포인터에 대하여

포인터란?

포인터란 어떤 변수나 배열을 가리키기 위하여 메모리상의 위치를 저장하는 변수이다. 간단히 포인터는 영어로 지시봉의 뜻을 가지고 있으니 무언가를 가리키는 변수라 생각하면 된다.포인터는 메모리에서 4byte를 차지한다.

포인터를 선언하는 방법

(변수의 타입) \*변수명; ex) int \*a; char \*ch; float \*f; 이런식으로 선언을 한다.

변수의 타입이 필요한 이유는 포인터는 할당받은 메모리중 가장 앞에 있는 주소만 기억을 하는데, 변수의 타입에 따라 읽는 방법과 할당받은 크기가 다르기 때문이다.

선언시 주의할점.

Int\* a 또는 int \* a 또는 int \*a 이렇게 쓰는 것이 모두 가능하지만 변수 여러 개를 동시에 선언시 어떤 것이 포인터인지 헷갈리는 경우가 발생하게 될 수 있으므로 int \*a로 쓰는 것을 추천합니다. ex) int\* a, b; 인 경우 a는 포인터지만 b는 일반 int형 변수이다. 만약 둘다 포인터이길 바라면 int \*a, \*b; 또는 int\* a, \*b; 또는이렇게 사용해야 한다.

#include <stdio.h>

int main()

{

int\* a,b;

a=&b;

b=1;

printf("%d", b);

return 0;

}

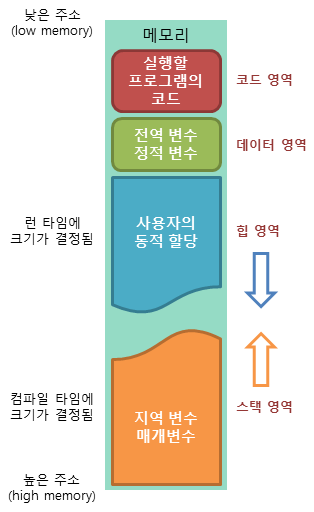
포인터의 배경 이해

컴퓨터의 메모리[[1]](#footnote-1)

|  |  |
| --- | --- |
| 메모리 속 값 | 메모리 주소 |
|  | 1000 |
| 1 | 1004 |
|  | 1008 |

옆의 표처럼 메모리가 있다고 하자.

이때 int a; 를 하여 a라는 변수를 만들었고, 이 변수는 1004 ~ 1007[[2]](#footnote-2)을 할당 받았다고 하자. 이때 우리는 a에 1이라는 값을 넣고 싶으면 a=1이라고 쓰고 그러면 컴퓨터는 a가 할당된 위치인 1004 ~ 1007 에 1이라는 수를 넣게 된다.[[3]](#footnote-3)

즉 우리는 메모리에 값을 저장하고 싶을 때 변수를 사용하여 메모리의 주소를 할당을 받고 그 공간에 원하는 값을 넣게 되는 것이고, 다르게 말하면 우리는 메모리의 주소를 알면 그 메모리에 있는 값을 사용 할 수 있는 것이다.

하지만 임의로 아무 메모리나 참조하게 되면 프로그램에 문제가 발생할 수 있고, 다른 프로그램한테 까지 영향을 미칠 수 가 있다.

프로그램을 실행하면 메모리를 오른쪽 그림과 같이 메모리를 할당을 하게 되는데 이때 코드영역에 프로그램의 코드가 올라오게 된다. 만약 포인터로 아무 주소를 참조하여 수를 바꾸다가 프로그램의 코드가 저장되어 있는 부분을 바꾸어 버리면 프로그램이 어떻게 될지 알 수 없다.

따라서 c언어에서는 주소값을 이용하여 값을 참조하기 위해서는 몇가지 제약을 걸게 되었고,[[4]](#footnote-4) 이에 따라 일반 변수로 주소값을 접근하는게 불가해 졌다. 그대신 메모리의 주소를 접근할 수 있는 전용 변수를 만들게 되었다.

포인터의 기본

&a는 a의 주소값

\*b는 b가 가리키는 곳의 주소값

포인터의 이해

|  |  |
| --- | --- |
| 메모리 속 값 | 메모리 주소 |
|  | 1000 |
| 1 | 1004 |
|  | 1008 |

Int \*a라는 포인터가 있다고 하자. 이때 a는 1000 ~ 1003을 할당 받았다고 하자. 그리고 int b가 1004 ~ 1007를 할당 받았고 b=1이라 하자.

현재 a에는 아무런 값이 들어 있지 않기 때문에 어느것도 가리키지 않고 있다. 무언가를 가리키기 위해서는 해당하는 메모리의 주소값을 입력을 받아야한다. 하지만 c에서는 잘못된 메모리 참조로 인한 참사를 막기위해서 제한을 하기 때문에 직접 숫자를 입력할 수 없고, 이미 있는 변수의 메모리 주소 값을 가져야 한다.

|  |  |
| --- | --- |
| 메모리 속 값 | 메모리 주소 |
| 1 | 1000 |
| 1 | 1004 |
|  | 1008 |

a를 사용하여 b라는 변수를 가리키기 위해서는

a=b라고 하면 안된다. 만약 이렇게 하면 말그대로 b에있는 1을 가지게 된다. 이렇게 되면 웬만해서는 잘못된 메모리 참조로 에러가 나게 된다.

이때 사용하는 것이 &인데 이는 단항연산자로 사용되면 해당 변수의 주소값을 알려준다.

|  |  |
| --- | --- |
| 메모리 속 값 | 메모리 주소 |
| 1004 | 1000 |
| 1 | 1004 |
|  | 1008 |

따라서 a=&b라고 사용하는 것이 바람직한 사용방법이다.

|  |  |
| --- | --- |
| 메모리 속 값 | 메모리 주소 |
| 1004 | 1000 |
| 1 | 1004 |
|  | 1008 |

Int \*a 라는 포인터가 있다고 하자. 이때 a는 1000 ~ 1003를 할당 받았고 b를 가리킨다고 하자. 그리고 int b가 1004 ~ 1007를 할당 받았고 b=1이라 하자.

|  |  |
| --- | --- |
| 메모리 속 값 | 메모리 주소 |
| 2 | 1000 |
| 1 | 1004 |

앞에서 말했던 것처럼 우리는 메모리의 주소값을 안다면 그 주소값에 있는 값을 참조할 수 있다고 했다.

|  |  |
| --- | --- |
| 메모리 속 값 | 메모리 주소 |
| 1004 | 1000 |
| 2 | 1004 |

하지만 a=2같은 것을 하면 a가 가리키고 있는 b의 값이 2가 되는 것이 아니라 말그대로 a가 할당 받은 주소의 값이 2로 바뀌게 된다.

따라서 우리는 a가 가리키고 있는 메모리를 참조하고 싶을 때 a=2 가 아닌 \*a=2를 사용하게 된다.

이때 \*a는 “a가 가리키는 주소의 값을 a가 선언될 때 설정된 자료형인 int라 생각을 하고 읽을 것 이다.”와 같다.[[5]](#footnote-5)

포인터의 연산

포인터 변수 + 1 = 포인터에 저장된 주소값 + 포인터 변수의 자료형의 크기

포인터 변수 - 1 = 포인터에 저장된 주소값 - 포인터 변수의 자료형의 크기

포인터 변수 + a = 포인터에 저장된 주소값 + 포인터 변수의 자료형의 크기 \* a

포인터 변수 - a = 포인터에 저장된 주소값 - 포인터 변수의 자료형의 크기 \* a

(단, a는 임의의 정수[[6]](#footnote-6))

포인터 변수 a – 포인터 변수 b =

(a에 저장된 주소값 – b에 저장된 주소값) / 포인터 변수의 자료형의 크기

이때 두 a, b의 자료형의 크기는 같아야 하고, 다를 경우 타입 에러가 난다.

포인터의 연산 이해

우리가 포인터에서 덧셈 뺄셈[[7]](#footnote-7)하는 이유는 배열을 이용할 때 다음 주소에 있는 값을 보고싶어서 할 때가 많다. 그런데 만약 연산을 할 때 자료형의 크기만큼 변하지 않고, 숫자 그대로 변한다면 우리는 배열의 다음 원소를 보기위해서 +1을 해주는 것이 아니라 + (포인터 변수의 자료형의 크기)를 직접 해주어야 한다. 이러한 불편함을 해소하기 위해서 연산을 자체적으로 자료형의 크기씩 계산을 해주게 된다.

우리가 포인터 변수끼리 뺄셈[[8]](#footnote-8)을 할 경우에는 두 변수 사이에 몇 개[[9]](#footnote-9)의 인덱스가 있는지를 원해서 하는 경우가 많다. 그런데 만약 숫자 사이의 연산처럼 나오면 직접 자료형의 크기로 나누어 주어야 한다. 이러한 불편함을 없애기 위해 값이 이렇게 나온다.

상수 포인터

const (변수의 타입) \*(변수명) : 포인터가 가리키는 주소의 값을 바꿀 수 없게 한다.

간단히 말해서 (변수의 타입)이 const(상수)인 것. 즉, 상수 자료형

ex) const int \*a는 상수 int를 가리키는 포인터 a

(변수의 타입) const \*(상수명) : 포인터가 가리키는 주소를 바꿀 수 없게 한다.

간단히 말해서 (상수명)이 const(상수)인 것. 즉, 상수

ex) int const \*a는 int를 가리키는 상수 a

포인터와 배열

배열

배열의 각 원소는 연속된 메모리의 공간을 차지한다. 왜냐하면 연속되어 있는 것이 찾거나 무언가를 할 때 편하기 때문이다. 비슷한 예로 책장에 책을 비슷한 것끼리 연속되게 꽂는 것이 있다. 이와 같이 한배열의 원소를 연속되게 뭉쳐 놓는 것이다.

이때 배열의 각 원소는 연속된 위치에 있으므로 포인터를 이용하여 주소를 하나씩 늘리거나 낮추면서 배열을 확인 할 수 있다.

배열은 (배열의 타입) const \*(배열이름)으로 정의 되어있고, 배열은 배열의 첫번째 원소 값을 가리킨다. 배열이 const인 이유는 배열은 “내 집은 이마트 옆이고 친구집은 내집 옆이야”처럼 기억하는데 만약 갑자기 이마트가 사라져서 이마트가 어디에 있었는지를 모르면 집을 찾아갈 수 없기 때문이다. 그렇기 때문에 절 때 바뀌면 안된다.

이때 배열이 포인터이므로 당연히 다른 포인터로도 배열을 사용할 수 가 있다.

밑에와 같이 배열을 했다고 하자.

int arr[4] = {1,2,3,4}; //arr의 첫번째 원소의 주소는 1000 이라고 하자.

int \*p = arr; // 이때 arr자체는 포인터이기 때문에 주소를 나타낸다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 주소 | 1000 | 1004 | 1008 | 1012 |
| 값 | 1 | 2 | 3 | 4 |

이때 p에 arr의 주소를 넣었기 때문에 arr[1] = p[1]이다.

2차 배열

우리는 2차배열을 인식할 때

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

이런식으로 생각을 하지만 컴퓨터에서 메모리상에 저장이 될때는 이를 1차원 배열처럼 저장을 한다.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

그렇기 때문에 2차배열임을 컴퓨터가 인식하기 위해서는 한열에 몇 칸인지를 알아야 한다.

따라서 2차원 배열의 포인터를 만들기 위해서는 한열에 몇 칸인지를 알려줘야한다.

따라서 선언시

int (\*a)[3] 이런식으로 선언을 하게 된다. 이때 ( )를 쓰는 이유는 포인터 배열을 선언하고 싶은것이 아니기 때문에 이렇게 한다.

포인터 배열

포인터도 하나의 변수의 일종 이므로 포인터들로 배열을 선언할 수 있다.

선언 방법 : (변수의 타입) \*(배열의 이름) [인덱스의 개수]

이중 포인터

포인터도 하나의 변수의 일종 이므로 이를 가리키는 포인터를 만들 수 있다.

선언 방법 : (변수의 타입) \*\*(변수명)

이때 \*\*을 붙이는 이유는 \*하나만 붙였을 때는 이 포인터가 가리키는 것은 일반 변수를 의미 하여 만약 포인터를 \*하나자리로 가리킬 시 주소값으로 생각하지않고, 일반 값으로 인식한다. 따라서 포인터가 가리키는 포인터의 가리키는 값을 확인하기 위해서는 \*\*로 해야한다.

함수

함수란 우리가 잘 아는 것처럼 일정한 입력값이 있을을 때 우리가 원하는 값이 나오도록 하는 것이다. 함수를 사용하면 반복적인 작업을 할 때 코드를 깔끔하게 해주거나 쓸데없이 같은내용을 계속 타이핑하는 수고를 줄일 수 있다. 또한 디버깅시 함수단위로 확인하면 되서 편리하다.

함수를 선언할때는

(리턴 타입) (함수명) ( (매개변수) )

{

함수의 내용

return (리턴하고 싶은 내용) //이때 위의 리턴 타입과 일치해야 하며, 타입이 void면 없어도 됨.

}

함수는 실행 시 main이 있는곳과 다른 메모리를 따로 할당을 하기 때문에 다른 함수에 있는 변수와 함수에 있는 변수의 이름이 같더라도 전혀 다른 변수이다 (지역변수).

그렇기 때문에 매개변수에 a라는 변수를 준 뒤 a를 변화시켜도 함수속의 a는 바뀌어도 원래 a는 바뀌지 않는다. (call by value)

하지만 다른 함수에서 원래 있던 변수의 값을 바꾸고 싶을 때가 있다. 이런 때에 포인터가 사용된다. (사실 함수가 없었으면 포인터가 없었을 것) (call by reference)

1. 메모리 주소가 4씩 커지는 이유는 int형을 사용할 때 4byte씩 사용하기 때문에 이렇게 나타냄 [↑](#footnote-ref-1)
2. int형은 4byte이기 때문에 4칸을 차지하게 된다. [↑](#footnote-ref-2)
3. 실제론 컴파일이 된 후 실행이 될 때는 컴퓨터는 a라는 변수명을 기억하여 사용하지 않고 메모리 주소값인 1004를 사용한다. (이때 int형은 4byte이기 때문에 1004부터 4자리를 읽게 된다.) 즉 변수 a라는 것은 사람이 직접 메모리 주소를 써가면서 프로그램 하기에 힘들기 때문에 잠시 별명을 사용하는 것과 같다. [↑](#footnote-ref-3)
4. 이게 싫다면 보안이 낮은 옛날 운영체제를 사용하고 어셈블리어를 사용하면 된다! [↑](#footnote-ref-4)
5. 만약 int \*a가 아니라 char \*a로 선언 됐다면 char라 생각하고 값을 읽을 것이다. 이때 char은 1byte만 사용하는 타입이므로 b가 아래 표처럼 저장되어 있다고 하면, a가 가리키는 1004부터 1byte를 읽어 2를 참조하게 된다.

   |  |  |
   | --- | --- |
   | 메모리 속 값 | 메모리 주소 |
   | 2 | 1004 |
   | 0 | 1005 |
   | 0 | 1006 |
   | 0 | 1007 |

   어떤 엔디안으로 되어 있는지에 따라 저장되는 방법이 다르므로 다른 결과가 있을 수 있다. [↑](#footnote-ref-5)
6. 정수가 아니면 에러가 난다. [↑](#footnote-ref-6)
7. 즉, 주소값과 일반 정수의 덧셈 뺄셈 (참고로 곱셈, 나눗셈은 불가하다.) [↑](#footnote-ref-7)
8. 즉, 주소값과 주소값의 뺄셈 (참고로 덧셈, 곱셈, 나눗셈은 불가하다.) [↑](#footnote-ref-8)
9. 0부터 센다. [↑](#footnote-ref-9)