c 언어 보고서

2021210088 허지혜

9장. 포인터

09-1. 포인터의 기본 개념

- 포인터는 다른말로 주소라고 한다. 즉, 포인터는 메모리를 사용하는 또 다른 방법이다.
- 주소 연산자 &로 변수가 할당된 메모리의 위치를 확인한다. “&변수명” 이렇게 쓴다.
- 포인터를 가리키는 변수를 사용할 때 간접 참조 연산자 *를 쓴다.
- const를 변수 앞에 붙이면 값을 변경하지 못하도록 하며 해당 변수를 상수 취급한다.
이는 변수의 초기값이 변하지 않게 할 때 쓴다.

구분	사용 예	기능
주소 연산자	int a; &a;	변수 앞에 붙어 사용하며, 변수가 할당된 메모리의 시작 주소 값을 알린다.
포인터	char *pc; int *pi; double *pd;	시작 주소 값을 지정하는 변수를 가리키며 가리키는 자료형을 표시하여 선언한다.
간접 참조 연산자	*pi = 18;	포인터에 사용되며 포인터가 가리키는 변수를 사용한다.

09-2. 포인터 완전 정복을 위한 포인터 이해하기

- 주소와 포인터는 상수와 변수의 차이가 있다.
주소 : 변수에 할당된 메모리 공간의 시작 주소 값 자체
포인터 : 그 값을 저장하는 또 다른 메모리 공간
특정 변수의 주소 값은 바뀌지 않지만 포인터는 다른 주소를 대입하여 바꿀 수 있다.
- 포인터의 크기는 주소의 크기와 같다.
- 포인터에 주소를 저장할 때는 가리키는 자료형이 같아야 한다.
- 형 변환을 사용한다면 포인터의 대입은 언제나 가능하다.
- 포인터의 주요 기능 중 하나는 함수 간에 효과적으로 데이터를 공유하는 것이다.

구분	변수 a 사용	포인터 pa 사용
대입 연산자 왼쪽	a=10;	*pa=10;
대입 연산자 오른쪽	b=a;	b=*pa;
피연산자	a+20;	*pa+20;
출력	printf(“%d”,a);	printf(“%d”,*pa);
입력	scanf(“%d”,&a);	scanf(“%d”,&*pa); scanf(“%d”,pa);

구분	사용 예	기능

포인터	int a,b;
	int *p=&a;
	p=&b;
		포인터는 변수이므로 그 값을 다른 주소로 바꿀 수 있다.

포인터 크기	int *p; sizeof(p)	포인터의 크기는 컴파일러에 따라 다를 수 있으며 sizeof 연산자로 확인 가능하다.
포인터 대입 규칙	int *p; double *pd; pd=p; (이거 앙대)	포인터는 가리키는 자료형이 일치할 때만 대입한다.

10장. 배열과 포인터

10-1. 배열과 포인터의 관계

- 배열을 자료형이 갖는 변수를 메모리에 연속으로 할당한다.
- 각 배열 요소는 일정 간격으로 주소를 가진다.
- 첫 번째 요소의 주소를 알면 나머지 요소의 주소도 쉽게 알 수 있으며, 간접 참조 연산을 수행하면 모든 배열 요소를 사용할 수 있다.
- 주소는 정수처럼 보이지만 자료형에 대해 정보를 가지고 있는 특별한 값으로 연산을 자유롭게 할 수 없고 정해진 연산만 가능하다.
- int형 변수 a의 주소 100에 1을 더하면 int형은 4byte가 더해져 104가 된다.
- 주소 + 정수 = 주소 + (정수 * 주소를 구한 변수의 크기(sizeof))
- 배열명은 첫 번째 요소의 주소이고 포인터에 저장 가능하다. 이 경우 포인터로도 연산식이나 대괄호를 써서 배열 요소를 쉽게 사용할 수 있다.
- 배열과 포인터의 차이점

1) sizeof 연산의 결과가 다르다.

배열명을 사용하면 배열 전체의 크기를 구한다. 포인터에 사용하면 포인터 하나의 크기를 구한다.

2) 변수와 상수의 차이가 있다.

포인터는 그 값을 바꿀 수 있지만 배열명은 상수이므로 값을 바꿀 수 없다.

- 포인터의 뺄셈과 관계 연산

포인터에는 자료형이 같으면 포인터끼리 뺄셈도 가능하다.

포인터 - 포인터 = 값의 차 / 가리키는 자료형의 크기

구분	사용 예	기능
배열명	int ary[3]; ary == &ary[0];	배열명은 첫 번째 요소의 주소
배열명 + 정수	int ary[3]; ary + 1;	가리키는 자료형의 크기를 곱해서 더한다. 주소 + (정수 * 주소를 구한 변수의 크기(sizeof))

배열명과 포인터의 공통점	int ary[3];
	int *pa = ary;
	pa[1] = 10;
		포인터가 배열명을 저장하면 배열명처럼 쓸 수 있다. 위 예시는 두 번째 배열 요소에 10을 대입한다는 의미이다.

배열명과 포인터의 차이점	ary++ (앙대) pa++(됑)	배열명은 상수이므로 그 값을 바꿀 수 없지만 포인터는 가능하다.

10-2. 배열을 처리하는 함수

- 배열명을 꼭 포인터에 넣는 방식으로 배열을 처리할 필요가 없지만, 함수로 처리할 경우 포인터가 꼭 필요하다.
- ary 배열에서 배열명 ary는 첫 번째 배열 요소의 주소인데, 이 주소 값을 함수의 인수로 주면 함수는 이 값을 받아 주소 계산을 통해 모든 배열 요소를 사용할 수 있다. 이 때 배열명을 받을 함수의 매개변수 자리에 포인터가 필요하다.

```c
#include <stdio.h>

void print_ary(int* pa);

int main(void)
{
    int ary[5] = { 10, 20, 30, 40, 50 };

    print_ary(ary);

    return 0;
}

void print_ary(int* pa)
{
    int i;

    for (i = 0; i < 5; i++)
    {
        printf("%d ", pa[i]);
    }
}
```

```
Microsoft Visual Studio 디버그 콘솔
10 20 30 40 50
C:₩Users₩chy20₩source₩repos₩Project1₩Debug₩Project1.exe(프로
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션]
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

예제에서 print_ary 함수를 정의해줬는데 이때 main 안에 ary를 쓸 수 없으니 받아올 수 있는 인자인 pa를 넣어 출력해야 한다.

- 배열의 값을 출력하는 함수는 첫 번째 배열 요소의 주소만 알면 되므로 배열 요소 개수가 달라도 구현 방식은 동일하다.
- 배열 요소의 개수가 바뀌면 출력문의 반복 횟수가 달라지므로 함수 수정은 불가피하다.

- 배열에 값을 입력하는 함수도 배열의 값을 출력하는 함수와 구현 방법이 동일하다.
- 입력 함수는 데이터를 저장할 배열의 위치가 필요하므로 함수 안에서 포인터를 직접 사용한다.

	배열을 출력하는 함수	배열을 입력하는 함수
호출	int ary[5]={10,20,30,40,50}; print_ary(ary,5);	int ary[5]; input_ary(ary,5);

정의	void print_ary(int *pa, int size)
	{
	int i;
	for (i=0,i<size,i++)
	{
	printf("%d", pa[i]);
	}
	}
		void input_ary(int *pa, int size)
		{
		int i;
		for (i=0;i<size;i++)
		{
		scanf("%d", pa+i);
		}
		}

11장. 문자

11-1. 아스키 코드 값과 문자 입출력 함수

- a : 변수명, 'a' : 문자 상수 로 컴파일러는 해석한다. 문자 상수는 아스키 코드 값으로 바뀌여서 나온다. 예로 'a'는 정수 값 97로 바뀐다. 즉, 문자는 메모리에 저장되는 방식이 같고 int형 변수에 저장하고 정수처럼 연산이 가능하다.

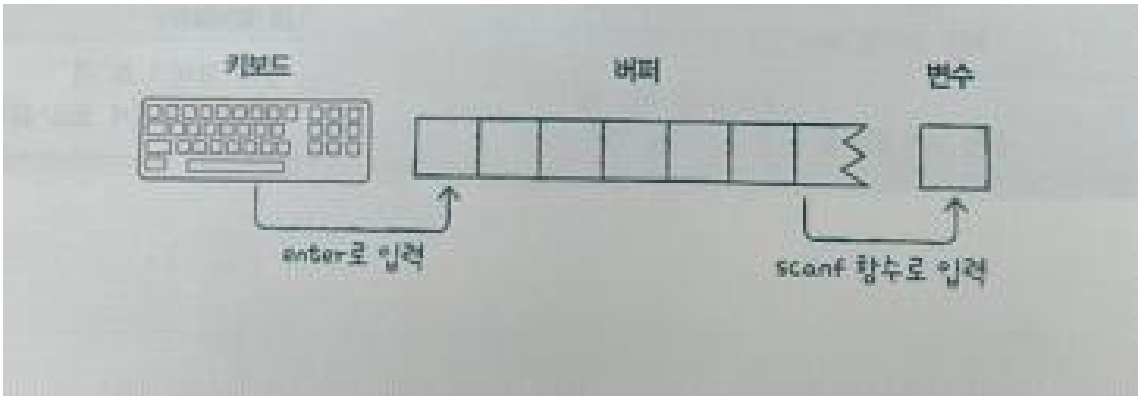
종류	문자 상수	아스키 코드 값	출력
숫자 문자(10개)	'0' ~ '9'	48 ~ 57	문자 출력
대문자(26개)	'A' ~ 'Z'	65 ~ 90	문자 출력
소문자(26개)	'a' ~ 'z'	97 ~ 122	문자 출력
특수 문자(33개)	' ', '$', '&', ...	32,36,38,...	문자 출력
제어 문자(33개)	'\0','\t','\n','\r', ...	0,9,10,13,...	제어 기능 수행

- scanf 함수로 문자를 입력할 때는 %c 변환 문자를 사용한다.
- %c는 알파벳이나 숫자 모양의 문자 등 형태가 있는 문자를 입력하면 공백(space), 탭(tab), 개행(enter) 문자와 같은 제어 문자도 입력하므로 주의해야 한다.
- 이들 세 문자는 숫자를 입력할 때값을 구분하기 위해 사용하지만 문자를 입력할 때 그 자체가 하나의 입력 데이터가 된다.
- scanf 함수는 문자,숫자 모두를 입력하는 기능이 있어 문자만 입력하는 함수에 비해 크기가 크다. printf도 마찬가지다. 따라서 문자만 사용할 시 문자 전용 입출력 함수 getchar(), putchar()를 사용하자.
- 모든 문자 상수는 아스키 코드 값으로 바뀌어 숫자로 저장되고 연산된다.
- %c 변환 문자는 화이트 스페이스도 입력하며 %c 앞에 공백을 사용하면 화이트 스페이스를 입력에서 제외할 수 있다.

구분	사용 예	기능
입력	char ch; scanf("%c", &ch);	char형 변수 사용 %c 변환 문자로 입력 공백 문자, 탭 문자, 개행 문자도 입력
	int ch; ch = getchar();	int형 변수 사용 입력 문자의 아스키 코드 값 반환 공백 문자, 탭 문자, 개행 문자도 입력
출력	printf("%c", ch); putchar(ch);	%c 변환 문자 사용 문자 출력 전용 함수, 출력할 문자 전달

11-2. 버퍼를 사용하는 입력 함수

- scanf, getchar 함수와 같은 표준 입력 함수들은 데이터를 입력시 버퍼를 거친다. 버퍼란 프로그램이 실행하는 중 운영체제가 자동으로 할당하는 메모리의 저장 공간이다.
- 데이터는 일단 버퍼에 저장된 후, scanf 함수에 의해 변수에 입력된다.



- 프로그램 사용자가 키보드로 한 줄을 입력할 때 입력을 끝내려면 엔터키를 누르면 된다.
- 개행 문자 또한 하나의 입력 데이터로 쓴다면 입력을 종료하는 별도의 신호가 필요한데 scanf 함수 반환값을 사용한다.
- scanf 함수는 ctrl + z를 누르면 -1을 반환한다. -1을 반환하기 전까지 반복 입력하면 개행 문자를 포함한 모든 문자를 데이터를 사용할 수 있다.
- getchar 함수도 버퍼를 사용하는 문자 입력 함수이다. getchar 함수를 반복 사용하먄 한 줄의 문자열을 char 배열에 입력할 수 있다.
- scanf 함수와 getchar 함수는 같은 버퍼를 사용하며 입력 데이터를 공유한다. 따라서 앞서 실행한 입력 함수가 버퍼에 남겨둔 데이터를 후에 수행하여 잘못 가져갈 수도 있다.
- 따라서 버퍼에 남아있는 불필요한 데이터는 미리 제거하는 것이 좋다.(scanf 함수로 데이터를 입력할 때 버퍼의 상태를 확인하자.)
- 버퍼에 저장되는 데이터의 끝에는 항상 개행 문자가 있다.

구분	함수 사용법	기능
입력	int ch; ch = fgetc(stdin);	int형 변수에 입력 공백 문자, 탭 문자. 개행 문자도 입력 입력 문자의 아스키 코드 값 반환 입력 버퍼 stdin 사용
출력	fputc(ch, stdout);	문자 출력 전용 함수 출력할 문자와 출력 버퍼 stdout 사용

12. 문자열

12-1. 문자열과 포인터

- 문자열은 크기가 일정하지 않기 때문에 컴파일러는 문자열 상수를 독특한 방법으로 처리한다. 문자열을 char형 배열 형태로 따로 보관하고 문자열 상수가 있던 곳에는 배열의 위치 값을 사용한다.
- 문자열을 char 포인터에 대입하면 문자열에 이름을 붙여 사용할 수도 있고 다른 문자열로 쉽게 바꿀 수도 있다.
- 배열에 문자열을 입력하는 방법은 다양한데 scanf로 해보자.

scanf 함수는 %s를 이용하여 공백이 없는 연속된 문자들을 입력받는다.

- 공백이 있는 문자를 넣고 싶다면 gets 함수를 사용한다. gets 함수는 버퍼에서 개행 문자를 가져오지만 배열에는 널 문자로 바꾸어 저장한다. gets 함수의 입력을 끝내는 방법은 enter를 누르는 것이다.
- fgets를 사용하여도 문자열을 입력할 수 있다. 차이점은 다음과 같다.

scanf, gets : 문자열의 크기가 배열 크기를 넘어설 위험성이 있다.

fgets : 배열 크기를 확인하여 배열 크기를 넘어서지 않게 할 수 있다.

형태 : fgets(str, sizeof(str), stdin(표준 입력));

gets와 fgets는 입력하는 방식은 같으나 개행 문자의 처리 방식이 다르다.

- scanf나 getchar은 입력 버퍼를 공유한다. 따라서 gets나 fgets 함수에서 개행 문자를 입력의 종료 조건으로 사용하게 되면 문제가 생긴다.
- 문자열을 출력할 때는 puts, fputs를 사용한다.

구분	char 포인터	char 배열
초기화	char *pc = “”mango“;	char str[80] = “mango”;
대입	pc = “banana”;	strcpy(str,“banana”);
크기	sizeod(pc) - 4byte	sizeof(str) - 80byte
수정	pc[0] = ‘t’ (앙대)	str[0] = ‘t’ (가능)
입력	scanf(“%s”, pc); (앙대)	scanf(“%s”, str); (가능)

12-2. 문자열 연산 함수

- CHAR 배열은 문자열을 저장하는 변수의 역할을 하며 쉽게 문자열로 초기화할 수 있다. 그렇지만 다른 문자열로 바꾸는 것은 문자를 하나씩 옮기는 번거로운 일을 수행해야 한다. 그런 번거로운 일을 하지 않기 위해 한 번에 대입하는 STRCPY 함수를 사용한다.
- string.h 헤더 파일을 include 해야 한다.

- strcpy 함수는 char 배열에 문자열을 복사하는 대입 연산 기능을 수행한다.
1) 첫 번째 인수는 char 배열이나 배열명을 저장한 포인터만 사용할 수 있다.
2) 두 번째 인수는 문자열의 시작 위치를 알 수 있다면 어떤 것이든 사용할 수 있다.
- strcpy 함수는 문자열을 복사할 때 문자의 수를 지정할 수 있다. 뒤에 정수를 붙이면 된다.
- strcpy 함수는 초기화된 문자열을 지우고 새로운 문자열을 바꿀 때 사용한다. 반면, 배열에 있는 문자열 뒤에 이어 붙일 때는 strcat 또는 strncat을 쓴다.
- strcat은 문자열을 이어 붙이며, strncat은 지정한 문자의 개수만큼 붙인다.
- strcat 사용시 주의 사항이 있다.
1) strcat 함수는 메모리를 침범할 수 있다.
2) strcat 함수를 사용할 때는 배열을 초기화해야 한다.
- 문자열의 길이를 계산할 때는 strlen 함수를 이용한다.
- 문자열을 비교할 때는 strcmp, strncmp 함수를 사용한다.
- strcmp는 두 문자열의 사전 순서를 판단하여 결과를 반환한다.
strcmp(str1, str2); : str1이 str2보다 사전에 나중에 나오면 1 반환
str1이 str2보다 사전에 먼저 나오면 -1 반환
str1과 str2가 같은 문자열이면 0 반환

연산 가능	사용 방법	실행 결과
대입	strcpy(str1, str2);	문자열 str2를 str1에 복사
길이 계산	strlen(str);	문자열 str의 길이를 구해 반환
붙이기	strcat(str1,str2);	str2를 str1 문자열 뒤에 이어 붙임
비교	strcmp(str1, str2);	str1 > str2 : 1 str1 < str2 : -1 str1 = str2 : 0