C Language

Lecture 11. 동적 할당

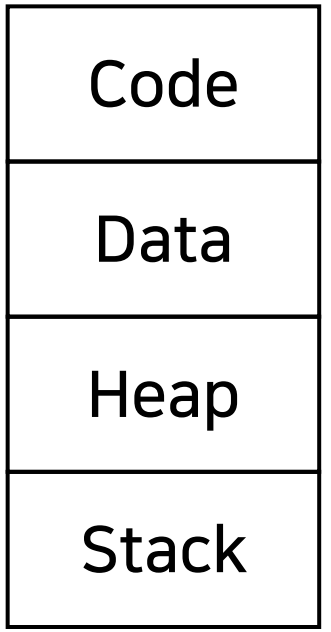
BAE SEKANG(배세강)

Contents

Memory Allocation

♣ Memory Structure

- Code 영역: 프로그램의 실행 코드가 저장되는 영역
- Data 영역: 전역 변수, 정적 변수가 저장되는 영역
- Heap 영역: 프로그램 실행 중 동적으로 할당해 사용하는 영역
- Stack 영역: 지역 변수, 매개변수 등이 저장되는 영역



Memory Allocation

♣ Static Allocation

- 프로그램 실행 전 크기가 결정되는 메모리 할당 방식
- 전역 변수, 정적 변수, 고정 크기 배열 등

```
#include <stdio.h>

int a = 5;
static int b = 10;

int main(void) {
    printf("%d %d₩n", a, b);
    return 0;
}
```

Memory Allocation

♣ Automatic Allocation

- 함수가 호출될 때 stack에 메모리가 할당되고, 함수가 끝나면 자동으로 해제되는 방식

```
#include <stdio.h>

void func(void) {
    int a = 10;
    int arr[5];
}

int main() {
    func();
    return 0;
}
```

Memory Allocation

♣ Dynamic Allocation

- 프로그램 실행 중 필요한 만큼 heap 메모리를 요청하여 할당하는 방식

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

int main() {
    int n;
    scanf("%d", &n);
    int *arr = malloc(sizeof(int) * n);
    for (int i = 0; i < n; i++) scanf("%d", &arr[i]);
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);

    free(arr);
    return 0;
}
```

Allocation Function

♣ stdlib.h

- 동적 할당, 프로그램 종료, 난수 생성, 정렬 등의 기능을 갖춘 C 표준 라이브러리

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

int compare_int(const void *a, const void *b) {
    int x = *(const int *)a;
    int y = *(const int *)b;
    return x - y;
}

int main(void) {
    char str[] = "5"; int n = atoi(str);
    int *arr = malloc(sizeof(int) * n);
    if (arr == NULL) return 1;
    for (int i = 0; i < n; i++) arr[i] = rand() % 100;
    qsort(arr, n, sizeof(int), compare_int);
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);

    free(arr);
    return 0;
}
```

Allocation Function

♣ malloc()

- heap 영역에 원하는 크기만큼 메모리를 할당하는 함수
- void *malloc(size_t size);

```
#include <stdlib.h>

int fixed_arr[100]; // 고정 크기의 배열

int n;
scanf("%d", &n);
int *arr = (int *)malloc(sizeof(int) * n); // 동적 할당
```

- *arr: 할당된 메모리의 시작 주소를 저장할 포인터
- sizeof(int) * n: 정수 n개를 저장할 수 있는 크기
- (int *): void *를 int *로 형 변환

Allocation Function

♣ NULL check

- malloc은 할당 실패시 NULL을 반환한다.
- malloc으로 할당된 값은 초기화되지 않는다. (garbage value)

```
// NULL checking

if (arr == NULL) {
    printf("Memory allocation failed₩n");
    return 1;
}
```

Allocation Function

♣ Example: malloc()

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

int main(void) {
    int n;
    int *arr;
    printf("Enter size: ");
    scanf("%d", &n);
    arr = (int *)malloc(sizeof(int) * n);
    if (arr == NULL) {
        printf("Memory allocation failed₩n");
        return 1;
    }

    for (int i = 0; i < n; i++) arr[i] = i + 1;
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    free(arr);
    return 0;
}
```

Allocation Function

♣ realloc()

- 이미 할당한 동적 배열의 크기를 실행 중에 바꾸는 함수
- void *realloc(void *ptr, size_t new_size);

```
// 처음에는 정수 3개를 저장할 수 있는 공간 할당
int *arr = (int *)malloc(sizeof(int) * 3);

// ...

// 정수 6개를 저장할 수 있는 공간으로 크기 변경
arr = (int *)realloc(arr, sizeof(int) * 6);
```

[Before realloc]

arr[0]	arr[1]	arr[2]
10	20	30

[After realloc]

arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]
10	20	30	?	?	?

Allocation Function

♣ realloc()

- 안전한 realloc 방식: realloc 실패 시 NULL을 반환하여 기존의 주소를 잃어버릴 수 있으므로, 임시 포인터를 활용한다.

```
int *arr = (int *)malloc(sizeof(int) * 3);
int *temp = (int *)realloc(arr, sizeof(int) * 6);

if (temp == NULL) {
   free(arr);
   return 1;
}

arr = temp;
```

Allocation Function

♣ Example: realloc()

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

int main(void) {
   int *arr;
   arr = (int *)malloc(sizeof(int) * 3);

   if (arr == NULL) {
      printf("Memory allocation failed₩n");
      return 1;
   }

   arr[0] = 10; arr[1] = 20; arr[2] = 30;

   int *temp = (int *)realloc(arr, sizeof(int) * 6);
   if (temp == NULL) {
      printf("Memory reallocation failed₩n");
      free(arr);
      return 1;
   }

   arr = temp; arr[3] = 40; arr[4] = 50; arr[5] = 60;

   for (int i = 0; i < 6; i++) printf("%d ", arr[i]);

   free(arr);
   return 0;
}
```

Allocation Function

♣ calloc()

- 동적 할당을 하되, 할당된 메모리의 모든 비트를 0으로 초기화하는 함수
- void *calloc(size_t count, size_t size);
- calloc 실패 시 NULL을 반환한다.

```
// 할당하는 6개의 정수 공간 모두 0으로 채워진다.
int *arr = (int *)calloc(6, sizeof(int));
```

0	0	0	0	0	0
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]

Allocation Function

♣ Example: calloc()을 활용한 빈도수 배열

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

int main(void) {
   int data[8] = {1, 3, 3, 5, 1, 9, 3, 5};
   int *count;

   count = (int *)calloc(10, sizeof(int));

   if (count == NULL) {
      printf("Memory allocation failed₩n");
      return 1;
   }

   for (int i = 0; i < 8; i++) count[data[i]]++;
   for (int i = 0; i < 10; i++) printf("%d: %d₩n", i, count[i]);
   free(count);

   return 0;
}
```

Free

♣ free()

- heap 영역에 할당한 메모리 공간은 프로그래머가 직접 회수
- free하지 않으면 불필요한 공간 낭비
- void free(void *ptr);

```
int *arr = (int *)malloc(sizeof(int) * 5);

/* use arr */

free(arr);
```

- free(arr): arr이 가리키는 heap 메모리 영역을 해제함
  (arr이라는 포인터 변수 자체가 사라지는 것이 아님에 주의!)
- free(arr) 이후 arr이 자동으로 NULL이 되지 않는다.

Free

♣ Dangling Pointer

- 더 이상 사용할 수 없는 메모리 주소를 가리키고 있는 포인터

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

// 이제 p는 dangling pointer에 해당한다.

// 안전한 처리
p = NULL;
```

- 만약 p를 통해 또다시 heap 영역에 접근하면? ➔ Use After Free

Free

♣ Use After Free

- 해제한 메모리에 접근하는 문제

```
int *p = (int *)malloc(sizeof(int));
*p = 10;
free(p);
printf("%d₩n", *p); // UAF
```

- Undefined Behavior가 발생 ➔ 실행 결과 예측 불가능
  ex) 10이 출력, 이상한 주소 값 출력, 프로그램 비정상 종료 등

Free

♣ Double Free

- 해제한 메모리를 또다시 free()하는 문제

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

free(p);
// p는 이제 dangling pointer

free(p);  // double free
```

- free()는 heap memory를 메모리 관리자에게 반납하는 동작
- double free시 메모리 관리자가 heap 상태를 잘못 이해하고 Undefined Behavior가 발생 ➔ 실행 결과 예측 불가능

Free

♣ Double Free

- free(NULL)은 아무런 문제가 발생하지 않는다.

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

free(p);
// p는 이제 dangling pointer
p = NULL;
free(p);  // Not a problem
```

Free

♣ Memory Leak

- heap에 할당한 메모리를 사용 후 free()하지 않아서, 프로그램이 불필요하게 메모리를 계속 차지하는 문제

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

int main(void) {
    int *p = (int *)malloc(sizeof(int));
    if (p == NULL) {
        printf("Memory allocation failed₩n");
        return 1;
    }
    *p = 10;
    printf("%d₩n", *p);

    // free(p)를 하지 않음
    return 0;
}
```

Free

[Quiz] – 다음 프로그램에 존재하는 보안상 취약점은?

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

int main(void) {
   int *p = (int *)malloc(sizeof(int));

   if (p == NULL) {
      printf("Memory allocation failed₩n");
      return 1;
   }

   *p = 10;

   int *q = p;

   free(p);
   p = NULL;

   free(q);
   return 0;
}
```

Free

[Quiz] – Answer: Double Free

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

int main(void) {
   int *p = (int *)malloc(sizeof(int));

   if (p == NULL) {
      printf("Memory allocation failed₩n");
      return 1;
   }

   *p = 10;

   int *q = p;

   free(p);
   p = NULL;

   free(q); // Double Free
   return 0;
}
```

Exercise

[Practice]

학생 수 N을 입력 받고, N명의 이름과 수학 점수를 입력 받는다. 전체 학생 목록, 평균 점수, 최고 점수를 받은 학생의 이름과 점수를 예시와 같이 출력한다. 반드시 동적 할당을 활용한다. (단, 학생의 이름은 20자 이하이다.)

[입력 1]
3
Kim 90
Lee 85
Park 95

[출력 1]
Student List
Kim 90
Lee 85
Park 95

Average: 90.00
Top Student: Park 95

Exercise

[Practice] - Answer

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

typedef struct {
    char name[21];
    int score;
} Student;

int main(void) {
    int n;
    Student *students;
    int sum = 0;
    int top_index = 0;

    scanf("%d", &n);
    students = (Student *)malloc(sizeof(Student) * n);

    if (students == NULL) {
        printf("Memory allocation failed₩n");
        return 1;
    }

    for (int i = 0; i < n; i++) {
        scanf("%20s %d", students[i].name, &students[i].score);
        sum += students[i].score;

        if (students[i].score > students[top_index].score) {
            top_index = i;
        }
    }

    printf("Student List₩n");

    for (int i = 0; i < n; i++) {
        printf("%s %d₩n", students[i].name, students[i].score);
    }

    printf("₩n");
    printf("Average: %.2f₩n", (double)sum / n);
    printf("Top Student: %s %d₩n",
           students[top_index].name,
           students[top_index].score);
    free(students);
    students = NULL;

    return 0;
}
```

Exercise

[Quiz] – 다음 프로그램에 존재하는 보안상 취약점은?

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

int main(void) {
   int *p = (int *)malloc(sizeof(int));
   int *q = (int *)malloc(sizeof(int));

   if (p == NULL || q == NULL) {
      printf("Memory allocation failed₩n");
      free(p);
      free(q);
      return 1;
   }

   *p = 10;
   *q = 20;
   p = q;
   free(p);

   return 0;
}
```

Exercise

[Quiz] – Answer: Memory Leak

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

int main(void) {
   int *p = (int *)malloc(sizeof(int));
   int *q = (int *)malloc(sizeof(int));

   if (p == NULL || q == NULL) {
      printf("Memory allocation failed₩n");
      free(p);
      free(q);
      return 1;
   }

   *p = 10; *q = 20;
   p = q; // 원래 p가 가리키는 공간을 더 이상 가리키는 포인터가 없고, 이를 free()하지 않고 프로그램 종료
   free(p);

   return 0;
}
```

THANK YOU!

Any Question?