정적 메모리: 프로그램이 시작되기 전 미리 정해진 크기의 메모리가 할당되고, 프로그램 종료 전 까지 사라지지 않는다. (변수 선언)

1. 처음 입력 크기보다 큰 입력 처리x / 공간 남으면 메모리 낭비

**동적 메모리: 요청 할 때 마다 그때그때 사용**

1. 실행 도중 동적으로 메모리 할당 / 사용 끝나면 반납
2. 메모리의 효율적 사용
3. **int \* score = (int\*)malloc(100\*sizeof(int));**

* **malloc을 사용해 동적 메모리 할당(<stdlib.h>사용 필수!!)**
* 원래 반환형은 void\*이므로 형변환
* 100\*sizeof(int)라고 설정하는 이유: 나중에 코드를 봤을 때 “100개의 정수를 위해서 할당 했군” 이라고 알 수 있게 해준다.
* score는 포인터 변수이므로 데이터 세그먼트에 저장되고, 주소를 찾아가(동적메모리 있는 곳)byte를 쳐다보고 그것을 어던 자료형인지, 크기는 얼마인지 해석
* 메모리 할당 후, **if(score == NULL){**

**printf(“동적메모리 할당 오류”);**

**exit(1); }**

와 같은 처리 해줘야 메모리 할당이 제대로 되었는지 확인 가능

(할당 실패 할 수도 있기 떄문에…..예외 처리와 비슷함)

동적 메모리 사용법

1. **포인터**를 통해 사용

\*score = 100;

\*(score+1) = 200;

1. 동적 메모리를 **배열과 같이 취급**

score[0] = 100;

**동적 메모리 반납:**

* **free(score);**

1. 메모리 반납 후엔 변수 score 사용 x

(메모리를 반납하면 ‘dangly reference’가 되므로 이 변수를 사용하는 것은 심각한 문제를 발생시킴)

**calloc():** **0으로 초기화** 된 메모리를 할당

1. 바이트 단위가 아닌 항목의 단위로 메모리를 할당

Int \*p = (int\*)**calloc(5(항목의 개수),** sizeof(int)**(항목 하나의 크기));**

**realloc():** 할당했던 메모리 블록의 크기를 변경

int \*p = (int\*)malloc(5\*sizeof(int));

**p = realloc(p(이전 주소, 크기 변경할 주소), 7\*sizeof(int)(바꿀 크기));**

ex) int \*list = (int\*)malloc(2\*sizeof(int)):

list[0] = 10;

list[1] = 20;

**int \*new\_list = (int\*)realloc(list, 3\*sizeof(int)):**

**(new\_list에 list[0], [1]의 내용이 유지 된다.)**

new\_list[2] = 30; **//new\_list[0]은 여전히 10 이고, [1]은 20이다.**

**연결 리스트(linked list): 각 항목이 포인터를 사용해 다음 항목을 가리킴**

1. **중간에 데이터를 삽입, 삭제하기 쉽다(배열의 단점 보완)**
2. **랜덤 접근은 어렵다. (처음부터 차근차근 찾아가야 함)**
3. **데이터를 저장할 공간이 필요할 때마다 동적으로 공간을 만들어서 쉽게 추가할 수 있다. (배열의 단점 보완)**
4. **구현 어렵고, 오류 나기 쉬움**
5. **노드(node) = 데이터 필드(data field) + 링크 필드(link field)**

**(데이터 필드는 데이터를 저장하고, 링크 필드는 다음 노드를 가리키는 포인터로 이루어져 있는 구조체이다.)**

* **첫 번째 노드를** 가리키는 것을 **헤드 포인터라고** 한다.
* 노드는 동적을 생성된다.

**자기 참조 구조체: 구조체의 멤버 중에 구조체와 같은 타입의 구조체를 가리키는 포인터가 존재하는 것**

**typedef struct NODE {**

**int data;**

**struct NODE \*link; //다음 struct NODE 타입의 구조체를 가리키는 포인터**

**}NODE;**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct book {

char title[30];

int year;

struct book\* link;

}BOOK;

int main(void) {

BOOK\* p; //노드를 가리킬 포인터//노드가 생성 될 때마다 가리키는 수시로 바뀌는 놈

BOOK\* head = NULL; //헤드 포인터

BOOK\* prev; //

char buffer[30];

BOOK\* next;

while(1) {

printf("책의 제목을 입력하시오: (종료하려면 엔터)");

gets\_s(buffer, sizeof(buffer));

if (buffer[0] =='\0') {//엔터 누르면 NULL이 들어가고 종료

break;

}

p = (BOOK\*)malloc(sizeof(BOOK));

strcpy(p->title, buffer);

printf("책의 출판연도를 입력하시오:");

gets\_s(buffer,sizeof(buffer));

p->year = atoi(buffer);

if (head == NULL) { //헤드가 비어 있다면

head= p; //헤드가 첫 번째 노드를 가리킴

}

else {

prev->link = p; //노드와 노드를 연결

}

p->link = NULL;//다음 노드 생성 시,연결하려면 현재 노드의 link가 NULL이여야 함

prev= p;

printf("\n");

}

printf("\n");

//노드의 책 정보 출력

p = head;//처음부터 출력해야 하므로 head부터 다시 가리킴

while (p != NULL) {

printf("책이 제목:%s 출판연도:%d\n", p->title, p->year);

p = p->link;//다음 노드 가리킴

}

//동적 메모리 반납

p = head;//처음부터 반납해야 하므로 head부터 다시 가리킴

while (p != NULL) {

next = p->link;

free(p);

p = next;

}

return 0;

}