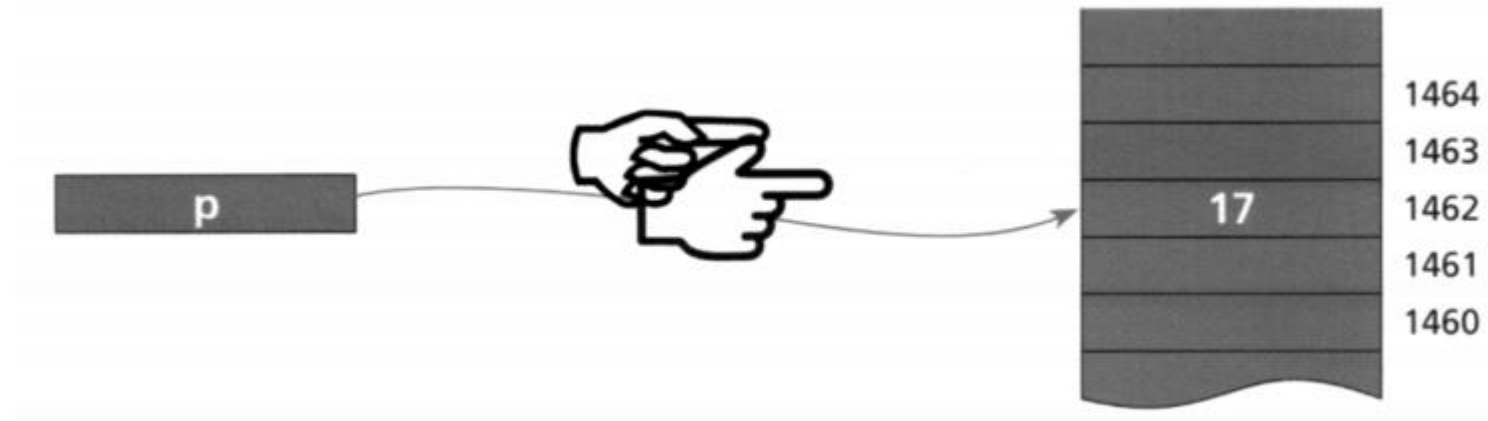


포인터와 동적 객체 생성

포인터

■ 포인터(pointer)

- 메모리의 주소값을 저장하는 변수
- 변수 선언시 타입 뒤에 *를 지정
- `int *p; // 정수를 가리키는 포인터 선언`



포인터

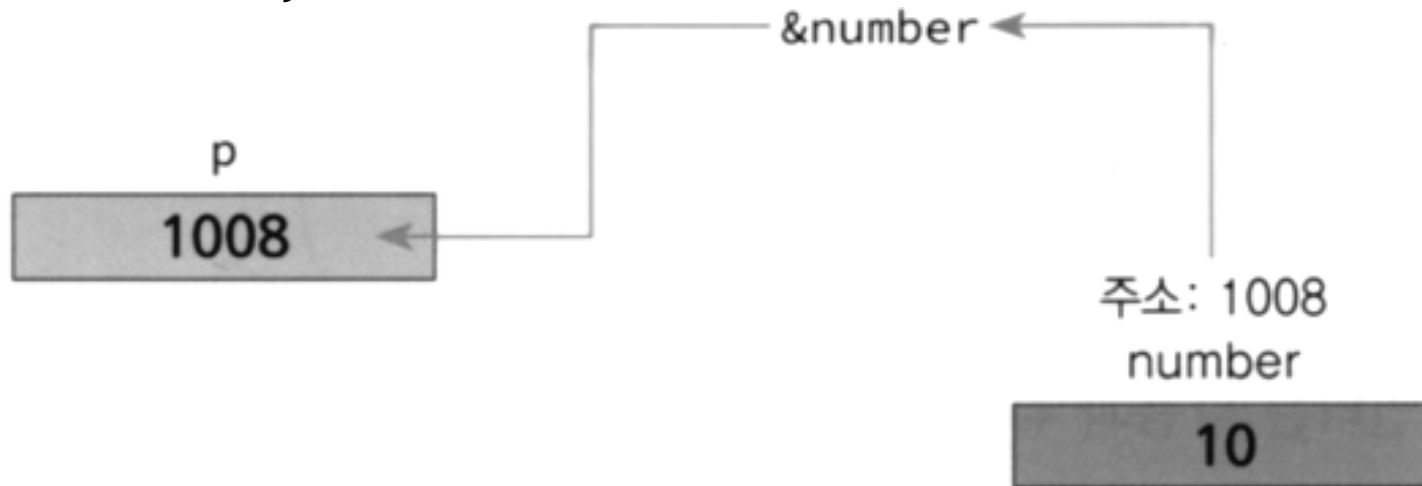
■ 주소 연산자 &

- 기존 변수의 주소 값을 획득하여 포인터 변수에 저장할 때 사용

- `int number = 10;`

`int *p; // 정수를 가리키는 포인터 선언`

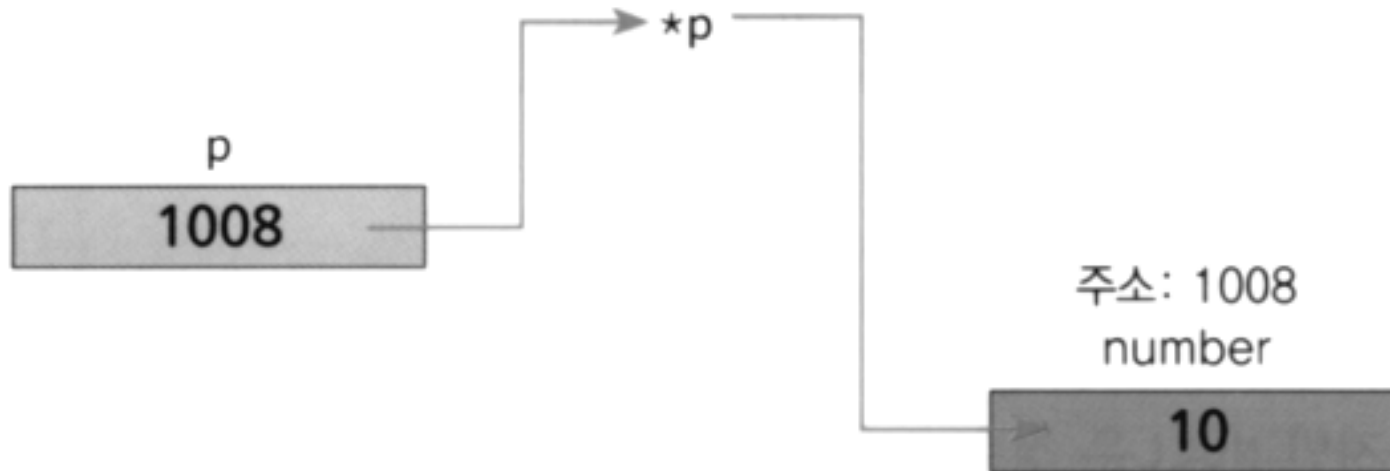
`p = &number;`



포인터

■ 간접 참조 연산자 *

- 포인터 변수에는 주소가 저장되어있음
- 그 주소에 저장되어 있는 데이터를 얻을 때 사용



포인터

■ chapter07/ex01_pointer.cpp] 포인터

```
#include <iostream>
using namespace std;

int main() {
    int number = 0;
    int *p = &number;

    cout << p << endl;
    cout << *p << endl;

    return 0;
}
```

0x7ffdde01928c

0

포인터

■ NULL

- 포인터가 아무것도 가리키지 않는 것을 의미하는 특수한 데이터
 - 0으로 해석되므로 int이기도 하면서 포인터 이기도 함
- 포인터 변수를 초기화할 때 사용
- nullptr 사용 가능
 - 포인터로만 해석

포인터

■ chapter07/ex02_null.cpp] NULL

```
#include <iostream>
using namespace std;

void f(int i) {
    cout << "f(int)" << endl;
}

void f(char *p) {
    cout << "f(char *)" << endl;
}

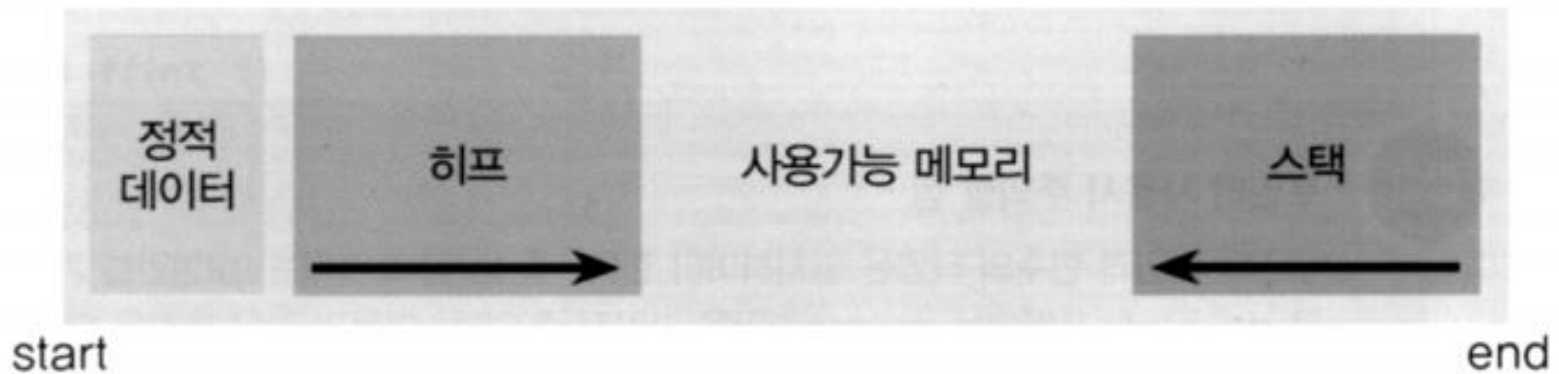
int main() {
    // f(NULL); -- int, char * 둘 다 가능하므로 에러
    f(nullptr);
    return 0;
}
```

f(char *)

동적 할당 메모리

■ 프로그램에서의 메모리

- 스택(Stack)
 - 지역 변수 할당
 - 시스템에 의해 관리
- 힙(Heap)
 - 동적 메모리 할당
 - 개발자에 의해 관리



동적 할당 메모리

■ new, delete

- new
 - 동적으로 힙에 메모리를 할당
- delete
 - 동적으로 힙에 할당된 메모리를 회수
 - 파괴자가 호출됨
 - 동적 메모리를 회수 하지않으면 가비지(garbage) 증가 -- 메모리 누수
-

```
class T {};
```

```
T *p = new T;
```

```
T *p = new T[N];
```

```
T *p = new T[N] { initializer1, ... , initializeerN};
```

동적 할당 메모리

■ new, delete

○

```
int *p;
```

```
p = new int[5];
```



```
int *p = new int[5] {0, 1, 2, 3, 4};
```

동적 할당 메모리

■ new, delete

○

```
int *p = new int;  
:  
delete p;  // 단일 데이터 삭제
```

```
int *p = new int[5] {0, 1, 2, 3, 4};  
  
delete [] p;  // 배열 데이터 삭제
```

동적 할당 메모리

■ chapter07/ex03_new_delete.cpp] new, delete

```
#include <iostream>
#include <time.h>
using namespace std;

int main() {
    int *ptr;

    srand(time(NULL));
    ptr = new int[10];
    for(int i=0; i<10; i++) {
        ptr[i] = rand();
    }

    for(int i=0; i<10; i++) {
        cout << ptr[i] << " ";
    }
    cout << endl;
    delete []ptr;
    return 0;
}
```

동적 할당 메모리

■ chapter07/ex04_garbage.cpp] 가비지(garbage)

```
#include <iostream>
#include <time.h>
using namespace std;

int main() {
    int *ptr = new int;

    *ptr = 99;

    return 0;
}
```

동적 할당 메모리

■ 스마트 포인터

- 포인터의 동적 메모리 회수를 자동으로 처리해줌
 - 포인터 변수가 제거될 때 자동으로 delete 호출
- `#include <memory>` 추가 후 사용
- `unique_ptr`
 - 포인터에 대해 오직 하나의 소유자만 허용
- `unique_ptr<int[]> buf(new int[10]);`
- `shared_ptr`
 - 참조 횟수가 계산되는 스마트 포인터

동적 할당 메모리

■ chapter07/ex05_smart_pointer.cpp] 스마트 포인터

```
#include <iostream>
#include <memory>

using namespace std;

int main() {
    unique_ptr<int[]> buf(new int[10]);

    for(int i=0; i<10; i++) {
        buf[i] = i;
    }

    for(int i=0; i<10; i++) {
        cout << buf[i] << " ";
    }
    cout << endl;
    return 0;
}
```

객체의 동적 생성

■ 객체의 동적 생성

- 객체에 대해서도 동일한 원칙 적용
- 멤버 접근에 대한 표현이 다름
 - 포인터_변수->멤버
-

```
Dog *pDog = new Dog;
```

```
pDog->age = 10;
```

```
:
```

```
delete pDog;
```


객체의 동적 생성

■ chapter07/ex06_dynamic.cpp] 객체의 동적 생성

```
#include <iostream>
#include <string>
using namespace std;

class Dog {
public :
    int age;
    string name;

    Dog() {
        cout << "Dog 생성자 호출" << endl;
        age = 1;
        name = "바둑이";
    }

    ~Dog() {
        cout << "Dog 소멸자 호출" << endl;
    }
};
```

객체의 동적 생성

■ chapter07/ex06_dynamic.cpp] 객체의 동적 생성

```
int main() {  
    Dog *pDog = new Dog;  
    delete pDog;  
    return 0;  
}
```

Dog 생성자 호출

Dog 소멸자 호출

객체의 동적 생성

■ 포인터로 객체 멤버 접근하기

```
(*pDog).getAge();
```

```
pDog->getAge(); // 포인터로 멤버 접근할 때 -> 사용
```

객체의 동적 생성

■ chapter07/ex07_pointer.cpp] 포인터로 객체 멤버 접근하기

```
#include <iostream>
#include <string>
using namespace std;

class Dog {
public :
    int age;
    string name;

    Dog() {
        age = 1;
        name = "바둑이";
    }

    ~Dog() { }

    int getAge() { return age;}
    void setAge(int a) { age = a; }
};
```

객체의 동적 생성

■ chapter07/ex07_pointer.cpp] 포인터로 객체 멤버 접근하기

```
int main() {  
    Dog *pDog = new Dog;  
  
    cout << "강아지의 나이: " << pDog->getAge() << endl;  
  
    pDog->setAge(3);  
    cout << "강아지의 나이: " << pDog->getAge() << endl;  
  
    delete pDog;  
    return 0;  
}
```

강아지의 나이: 1
강아지의 나이: 3

객체의 동적 생성

■ chapter07/ex08_dynamic_member.cpp] 멤버도 동적 생성하기

```
#include <iostream>
#include <string>
using namespace std;

class Dog {
private:
    int *pAge;
    int *pWeight;
public:
    Dog() {
        pAge = new int{1};
        pWeight = new int{10};
    }
    ~Dog() {
        delete pAge;
        delete pWeight;
    }
    int getAge() { return *pAge;}
    void setAge(int a) { *pAge = a; }
    int getWeight() { return *pWeight;}
    void setWeight(int w) { *pWeight = w; }
};
```

객체의 동적 생성

■ chapter07/ex08_dynamic_member.cpp] 멤버도 동적 생성하기

```
int main() {  
    Dog *pDog = new Dog;  
  
    cout << "강아지의 나이: " << pDog->getAge() << endl;  
  
    pDog->setAge(3);  
    cout << "강아지의 나이: " << pDog->getAge() << endl;  
  
    delete pDog;  
    return 0;  
}
```

강아지의 나이: 1
강아지의 나이: 3

객체의 동적 생성

■ this 포인터

- 모든 객체가 가지는 멤버 변수
- 자신(인스턴스)에 대한 포인터 변수
- 멤버 변수와 매개 변수의 이름이 같은 경우 멤버 변수를 지칭하기 위해 사용

객체의 동적 생성

■ chapter07/ex09_this.cpp] this 포인터

```
#include <iostream>
#include <string>
using namespace std;

class Rectangle {
private:
    int length;
    int widht;

public:
    Rectangle() {
        length = 30;
        widht = 40;
    }
    ~Rectangle() {}
    void setLength(int length) { this->length = length;}
    int getLength() { return this->length; }
    void setWidth(int widht) { this->widht = widht;}
    int getWidth() { return this->widht; }
};
```

객체의 동적 생성

■ chapter07/ex09_this.cpp] this 포인터

```
int main() {  
    Rectangle rect;  
    cout << "사각형의 길이: " << rect.getLength() << endl;  
    cout << "사각형의 너비: " << rect.getWidth() << endl;  
  
    rect.setLength(20);  
    rect.setWidth(10);  
  
    cout << "사각형의 길이: " << rect.getLength() << endl;  
    cout << "사각형의 너비: " << rect.getWidth() << endl;  
    return 0;  
}
```

사각형의 길이: 30
사각형의 너비: 40
사각형의 길이: 20
사각형의 너비: 10

객체의 동적 생성

■ const 포인터

- `const int *p1;` `// 내용 수정 불가`
- `int * const p2;` `// 주소 수정 불가`
- `const int * const p3;` `// 내용, 주소 모두 수정 불가`