역 객체와 비교하여 정적 지역 객체의 장점은 무엇인가?

3. 전역 객체와 비교하여 정적 전역 객체의 장점은 무엇인가?

04. 정적 저장소 분류

LAB 9-4 개선된 입출금 처리 프로그램

[LAB 9-3]에서 만들었던 입출금 처리 프로그램에 기능을 추가해봅시다. 입출금마다 총입금액과 총출금액도 같이 출력합니다.

```
입출금액을 입력하세요 > 7000
입금: 7000원, 총입금액: 7000원, 잔고: 7000원
입출금액을 입력하세요 > -2000
출금: 2000원, 총출금액: 2000원, 잔고: 5000원
입출금액을 입력하세요 > 5000
입금: 5000원, 총입금액: 12000원, 잔고: 10000원
입출금액을 입력하세요 > -3000
출금: 3000원, 총출금액: 5000원, 잔고: 7000원
입출금액을 입력하세요 > 0
```

1. 잔고는 전역 변수 g_Balance로 만든다.
2. 총입금액과 총출금액은 함수 Deposit과 Withdraw의 정적 지역 변수 s_Total로 만든다. s_Total은 이름이 같지만 각각의 함수에서만 사용 가능한 개별적인 변수로서 입금 혹은 출금별로 누적 금액을 저장할 수 있다.

04. 정적 저장소 분류

LAB 9-4 정답

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <studio.h>
03
04  int g_Balance = 0;
05
06  void Deposit(int amount)
07  {
08      static int s_Total = 0;
09
10      g_Balance += amount;
11      s_Total += amount;
12      printf("입금: %d원, 총입금액: %d원, 잔고: %d원\r\n",
13                      amount, s_Total, g_Balance);
14  }
15
```

04. 정적 저장소 분류

LAB 9-4 정답

```
16  void Withdraw(int amount)
17  {
18      static int s_Total = 0;
19
20      g_Balance -= amount;
21      s_Total += amount;
22      printf("출금: %d원, 총출금액: %d원, 잔고: %d원\r\n",
23                      amount, s_Total, g_Balance);
24  }
25
26  int main( )
27  {
28      int amount;
29
30      while(1)
```

04. 정적 저장소 분류

LAB 9-4 정답

```
31      {
32          printf("입출금액을 입력하세요 > ");
33          scanf("%d", &amount);
34
35          if(amount > 0)
36              Deposit(amount);
37          else if(amount < 0)
38              Withdraw(-amount);
39          else
40              break;
41      }
42  }
```

05

객체의 초기화

05. 객체의 초기화

I. 전역, 정적 객체의 초기화

- **전역 객체의 초기화**
  - 명시적으로 초기화를 하지 않을 경우 자동으로 0으로 초기화됨

[코드 9-18] 정적 지역 객체의 가시 범위 – 블록

```
01  #include <stdio.h>
02
03  int g1;         // 0 초기화
04  int g2 = 1;     // 1 초기화
05
06  int main( )
07  {
08     printf("g1:%d, g2:%d", g1, g2);
09  }
```

```
g1:0, g2:1
```

05. 객체의 초기화

I. 전역, 정적 객체의 초기화

- 전역 객체의 초기화

[코드 9-19] 정적 변수의 비상수 초기화

```
01  int get( )
02  {
03      return 1;
04  }
05
06  int g1 = 1;
07  int g2 = g1;
08  int g3 = get();
09
10  int main( )
11  {
12  }
```

오류

05. 객체의 초기화

I. 전역, 정적 객체의 초기화

- **정적 객체의 초기화**
  - 명시적으로 초기화를 하지 않을 경우 프로그램이 시작할 때 자동으로 0으로 초기화됨
  - 선언과 동시에 초기화할 경우 초기값은 오직 상수여야 함

05. 객체의 초기화

II. 지역 객체의 초기화

- 명시적으로 초기화를 하지 않을 경우 자동으로 초기화되지 않음
- 지역 객체가 차지하고 있는 메모리 영역의 이전 상태를 그대로 사용함
- 초기화되지 않거나 특정 값이 대입되지 않은 상태를 미정의 값이라고 함

[코드 9-20] 지역 변수의 초기화

```
01  #include <stdio.h>
02
03  int main
04  {
05      int i;
06      printf("%d\r\n", i);
07
08      int arr[1];
09      printf("%d", arr[0]);
10  }
```

오류

05. 객체의 초기화

확인문제5

1. 전역 저장소 분류와 정적 저장소 분류의 공통점을 설명하시오.

2. 다음 빈칸에 들어갈 단어를 채우시오.

미정의 값이란 변수가 [ ] 되지 않거나 [ ] 된 적이 없는 변수의 상태를 나타낸다.

05. 객체의 초기화

LAB 9-5 덧셈이 누적되는 프로그램

다음 실행 결과를 참고하여 입력하는 수들의 덧셈 과정을 보여주는 프로그램을 작성해봅시다. 단, 입력하는 수가 0인 경우 프로그램을 종료합니다.

```
더할 수를 입력하세요 > 3
3 = 3
더할 수를 입력하세요 > 7
3 + 7 = 10
더할 수를 입력하세요 > -2
3 + 7 - 2 = 8
더할 수를 입력하세요 > 0
```

1. 더할 수가 입력될 때마다 0이면 입력 반복문을 빠져나가고, 0이 아니면 add 함수를 호출하여 출력 처리를 진행한다.
2. add 함수 안에는 정적 지역 배열과 정적 지역 변수를 선언하여 입력된 수들을 모두 배열에 보관하고 인덱스를 관리한다.
3. add 함수는 전역 변수인 g_Sum에 합계를 누적시키고, for문을 통해서 지금까지 입력된 수들의 양수, 음수 여부에 따라서 더하기(+)나 빼기(-) 기호를 출력하면서 계산 과정을 나열한다.

05. 객체의 초기화

LAB 9-5 정답

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <studio.h>
03
04  int g_Sum;
05
06  void add(int n)
07  {
08      static int s_Index;
09      static int s_Arr[1024];
10
11      g_Sum += n;
12      s_Arr[s_Index] = n;
13      s_Index++;
14
15      printf("%d ", s_Arr[0]);
```

05. 객체의 초기화

LAB 9-5 정답

```
16      for(int i = 1; i < s_Index; i++)
17      {
18         if(s_Arr[i] > 0)
19            printf("+ %d ", s_Arr[i]);
20         else if(s_Arr[i] < 0)
21            printf("+ - %d ", -s_Arr[i]);
22      }
23
24      printf("= %d\r\n", g_Sum);
25   }
26
27   int main( )
28   {
29      while(1)
30      {
```

05. 객체의 초기화

LAB 9-5 정답

```
31              int n;
32              printf("더할 수를 입력하세요 > ");
33              scanf("%d", &n);
34
35              if(n)
36                  add(n);
37              else
38                  break;
39          }
40      }
```

[실전예제]

양수와 음수의 덧셈 누적하기

[실전예제] 양수와 음수의 덧셈 누적하기

[문제]

[LAB 9-5]에서 작성한 덧셈 과정을 보여주는 프로그램의 기능을 개선하여 다음과 같이 양수와 음수를 각각 모아서 보여주는 방식이 되도록 작성해봅시다.

실행 결과

```
더할 수를 입력하세요 > 7
( 7 ) = 7
더할 수를 입력하세요 > -2
( 7 ) - ( 2 ) = 5
더할 수를 입력하세요 > 5
( 7 + 5 ) - ( 2 ) = 10
더할 수를 입력하세요 > -4
( 7 + 5 ) - ( 2 + 4 ) = 6
더할 수를 입력하세요 > 0
```

[실전예제] 문자 배열 합치기

[해결]

1. 양수와 음수를 분리하여 처리하기 위하여 양수용 배열과 인덱스, 음수용 배열과 인덱스를 정적 지역 배열과 정적 지역 변수로 각각 선언한다.

2. 단 한 번이라도 양수나 음수가 입력된 경우에 한하여 괄호를 출력한다.

[실전예제] 문자 배열 합치기

[해결]

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int g_Sum;
05
06  void add(int n)
07  {
08      static int s_plusIndex;
09      static int s_plusArr[256];
10
11      static int s_minusIndex;
12      static int s_minusArr[256];
13
14      g_Sum += n;
15
16      if(n > 0)
17      {
18          s_plusArr[s_plusIndex] = n;
```

[실전예제] 문자 배열 합치기

[해결]

```
19          s_plusIndex++;
20      }
21      else if(n < 0)
22      {
23          s_minusArr[s_minusIndex] = n;
24           s_minusIndex++;
25      }
26
27      if(s_plusIndex > 0)
28      {
29          printf("( %d ", s_plusArr[0]);
30          for(int i = 1; i < s_plusIndex; i++)
31              printf("+ %d ", s_plusArr[i]);
32
33          printf(")");
34      }
35
36      if(s_minusIndex > 0)
```

[실전예제] 문자 배열 합치기

[해결]

```
37      {
38          printf(" - ( %d ", -s_minusArr[0]);
39          for(int i = 1; i < s_minusIndex; i++)
40              printf("+ %d ", -s_minusArr[i]);
41
42          printf(")");
43      }
44
45      printf(" = %d\r\n", g_Sum);
46  }
47
48  int main()
49  {
50      while(1)
51      {
52          int n;
53          printf("더할 수를 입력하세요 > ");
54          scanf("%d", &n);
```

[실전예제] 문자 배열 합치기

[해결]

```
55
56          if(n)
57              add(n);
58          else
59              break;
60      }
61  }
```

Thank you!

한빛아카데미 Hanbit Academy, Inc.