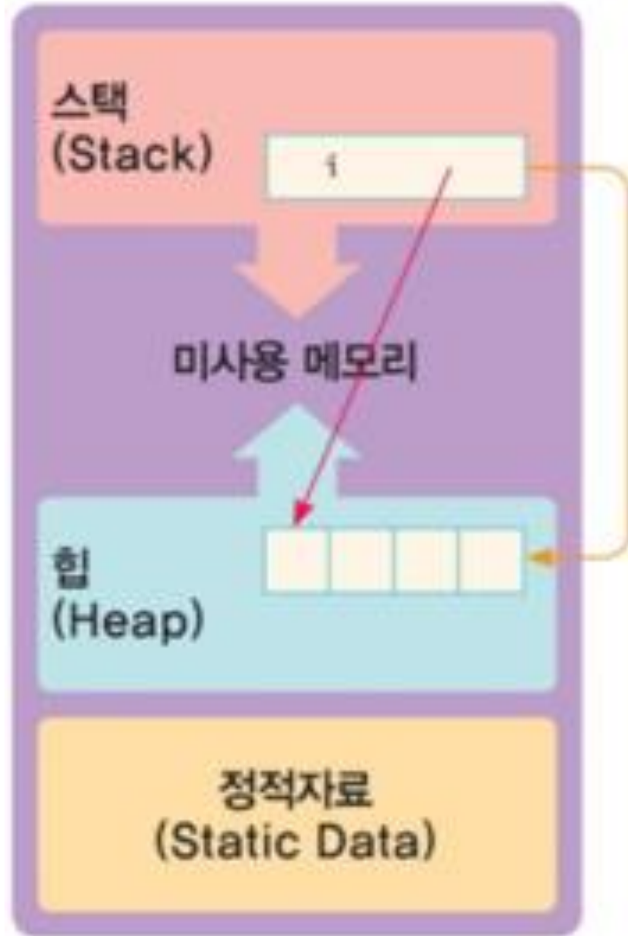
16. 동적 메모리와 전처리

14주차

메모리 영역과 메모리 할당방법



정적 메모리 할당:
- 변수 선언으로써 메모리 확보.
- 컴파일 시점(실행 전)에 크기 고정.

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

동적 메모리 할당:
- 변수 선언으로써 메모리 확보.
- 실행 중에 필요한 만큼 할당.
- 메모리 사용량 예측이 어려울 때 유용.
- heap 영역에서 할당됨.
- malloc(), free(), realloc() 함수 사용.

int형의 저장 공간을 동적으로 확보하기 (malloc.c)

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

int main(void) {
    int *pi = NULL;
    pi = (int *)malloc(sizeof(int));
    if (pi == NULL) {
        printf("메모리 할당에 문제가 있습니다.");
        exit(1);
    };
    *pi = 3;
    printf("주소 값: *pi = %d, 저장 값: p = %d₩n", pi, *pi);
    free(pi);

    return 0;
}
```

[결과]
주소 값: *pi = -1249227952, 저장 값: p = 3

메모리 할당 함수 malloc()으로 동적메모리 할당

동적메모리 할당 성공 검사

메모리 해제

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
int main(void) {
    int n = 0, sum = 0;
    int *ary = NULL;
    printf("입력할 점수의 개수를 입력 >> 
    scanf("%d", &n);
    if ((ary = (int *)malloc(sizeof(int)*n)) == NULL) {
        printf("메모리 할당에 문제가 있습니다."); exit(1); };
    printf("%d개의 점수 입력 >> ", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", (ary + i));
        sum += *(ary + i); //sum += ary[i];
    }
    printf("입력 점수: ");
    for (int i = 0; i < n; i++)
        printf("%d ", *(ary + i));
    printf("₩n");
    printf("합: %d  평균: %.1f₩n", sum, (double)sum / n);
    free(ary);
    return 0;
}
```

[결과]
입력할 점수의 개수를 입력 >> 3
3개의 점수 입력 >> 3
5
1
입력 점수: 3 5 1
합: 9 평균: 3.0

배열을 동적으로 할당 -> 성공 검사!

표준 입력으로 동적 할당된 배열에 내용 채우기

int형 배열 저장 공간을 동적으로 확보하기 (arraymalloc.c)

0으로 초기화된 동적 할당하기 (calloc.c)

```
#include <stdio.h>
#include <stdlib.h>
void myprintf(int *ary, int n);

int main(void) {
    int *ary = NULL;
    if ((ary = (int *)calloc(3, sizeof(int))) == NULL)       {
        printf("메모리 할당이 문제가 있습니다.₩n"); exit(EXIT_FAILURE); }
    myprintf(ary, 3);
    free(ary);
    myprintf(ary, 3);
    return 0;
}

void myprintf(int *ary, int n) {
    for (int i = 0; i < n; ++)
        printf("ary[%d] = %d₩n", i, *(ary + i));
}
```

calloc()은 모두 기본 값인 0 으로 초기화 해 줌.

메모리 해지 이후이므로 모두 쓰레기 값 출력

[결과]

```
ary[0] = 0
ary[1] = 0
ary[2] = 0
ary[0] = -572662307
ary[1] = -572662307
ary[2] = -572662307
```

이미 할당된 메모리를 변경 할당하기 (realloc.c)

```
#include <stdio.h>
#include <stdlib.h>
void myprintf(int *ary, int n) {
    for (int i = 0; i < n; i++)
        printf("ary[%d] = %d ", i, *(ary + i));
    printf("₩n");
}
int main(void) {
    int *reary, *cary;
    if ((cary = (int *)calloc(3, sizeof(int))) == NULL) {
        printf("메모리 할당이 문제가 있습니다.₩n"); exit(EXIT_FAILURE); }
    if ((reary = (int *)realloc(cary, 4 * sizeof(int))) == NULL) {
        printf("메모리 할당이 문제가 있습니다.₩n"); exit(EXIT_FAILURE); }
    myprintf(reary, 4);
    free(reary);
    return 0;
}
```

앞 3개는 기본 값인 0 출력,
마자막 하나는 다른 값

[결과]

```
ary[0] = 0 ary[1] = 0 ary[2] = 0 ary[3] = -842150451
```

자기참조 구조체(self reference struct)

- 자기참조 구조체(self reference struct)란
  - 구조체 멤버로 자기 자신인 구조체 포인터를 갖는 구조체
  - 자기 자신을 멤버로 가지는 것은 불가능

- 연결리스트
  - 동일한 데이터를 선형으로 동적인 개수만큼 유지하는 자료구조.
  - 자기 참조 구조체로써 동일 구조체의 표현 여러 개를 만들어 구현함.

```
struct selfref {
    int n;
    struct selfref *next;
    struct selfref data;
};
```

자기참조구조체를 이용한 연결리스트 기본 (selfrefstruct.c 1/2)

```
#include <stdio.h>
#include <stdlib.h>
struct selfref {
        int n;
        struct selfref *next;
        //struct selfref one;       //컴파일 오류 발생
};
```

자기참조구조체

```
[결과]
구조체 크기= 16

첫 번째 구조체:        자료의 주소값(first) = 1826312704
        자료값(first->n) = 100
        자료값(first->next) = 1826354416
        자료값(first->next->n) = 200

두 번째 구조체:        자료의 주소값(second) = 1826354416
        자료값(second->n) = 200
        자료값(second->next) = 0
```

```c
int main(void) {
    typedef struct selfref list;
    list *first = NULL, *second = NULL;
    first = (list *)malloc(sizeof(list));
    second = (list *)malloc(sizeof(list));
    first->n = 100;
    first->next = NULL;
    second->n = 200;
    second->next = NULL;
    first->next = second;
    printf("구조체 크기= %d₩n₩n", sizeof(list));
    printf("첫 번째 구조체: ");
    printf("₩t자료의 주소값(first) = %u₩n", first);
    printf("₩t자료값(first->n) = %d₩n", first->n);
    printf("₩t자료값(first->next) = %u₩n", first->next);
    printf("₩t자료값(first->next->n) = %d₩n₩n", first->next->n);
    printf("두 번째 구조체: ");
    printf("₩t자료의 주소값(second) = %u₩n", second);
    printf("₩t자료값(second->n) = %d₩n", second->n);
    printf("₩t자료값(second->next) = %u₩n", second->next);
    free(first);     free(second);
    return 0;
```

① 구조체 struct selfref를 하나의 자료형인 list 형으로 정의

② 리스트에 포함시킬 2개 항목을 할당받기

③ 2개의 구조체의 멤버를 초기화

first가 다음 구조체인 second를 가리키게 함.

자기참조구조체를 이용한

배열과 연결리스트

- 공통점 : 순차적 자료구조

- 배열
  - 장점 : 첨자(index)를 사용하여 원소를 직접 임의참조(random access) 가능
  - 단점 : 원소의 삽입, 삭제 시 많은 원소의 이동 필요
  - 컴파일 전에 배열의 크기 결정, 실행 중간에 배열크기 수정 불가능

- 연결리스트
  - 노드(배열의 원소에 해당. 자기참조구조체로 구현)가 순차적으로 연결된 자료구조.
  - 동적인 노드 생성, 삭제 가능 : 기억 장소를 미리 확보해 둘 필요 없음.
  - 링크(link) 이용하여 다음 노드를 가리킴.
  - 특정 노드를 임의 참조 할 수 없음. (순차적 참조만 가능)
  - 노드 삽입, 삭제 시 다른 노드에 대한 영향 최소화.

연결리스트의 기본 연산

- 노드 순회(node traversal)
  - 연결 리스트에서 모든 노드를 순서대로 참조하는 방법
  - 헤드부터 계속 노드 링크의 포인터로 이동
    - 링크가 NULL이면 마지막 노드
- 노드 추가 (마지막 노드로 추가)

  ❶ 추가할 노드를 메모리 할당 -> 자료 저장 -> 링크를 NULL로 만듬.
  ❷ 기존 연결 리스트를 순회하여 마지막 노드로 이동
  ❸ 마지막 노드의 링크 새로운 노드의 주소값 저장
- 노드 삽입 (노드A의 뒤에 새로운 노드를 삽입)

  ❶ 추가할 노드를 메모리 할당 -> 자료 저장
  ❷ 새로운 노드의 링크에 노드 A의 링크를 저장
  ❸ 노드 A의 링크에 노드 새로운 노드의 주소값 저장
- 노드 삭제

  ❶ 기존 연결 리스트를 순회하여 삭제할 노드의 바로 앞 노드 (노드 A) 찾기
  ❷ 노드 A의 링크에 삭제할 노드의 링크값 저장
  ❸ 삭제할 노드를 메모리 해제 (free())

연결리스트의 구현
(실행 결과)

- 2개의 소스 파일과 1개의 헤더 파일 사용
- 소스 파일
  - object file로 전환되어 실행파일(*.exe)이 됨
- 사용자 정의 헤더 파일
  - 공통으로 사용될 함수원형, 매크로, 자료형 재정의 등을 포함

- Visual studio project "LinkedList" 로 구현.

```
[결과]
이름을 입력하고 Enter를 누르세요. >>
jychoi
  1번째 노드는 jychoi
gdhing
  1번째 노드는 jychoi
  2번째 노드는 gdhing
jkkim
  1번째 노드는 jychoi
  2번째 노드는 gdhing
  3번째 노드는 jkkim
```

연결리스트 구현 (linkedlist.h 1/5)

```
//scanf(), gets() 등 오류를 방지하기 위한 상수 정의
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct linked_list {
        char *name;
        struct linked_list *next;
};

typedef   struct linked_list NODE;

typedef   NODE * LINK;

LINK createNode(char *name);
LINK append(LINK head, LINK cur);
int printList(LINK head);
void freeList(LINK head);
```

자기참조구조체 정의

구조체를 NODE 타입으로 재정의

NODE *를 LINK로 재정의

연결리스트 구현 (listlib.c 노드를 생성하는 함수 2/5)

```
#include "linkedlist.h"
LINK createNode(char *name) {
    LINK    cur;
    cur = (LINK) malloc(sizeof(NODE));
    if (cur == NULL) {
        printf("노드 생성을 위한 메모리 할당에 문제가 있습니다.₩n");
        return NULL;
    }

    cur->name = (char *)malloc(sizeof(char) * (strlen(name) + 1));
    strcpy(cur->name, name);

    cur->next = NULL;

    return cur;
}
```

malloc()으로 할당된 메모리 주소를 포인터 cur에 저장

name 인자의 내용을 저장할 문자배열을 동적 할당하여 name 항목에 저장.

연결 작업은 차후에 할 것이므로 next 값은 NULL로 하기.

연결리스트 구현
(listlib.c 노드를 추가하는 함수 3/5)

```
LINK append(LINK head, LINK cur) {
    LINK nextNode = head;

    if (head == NULL) {
        head = cur;
        return head;
    }

    while (nextNode->next != NULL) {
        nextNode = nextNode->next;
    }

    nextNode->next = cur;

    return head;
}
```

- `LINK nextNode = head;` ← 연결리스트 순회를 위해 지역 변수 nextNode를 선언하고 head로 초기화.
- `if (head == NULL) { ... }` ← 리스가 비었을 때는 추가하려는 노드가 head가 됨
- `while (nextNode->next != NULL) { ... }` ← nextNode가 마지막 노드를 가리키게 될 때까지 리스트를 순회
- `nextNode->next = cur;` ← 추가 노드를 현재 노드의 next에 저장

연결리스트 구현
(listlib.c 연결 리스트의 모든 노드 출력 함수 4/5)

```
int printList(LINK head) {
    int cnt = 0; //방문한 노드의 수를 저장
    LINK nextNode = head;

    while (nextNode != NULL) {
        printf("%3d번째 노드는 %s₩n", ++cnt, nextNode->name);
        nextNode = nextNode->next;
    }

    //총 노드 방문 횟수를 반환
    return cnt;
}
```

nextNode를 이용하여 연결리스트의 처음부터 끝까지 순회

```
void freeList(LINK head) {
    LINK savenext = NULL;
    while (head != NULL) {
        savenext = head->next;
        free(head->name);
        free(head);
        head = savenext;
    }
}
```

연결리스트 구현
(linkedlist.c main함수 5/5)

```
#include "linkedlist.h"
int main(void) {
    char name[30];
    LINK head = NULL;
    LINK cur;
    printf("이름을 입력하고 Enter를 누르세요. >> ₩n");
    gets(name);
    while (strlen(name) > 0) {
        cur = createNode(name); //노드 동적 할당
        if (cur == NULL) {
            printf("동적메모리 할당에 문제가 있습니다.₩n");   exit(1);
        }
        head = append(head, cur);
        printList(head);
        gets(name);
    }
    freeList(head);
    return 0;
}
```

표준 입력으로 받기

동적 할당한 노드를 맨 뒤에 추가.