포인터

- 메모리를 위주로 -

안태진(taejin@codecure.smuc.ac.kr)

상명대학교 보안동아리 CodeCure

목 차

포인터란?

- 메모리 주소 (Memory Address)
  - 변수(Variable): 특정한 값이 저장되는 공간
  - 메모리(Memory): 변수가 저장되는 공간
    - CPU가 메모리에 접근할 때에는 주소로 접근하게 됨
  - 메모리 주소 또한 특정한 값 => 저장을 위한 변수가 필요
- 포인터 (Pointer)
  - 메모리 주소를 저장하고 있는 변수
  - 포인터 변수의 크기
    - 32bit: 4bytes
    - 64bit: 8bytes

목 차

포인터 활용법

- 포인터 변수 선언 (Declaration)
  - Type* ValName;
    - *의 위치는 타입과 이름의 사이면 됨
    - e.g., int* intPtr, char* charPtr
  - 보통 메모리 주소를 반환하는 연산자(&)와 같이 사용

```
int num = 10;
int* numPtr = &num;
```

목 차

포인터 활용법

- 역참조 (Dereference)
  - 포인터 변수의 값으로 들어가 값을 읽어내는 방법
  - 사용
    - *ptr
    - e.g., printf(“%d\n”, *numPtr);

목 차

포인터 활용법

- 포인터의 자료형 (Type)
  - 자료형을 쓰는 이유
    - 역참조 했을 때 몇 바이트를 읽어올 것인지,
      읽어온 데이터를 정수로 판단할 것인지

```c
// TypeExample.c
#include <stdio.h>

int main(void) {
    int num = 0x12345678;

    int* intPtr = &num;
    char* charPtr = intPtr;

    printf("%#x\n", *intPtr);
    printf("%#x\n", *charPtr);

    float f1 = 0.123456f;
    intPtr = &f1;
    float* floatPtr = &f1;

    printf("%d\n", *intPtr);
    printf("%f\n", *floatPtr);

    return 0;
}
```

```
jin-desk@JIN-DESK:
0x12345678
0x78
1039980160
0.123456
```

목 차

- 포인터란? (Pointer)
  - 메모리 주소
  - 포인터
- 포인터 활용법
  - 포인터 변수 선언
  - 역참조 (Dereference)
  - 포인터의 자료형 (Type)
  - 리틀 엔디언 vs 빅 엔디언 (Little, Big Endian)
- 포인터 심화
  - 포인터 연산 (Pointer Arithmetic)
  - 배열과의 연관성
  - 그외

포인터 활용법

- 리틀 엔디언 vs 빅 엔디언 (Little, Big Endian)
  - 리틀 엔디언 (Little Endian)
    - 작은 단위의 바이트가 앞에 오도록 저장하는 방법
  - 빅 엔디언 (Big Endian)
    - 큰 단위의 바이트가 앞에 오도록 저장하는 방법

```c
// EndianExample.c
#include <stdio.h>

int main(void) {
    char c1 = 0x12;
    short s1 = 0x1234;
    int i1 = 0x12345678;
    long long int lli1 = 0x1234567890123456;

    return 0;
}
```

```
gdb-peda$ x/b &c1
0x7fffffffee261: 0x12
gdb-peda$ x/2b &s1
0x7fffffffee262: 0x34    0x12
gdb-peda$ x/4b &i1
0x7fffffffee264: 0x78    0x56    0x34    0x12
gdb-peda$ x/8b &lli1
0x7fffffffee268: 0x56    0x34    0x12    0x90    0x78    0x56    0x34    0x12
```

목 차

포인터 심화

- 포인터 연산 (Pointer Arithmetic)
  - 문제 (intPtr)이 0x500이라고 할 때 (intPtr + 1)의 결과는?

1) 0x501
2) 0x502
3) 0x503
4) 0x504
5) 0x510

포인터 심화

- 포인터 연산 (Pointer Arithmetic)

```c
// PointerArithmetic.c
#include <stdio.h>

int main(void) {
    char* charPtr = 0x1000;
    short* shortPtr = 0x1000;
    int* intPtr = 0x1000;
    long long int* longPtr = 0x1000;

    printf("%#x\n", charPtr+1);
    printf("%#x\n", shortPtr+1);
    printf("%#x\n", intPtr+1);
    printf("%#x\n", longPtr+1);

    return 0;
}
```

```
jin-desk@JIN-DESK:
0x1001
0x1002
0x1004
0x1008
```

목 차

- 포인터란? (Pointer)
  - 메모리 주소
  - 포인터
- 포인터 활용법
  - 포인터 변수 선언
  - 역참조 (Dereference)
  - 포인터의 자료형 (Type)
  - 리틀 엔디언 vs 빅 엔디언 (Little, Big Endian)
- 포인터 심화
  - 포인터 연산 (Pointer Arithmetic)
  - 배열과의 연관성
  - 그외

포인터 심화

- 배열과의 연관성
  - 배열의 이름을 포인터처럼 사용 가능
  - 포인터를 배열처럼 사용 가능

  - arr[i] == *(arr + i) == ptr[i] == *(ptr + i)
  - i[arr] == *(i + arr) == [i]ptr == *(i + ptr)

포인터 심화

- 배열과의 연관성

```c
// RelationArry.c
#include <stdio.h>

int main(void) {
    int arr[10]
    = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

    int* ptr = arr;

    printf("sizeof arr: %ld\n", sizeof(arr));
    printf("sizeof ptr: %ld\n", sizeof(ptr));

    printf("%d\n", arr[2]);
    printf("%d\n", ptr[2]);
    printf("%d\n", *(arr + 2));
    printf("%d\n", *(ptr + 2));

    printf("%d\n", 2[arr]);
    printf("%d\n", 2[ptr]);

    return 0;
}
```

```
jin-desk@JIN-DESK:
sizeof arr: 40
sizeof ptr: 8
3
3
3
3
3
3
```

목 차

- 포인터란? (Pointer)
  - 메모리 주소
  - 포인터
- 포인터 활용법
  - 포인터 변수 선언
  - 역참조 (Dereference)
  - 포인터의 자료형 (Type)
  - 리틀 엔디언 vs 빅 엔디언 (Little, Big Endian)
- 포인터 심화
  - 포인터 연산 (Pointer Arithmetic)
  - 배열과의 연관성
  - 그외

포인터 심화

- 더블 포인터 (Double Pointer, 이중 포인터)
  - 포인터를 담고 있는 포인터
- 배열 포인터 vs 포인터 배열
  - Int (*intPtr)[4] vs int * intPtr[4]

참고 자료

- C언어 코딩도장
  - dojang.io
- 윤성우의 열혈 C프로그래밍(개정판)
- 교수님들의 강의
- 내 머리 (I’m a Genius)

감사합니다!