포인터와 동적 배열
(Chapter 9 Pointers and Dynamic Arrays)

숭실대학교
김강희 교수
(khkim@ssu.ac.kr)

순서

- 이론:
  - 포인터
  - 동적 배열
  - this 포인터
  - 소멸자
  - 복사 생성자

포인터(Pointer)란

- 포인터는 어떤 변수의 메모리 주소이다.
  - 메모리는 바이트의 1차원 배열로 취급된다.
  - 메모리 주소는 해당 변수의 이름으로 사용될 수 있다.
  - call-by-reference 방식의 함수 인자 전달은 사실상 실질 인자의 주소값을 전달하고 있다.
- 포인터는 (어떤 타입의 변수를 가리키든지 간에) 64비트 CPU 의 경우 8바이트 크기의 변수이다.
  - 포인터 자체는 메모리 주소값을 갖는 변수이기 때문이다.
    - 그러나, 주소값은 정수형으로 취급되지 않는다.
  - 타겟 주소에 어떤 타입의 변수가 있는지는 포인터에 "타입"을 부여함으로써 표현한다.
    - double *p; // 포인터 p 의 타겟 주소에서는 double 타입의 변수가 존재한다.
    - int *p, *q; // 포인터 변수 앞에는 반드시 '*'를 표기해야 한다.

포인터(Pointer)란

- & 연산자
  - 어떤 변수의 주소값을 표현하는 연산자이다.
  - call-by-reference 방식으로 전달되는 함수 인자 앞에도 사용된다.
  - 예시: int *p, v, w;

```
p = &v;
// 해석1: 포인터 변수 p 는 변수 v 의 주소와 같다.
// 해석2: 포인터 변수 p 는 변수 v 를 가리킨다.
```

- * 연산자
  - 포인터 변수가 가리키는 변수의 값을 꺼내오는 연산자이다. ('역참조'는 오역에 가까우니, 영어 발음대로 "dereference" 연산자로 발음하는 것이 좋을 듯)
  - 예시: w = v;

```
w = *p;
```

  - 그러나 다음 코드에서 * 는 변수의 값이 아니라 변수의 레퍼런스를 의미한다

```
*p = 10;
```

포인터(Pointer)란

- 포인터 변수들은 서로 assign 가능하다.
  - 예시: int v, *p1, *p2;

    p1 = &v; // p1 은 v 를 가리킨다.

    p2 = p1; // p2 도 v 를 가리킨다.

    // 다음 코드와 혼동하지 마라.

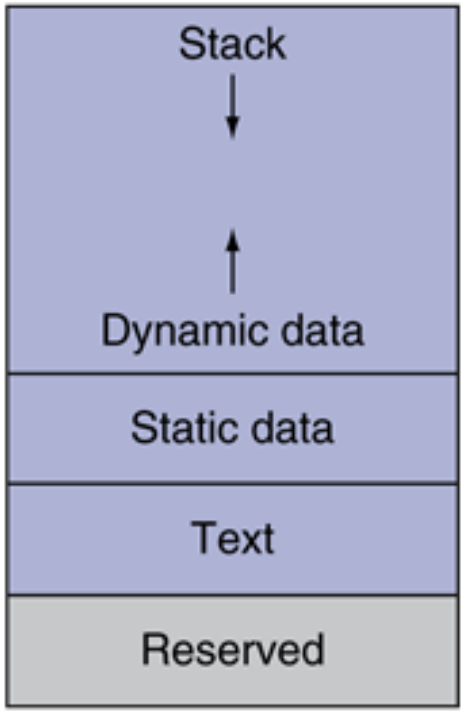
    *p2 = *p1
- 동적 할당(dynamic allocation, or heap allocation)하는 변수들을 가리키는 유일한 방법이다.
  - 예시: int *p;

    p = new int;

    // new 는 nameless 정수형 변수를 할당하고 주소를 리턴한다.
  - new 연산자는 할당되는 변수의 초기화를 함께 수행할 수 있다.
    - 예시: p = new int(10); // *p 를 10으로 초기화한다.
- 포인터는 함수의 인자형 또는 리턴형으로도 사용될 수 있다.

new 연산자

- heap 메모리 영역으로부터 변수를 할당한다.
  - 오른쪽 그림의 'dynamic data' 영역이 heap 영역이다.
  - stack 영역이 자라나고, heap 영역이 자라나서 충돌하거나 또는 물리 메모리가 부족하면 더 이상 변수를 할당할 수 없다.
  - 이 때, new 연산자는 NULL 을 리턴한다.
- new 연산자의 리턴값을 다음과 같이 체크하라.
  - 최신 컴파일러는 다음과 같은 코드가 없어도, NULL 를 스스로 체크하고 오류 메시지 출력 후 프로그램을 자동 종료시킨다.




```
int *p;
p = new int;
if (p == NULL)
{
    cout << "Error: Insufficient memory.₩n";
    exit(1);
}
```

C++11 nullptr

- NULL 은 사실상 정수 0을 의미한다. 그렇기 때문에 다음 경우에서 혼선을 일으킨다.
  - myfunc(NULL) 호출시 컴파일러는 다음 어느 함수를 호출하도록 코드 생성을 해야할까?
    - void myfunc(int i)
    - void myfunc(int *p)
- C++11 에서는 nullptr 이라는 상수를 도입함으로써 이 문제를 해결하였다.
  - nullptr 는 정수 0 이 아니고 포인터형 상수이다.

delete 연산자

- 동적 할당된 변수를 소멸시킨다.
  - 예시: int *p = new int;

    delete p; // p 가 가리키는 정수형 변수를 소멸시킨다.
- 이 때 포인터 p 는 소멸된 변수의 주소를 여전히 가지고 있는데(dangling pointer 라고 부른다), 이 주소를 참조하지 않도록 조심하라!!!
  - 예시: delete p;

    p = NULL; // p 가 dangling pointer 가 되지 않도록 NULL 지정함

Dynamic vs. Automatic Variables

- Dynamic variables (동적 변수들)
  - heap 영역에 할당된 변수들이다.
  - new 연산자로 할당하고, delete 연산자로 소멸시킨다.
- Automatic variables (자동 변수들)
  - stack 영역에 할당된 변수들이다.
  - 함수 시작 ('{') 시 할당되고, 함수 종료 ('}') 시 소멸된다.
  - local variables (지역 변수들)이라고도 부른다.

포인터 타입 정의

- 포인터형을 '*' 없는 다른 변수형과 동일하게 취급할 수 있다.
- 예시: typedef int* IntPtr;

```
IntPtr p; // 이 선언은 다음 선언과 동일하다.
int *p;
```