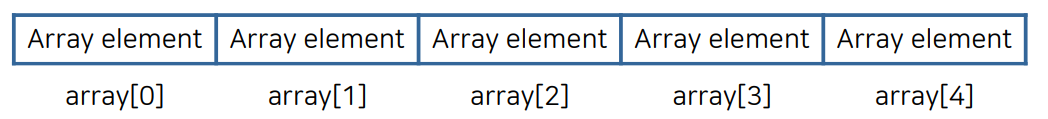
시스템 프로그래밍 기초

2022028886 조성흠

1. 배열

-배열

배열이 필요한 이유: 변수 선언과 출력을 한 번에 가능하다. 값 할당도 특수한 경우 한 번에 해결할 수 있다. 변수명을 관리하기에 유리함.

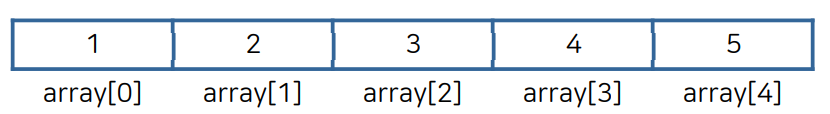
배열 사용하는 방법:  
배열 선언: int array[5];

배열의 index의 범위: 0 ~ 배열의 크기-1

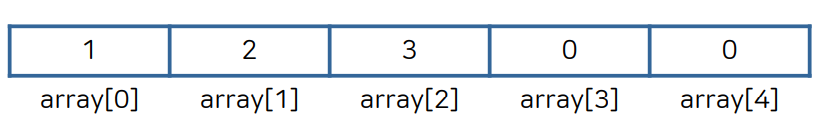
배열의 초기화:

만일 값을 중괄호 안에 채워서 대입하면, 0번 자리부터 하나씩 자동으로 채워짐.

Ex) int array[5] = {1, 2, 3, 4, 5};



만약 중괄호 안의 숫자 개수가 배열의 크기보다 작다면, 나머지 자리는 모두 0으로 채워짐

Ex) int array[5] = {1,2,3}

각 배열 초기 값의 자료형은 배열의 자료형과 일치해야 함.

Ex) double array\_d[] = {1.0, 2.1, 3.2, 4.3, 5.4};

Ex) char array\_c[] = {'a' , 'p', 'p' , 'l' , 'e'};

-배열의 크기 계산(sizeof)

배열의 크기 계산(sizeof): sizeof 연산자는 데이터 타입이 차지하는 메모리 크기를 바이트 단위로 반환함.

Ex) sizeof(char) = 1

Ex) int num; sizeof(num) = 4

배열의 이름은 배열 전체가 차지하는 메모리 크기를 바이트 단위로 반환함.

Ex) int ice[30]; sizeof(ice) = 120 // 4 \* 30

Ex) sizeof(ice)/sizeof(ice[0]) = 30 // 120 / 4

-배열 이름을 다루는 방법

배열의 각 요소는 스택 프레임에 연속적으로 저장됨 (Contiguous Memory)

Ex) 수식으로 나타내면 &rabbit[i] = &rabbit[0] + sizeof(int) \* i

배열의 이름은 배열의 시작 주소를 의미함

Ex) 수식으로 나타내면 &rabbit[i] = rabbit + sizeof(int) \* i

배열 요소를 sizeof 연산자의 인자로 사용하면 배열 요소 하나의 크기를 알 수 있음

Ex) 수식으로 나타내면 &rabbit[i] = rabbit + sizeof(rabbit[0]) \* i

-포인터

변수 이름: 변수의 이름은 데이터가 저장된 메모리 주소를 부르기 위한 별명

변수 주소: 변수의 주소는 데이터가 저장된 메모리 공간의 시작 주소를 가리킴

주소 연산자: 주소 연산자는 &이다.

간접 참조 연산자: 포인터는 \*(asterisk 라고 읽음) 연산자를 이용해 선언

int \*p; 라고 선언한 경우: p는 ‘ p가 가리키는 것의 주소’를 의미, \*p는 ‘ p가 가리키는 것’을 의미, int는 ‘ p가 가리키는 것의 데이터 타입이 정수형’이라는 의미

- 포인터 변수에 데이터 타입이 필요한 이유

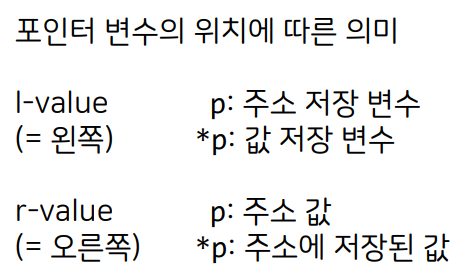
포인터 변수는 주소를 저장하는 것이므로, 타입에 상관 없이 동일한 크기로 저장됨, 포인터 변수의 크기는 address bus의 크기와 동일함.

포인터가 실제로 저장하는 것은 데이터를 저장한 메모리 공간의 시작 주소, 포인터가 가리키는 데이터 크기에 따라서 읽어야하는 메모리 크기가 달라짐

Cf) double은 8 바이트, int는 4 바이트, char는 1 바이트

포인터를 이용해서 데이터에 접근할 때 크기를 모르면, 데이터의 일부만 가져오는 문제가 생길 수 있음

Ex) double 타입 데이터인데, 4바이트만 가져올 수 있음 (원래보다 덜 읽음), int 타입 데이터인데, 8바이트만큼 가져올 수 있음 (원래보다 더 읽음)

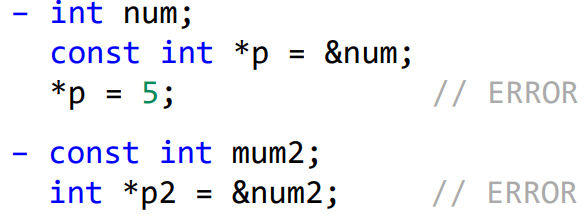
-포인터 유의 사항

포인터 변수는 L-value인지 R-value인지에 따라 의미가 달라짐, L-value일 경우 변수 자체를 의미하고, R-value일 경우 변수 값을 의미함

\* 연산자와 & 연산자는 역함수 관계로서 상쇄할 수 있음

-포인터와 const키워드

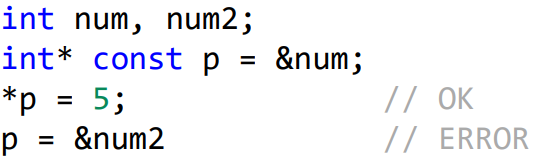
const int \*p는 (p가 아니라) \*p가 const라는 의미

p가 가리키는 대상이 상수라는 의미이므로 아래 코드는 오류

Ex)

int\* const p는 p 자체가 const라는 의미

p가 가리키는 대상은 바꿀 수 없지만, p가 가리키는 대상의 값은 바꿀 수 있음

Ex)

- 참조 호출 (call-by-reference)

call-by-value: C언어의 함수 호출은 기본적으로 값 호출

변수를 인자로 넣지만, 그 변수의 값을 복사하여 함수에 전달함

함수 안에서 값이 변하더라도, 인자로 넣은 변수에는 영향을 주지 않음

call-by-reference: 함수의 인자로 변수의 주소를 전달하는 것

주소를 주기 때문에, 그 주소에 접근하여 직접 값을 바꿀 수 있음

인자로 넣은 변수의 값을 함수 안에서 변경할 수 있음

- 포인터를 이용한 배열 전달

배열의 이름은 배열의 시작 주소를 가리키는 포인터

배열 이름을 함수의 인자로 전달하면 배열의 시작 주소만 전달됨 (속도/공간 측면에서 경제적)

2. 문자열

- C언어에서의 문자열 처리

문자열이란 문자를 나열한 것을 의미함

C++, Java 등의 언어에는 문자열을 다루는 별도의 타입이 있지만, C에는 없음

대신 문자들을 배열에 저장하고 첫 문자를 가리키는 포인터로 문자열을 대신함

문자 타입 배열의 첫 번째 주소를 만나면 포인터 산술 연산에 의해 줄줄이 접근할 수 있음

문자열의 끝을 알 수 있도록 제일 끝에 항상 널 문자(NULL Character) '\0'을 자동으로 붙임

C언어에서는 문자열 자체가 포인터, 즉 저장공간의 시작 주소를 의미하므로 아래와 같이 선언할 수 있음

Ex) char \*str = "hanyang"; char str[10] = "hanyang";

큰 따옴표로 둘러싸인 것은 문자열로 간주하며, 컴파일러가 자동으로 끝에 '\0'을 붙임

'\0'은 화면 출력 시 아무 것도 보이지 않으며, 아스키코드 0에 해당함

- C언어 문자열 선언 시 주의사항

printf는 문자열을 만났을 때, '\0'을 만나기 전까지 반복적으로 문자를 화면에 출력함

따라서 문자열의 끝에 '\0'이 없으면 이후로 0을 만날 때까지 계속 출력함

즉, 배열의 각 요소를 문자로 선언하는 경우 끝에 '\0'을 붙여줘야 함

-문자와 문자열

문자: 문자는 작은따옴표(Single quote)로 둘러싸임

Ex) char c[3] = {'a', 'b', 'c'};

위와 같이 사용하면 문자들의 집합으로 간주됨

만일 c[2] = '\0';하면 문자열 "ab"로 처리됨

문자열: 문자열은 큰따옴표(Double quote)로 둘러싸임

Ex) char \*str = "abc";

위와 같이 사용하면 실제로는 'a', 'b', 'c' 뒤에 '\0'이 추가되며, 문자열로 간주됨

Ex) char str[4] = {'a', 'b', 'c', '\0'};

문자열을 배열에 넣을 때는 '\0' 자리를 고려하여 한 칸 여유 있게 선언해야 함

-문자열 상수

Ex) char \*p = "apple";

컴파일러는 문자열을 코드 세그먼트(code segment)에 저장

첫 문자를 가리키는 포인터를 반환

반환된 포인터가 좌변의 문자형 포인터 p에 대입됨

Ex) printf("%p", p); // 문자열의 시작 주소를 출력

Ex) printf("%c", \*p); // 'a' 출력

-문자열 상수와 문자열 배열 차이

문자열 상수: 코드 세그먼트(code segment)에 저장됨, printf("Hello, world\n");의 큰따옴표 안 문자열도 모두 코드 세그먼트에 저장됨, 코드 세그먼트에 저장된 데이터는 수정할 수 없음, 코드 세그먼트의 상수 영역은 읽기 전용(Read only)임을 기억하자

문자열 배열: 문자열 배열은 스택 세그먼트(stack segment)에 저장됨, 스택 세그먼트의 스택 프레임에 지역 변수 영역에 저장되므로 문자열 상수와 달리 수정할 수 있음, 따라서 문자열의 값을 변경해야 할 경우 반드시 문자열 배열로 선언해야 함

Ex) char str1[10] = "apple"; char \*str2 = "apple";

str1 = "pie"; // error

str2 = "pie"; // OK