04

객체 포인터와 객체 배열, 객체의 동적 생성

학습 목표

1. 객체에 대한 포인터를 선언하고 활용할 수 있다.
2. 객체의 배열을 선언하고 활용할 수 있다.
3. new를 이용하여 동적으로 메모리나 배열을 할당 받고 delete를 이용하여 반환할 수 있다.
4. new를 이용하여 동적으로 객체나 객체 배열을 할당 받고 delete를 이용하여 반환할 수 있다.
5. this 포인터의 개념을 이해하고, 활용할 수 있다.
6. string 클래스를 이용하여 문자열을 다룰 수 있다.

메모리의 구성

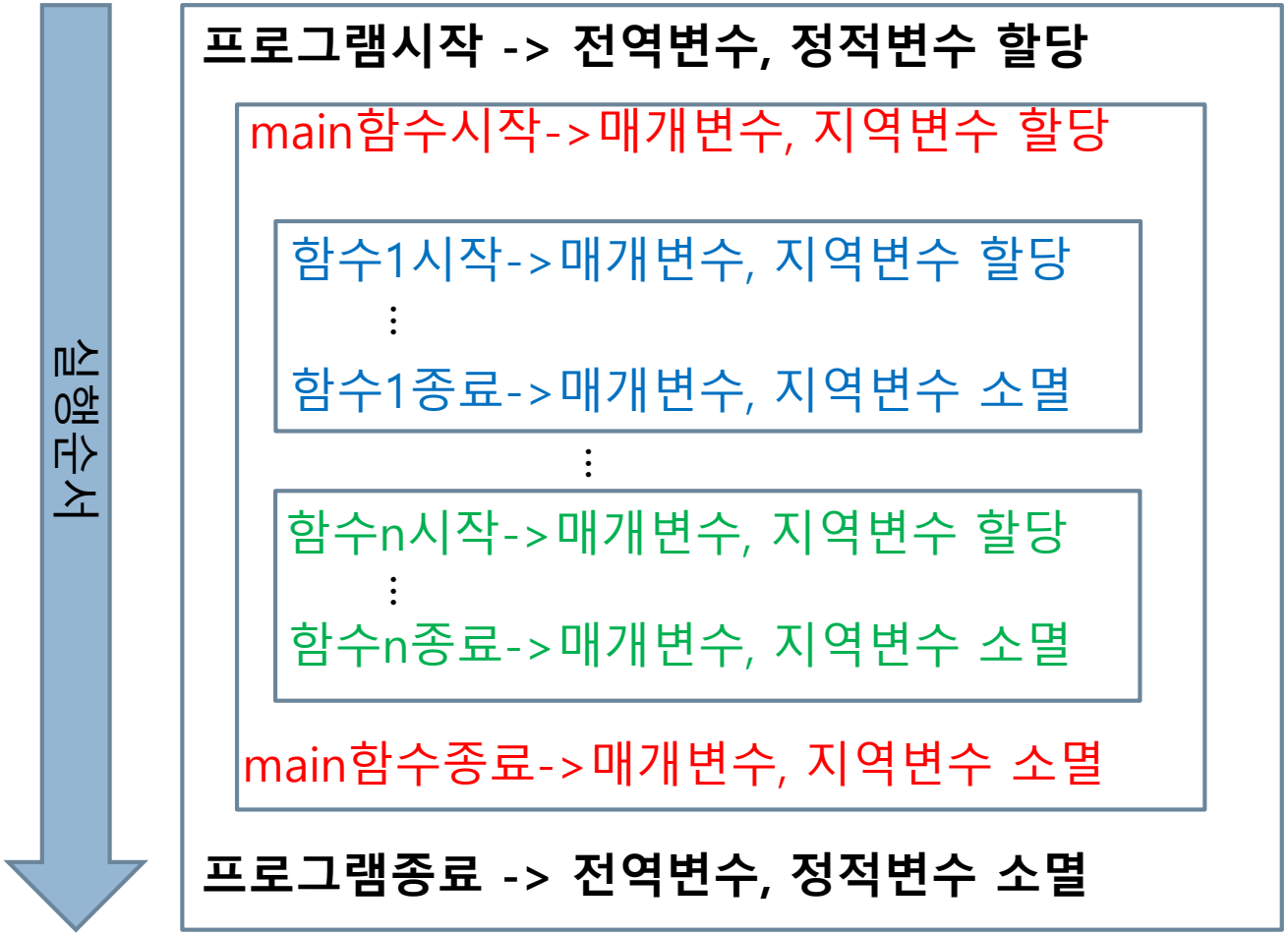
- 운영체제는 메모리를 코드 영역, 데이터 영역, 힙(Heap) 영역, 스택(stack) 영역으로 나누어 관리 -> 유사한 성격의 데이터를 묶어서 저장을 하면, 관리가 용이해지고 메모리의 접근속도가 향상

코드 영역	실행할 프로그램의 코드가 저장되는 메모리 공간 -> CPU는 코드 영역에 저장된 명령문을 하나씩 가져다가 실행
데이터 영역	전역변수와 static 변수가 할당되는 영역 -> 프로그램 시작과 동시에 할당되어 종료 시까지 남아있음
힙 영역	프로그래머가 원하는 시점에 메모리 공간에 할당 및 소멸을 하기 위한 영역
스택 영역	지역변수와 매개변수가 할당되는 영역 -> 함수 호출시 할당되고 함수 종료시 소멸되는 변수를 저장하는 영역

메모리의 구성


```
#include <iostream>
using namespace std;
int x;
int main() {
    char str[100];
    cout << "문자열 입력하세요: ";
    cin >> str;
    cout << "문자열 : " << str << endl;
    return 0;
}
```

메모리 할당 및 소멸순서


실행순서
프로그램시작 -> 전역변수, 정적변수 할당
main함수시작->매개변수, 지역변수 할당
함수1시작->매개변수, 지역변수 할당
함수1종료->매개변수, 지역변수 소멸
함수n시작->매개변수, 지역변수 할당
함수n종료->매개변수, 지역변수 소멸
main함수종료->매개변수, 지역변수 소멸
프로그램종료 -> 전역변수, 정적변수 소멸

메모리 할당의 문제점

- 전역변수, 정적변수는 프로그램 시작 시에 메모리가 할당되고 프로그램 종료 시에 소멸
- 매개변수, 지역변수는 함수 호출시 할당되고 함수 종료시 소멸
- **소스코드 작성단계(컴파일 단계)에서 메모리 크기와 위치가 모두 결정됨** -> 프로그램 실행 도중에는 메모리 크기를 변경할 수 없음
- 컴파일시 할당된 크기보다 프로그램 실행시 더 큰 데이터가 입력된다면 메모리 부족, 더 작은 데이터가 입력된다면 메모리 공간이 낭비 -> 비효율적

메모리 할당의 문제점

- 배열의 크기는 소스코드 작성시 고정되고 변경이 불가능함

```
#include <iostream>
using namespace std;
int main() {
	int n[4];		// 컴파일시 크기 고정
	for (int i = 0; i < 4; i++) {
		cout << i + 1 << "번째 정수: ";
		cin >> n[i];
	}
	int sum = 0;
	for (int i = 0; i < 4; i++) sum += n[i];
	cout << "평균 = " << sum / 4 << endl;
	return 0;
}
```

```
1번째 정수: 4 <엔터>
2번째 정수: 20 <엔터>
3번째 정수: -5 <엔터>
4번째 정수: 9 <엔터>
평균 = 7
```

메모리 할당의 문제점

- 소스코드 작성시 사용자가 입력할 문자열의 길이를 알 수 없음 -> 배열의 크기가 작으면 데이터를 저장할 수 없고 크기가 너무 크면 메모리 낭비 발생

```
#include <iostream>
using namespace std;
int main() {
	char str[100];		// 컴파일시 크기 고정
	cout << "문자열 입력하세요: ";
	cin >> str;
	cout << "문자열 : " << str << endl;
	return 0;
}
```

```
문자열을 입력하세요 : Hello <엔터>
문자열 : Hello
```

동적 메모리 할당

- 프로그램 실행 도중에 원하는 크기만큼 메모리를 할당 받고 사용이 끝나면 실행 도중에 메모리를 반납
- 메모리 할당과 반납을 실행 중 원하는 시점에 할 수 있음
- 필요한 만큼만 할당을 받고 사용 후 반납하므로 메모리를 매우 효율적으로 사용
- 힙(heap) 영역에 할당, 힙은 운영체제가 소유하고 관리하는 메모리

동적 메모리 할당 및 반환

- C언어의 동적 메모리 할당/반환 : malloc, free 함수 사용
- C++의 동적 메모리 할당/반환 : new, delete 연산자 사용
  - new 연산자
    - 기본 자료형, 구조체, 객체의 동적 할당
    - 힙 메모리에서 메모리를 할당 받음
    - 객체의 동적 할당 시 생성자 자동 호출
  - delete 연산자
    - new로 할당 받은 메모리 반환(해제)
    - 객체를 반환시 소멸자 호출 뒤 할당 받은 공간을 힙에 반환
  - C언어의 malloc/free 함수에 비하여 연산자가 사용하기 쉬움

new와 delete 연산자

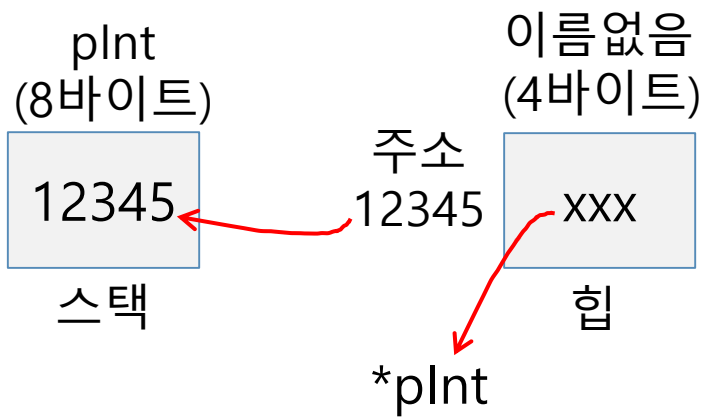
- new/delete 연산자를 이용한 동적할당 및 해제

```
자료형* 포인터 = new 자료형 ;
delete 포인터;
```

  - 자료형의 크기만큼 힙영역에 할당 받고 메모리 주소를 리턴함
  - 동적할당 받은 메모리는 이름은 없고 주소로만 접근가능

- new/delete의 사용 예

```
int* pInt = new int;
*pInt = 10;
…
delete pInt;
```



new와 delete 연산자

- new/delete의 사용 예

```
int* pInt = new int;                    // int 타입의 메모리 동적 할당
char* pChar = new char;                 // char 타입의 메모리 동적 할당
double * pDouble = new double;          // double 타입의 동적 할당

*pInt = 100;
*pChar = 'a';
*pDouble = 3.14;
...
delete pInt;                            // 할당 받은 정수 공간 반환
delete pChar;                           // 할당 받은 문자 공간 반환
delete pCircle;                         // 할당 받은 객체 공간 반환
```

new와 delete 연산자

- 힙 메모리가 부족하면 new연산자는 NULL(널포인터)을 리턴
- 연산결과가 NULL일때는 에러메시지 출력하고 프로그램을 종료
- 헤더파일 <iostream>에 선언 -> #define NULL 0

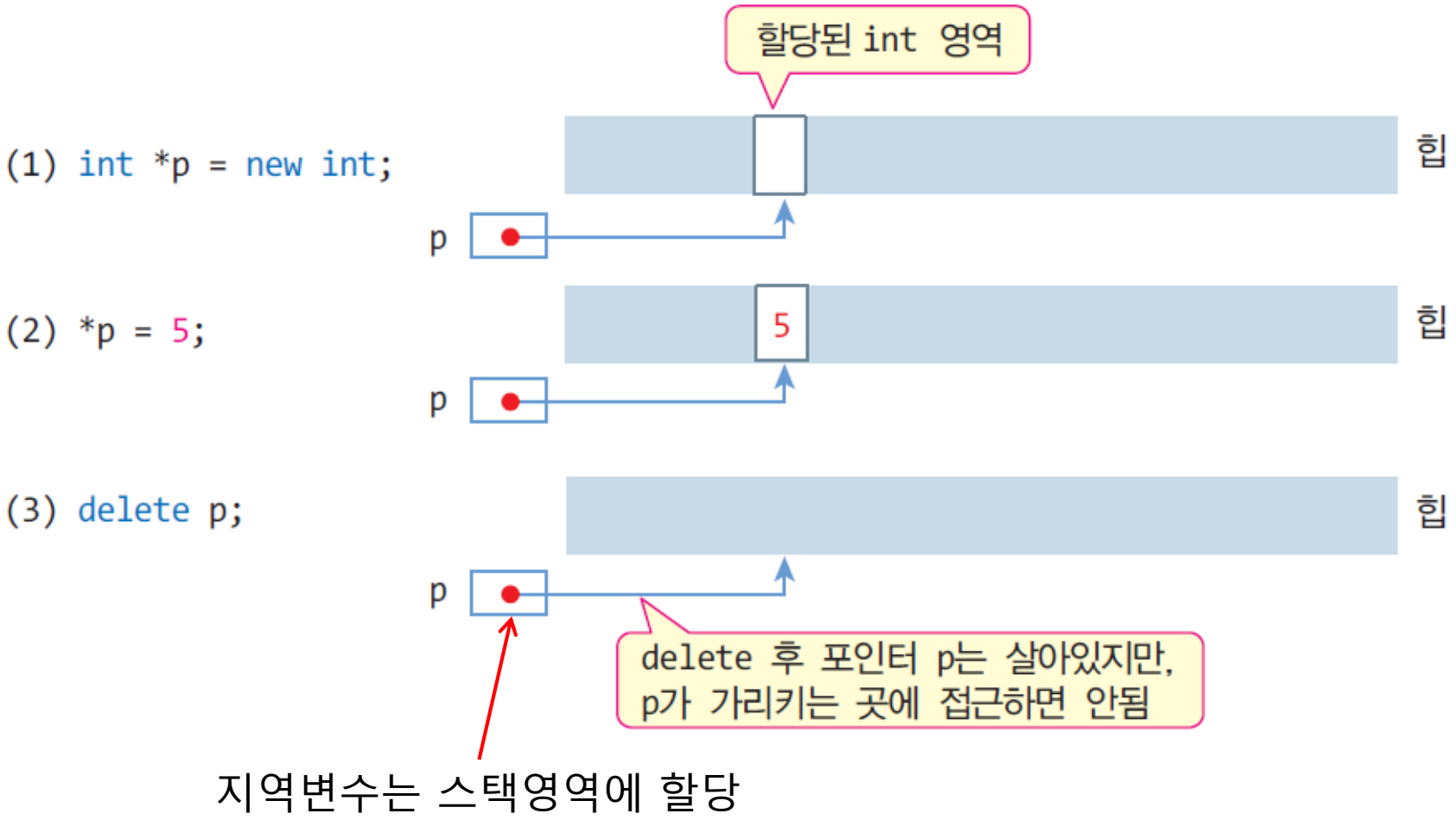
```
int* p = new int;
if (!p) {                                   // if(p == NULL)
      cout << "메모리 부족 " ;               // 에러메시지 출력 후 종료
      return -1;
}
```

예제 4-5 정수형 공간의 동적 할당 및 반환 예


```
#include <iostream>
using namespace std;
int main() {
    int* p = new int;                    // 메모리 동적할당
    if (!p) {
        cout << "메모리 부족";           // 에러메시지 출력
        return -1;                       // 비정상 종료시 -1리턴
    }
    *p = 5;                              // 할당 받은 공간에 5 대입
    int n = *p;
    cout << "*p = " << *p << '₩n';
    cout << "n = " << n << '₩n';

    delete p;                            // 메모리 반환
    return 0;
}
```

```
*p = 5
n = 5
```

예제 4-5 정수형 공간의 동적 할당 및 반환 예

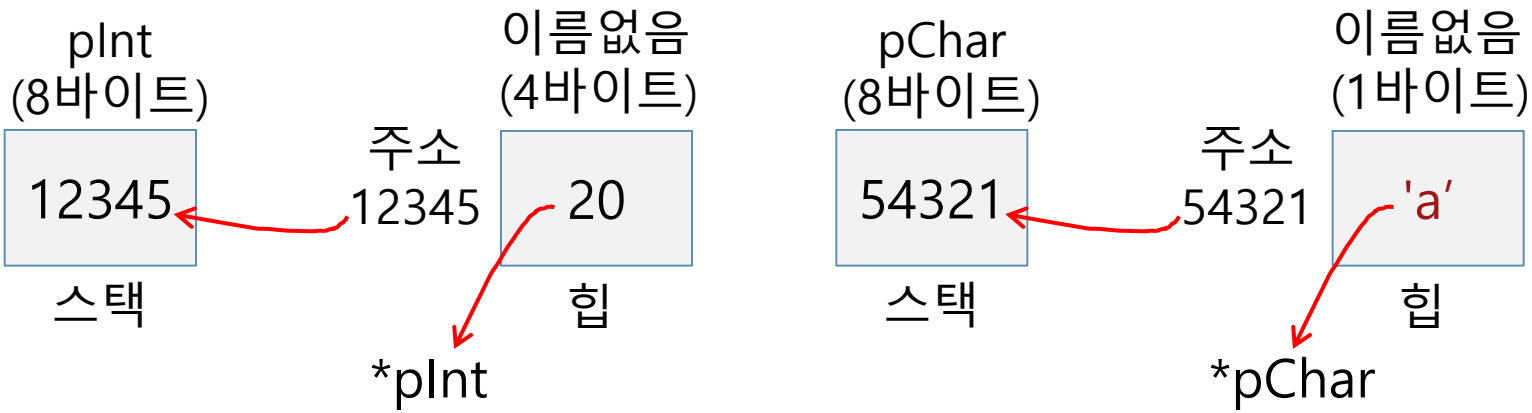


동적 할당 메모리 초기화

- 동적 할당 메모리 초기화 -> 객체 초기화와 유사

```
자료형* 포인터 = new 자료형(초깃값);
```

```
int* pInt = new int(20);          // 20으로 초기화된 int 타입 할당
char* pChar = new char('a');      // 'a'로 초기화된 char 타입 할당
```



delete 사용 시 주의 사항

- 적절치 못한 포인터를 delete하면 실행 시간 오류 발생
  - 동적으로 할당 받지 않는 메모리 반환 오류

```
int n;
int* p = &n;
delete p;           // 실행 시간 오류
          // 포인터 p가 가리키는 메모리는 동적으로 할당 받은 것이 아님
```

  - 동일한 메모리 두 번 반환 - 오류

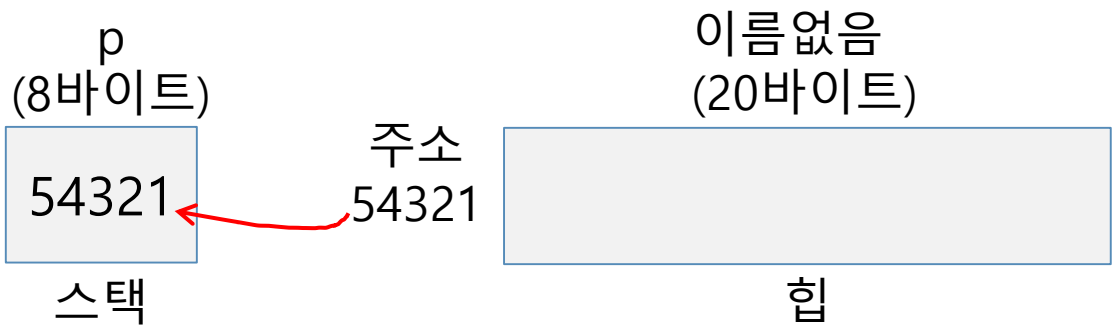
```
int* p = new int;
delete p;     //정상적인 메모리 반환
delete p;     //실행시간 오류. 이미 반환한 메모리를 중복 반환할 수 없음
```

배열의 동적 할당 및 반환

- new/delete 연산자의 사용 형식

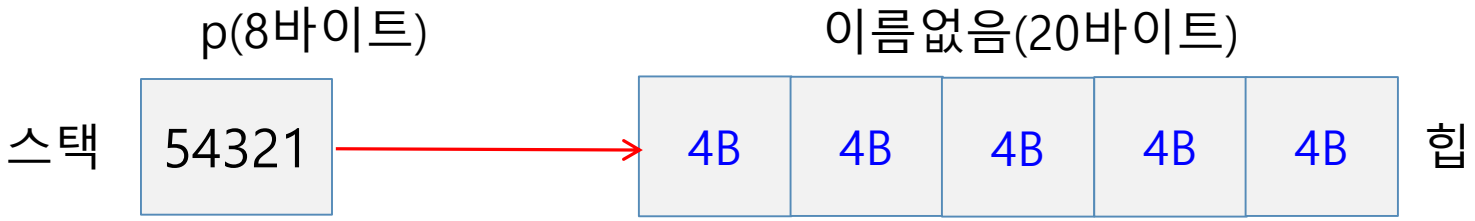
```
자료형* 포인터 = new 자료형[배열의 크기];
delete [ ] 포인터;
```

```
int* p = new int[5];
…

delete [ ] p;
```



배열의 동적 할당 및 반환

- 포인터를 이용한 배열의 동적 메모리 접근



주소	p+0	p+1	p+2	p+3	p+4
값	*(p+0)	*(p+1)	*(p+2)	*(p+3)	*(p+4)
값	p[0]	p[1]	p[2]	p[3]	p[4]

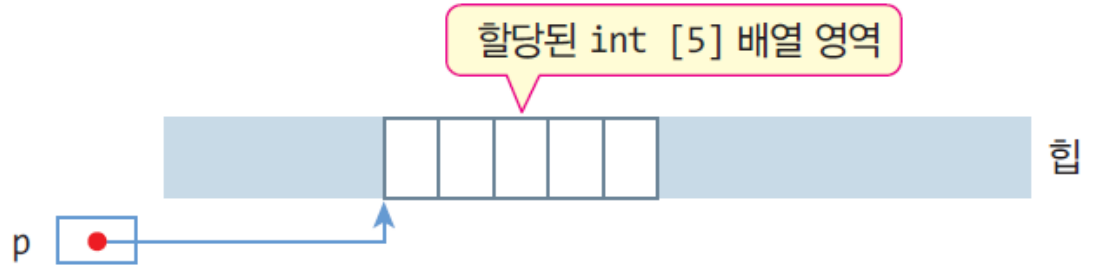
공식: p[i] <-> *(p+i)

예제 : 배열의 동적 할당

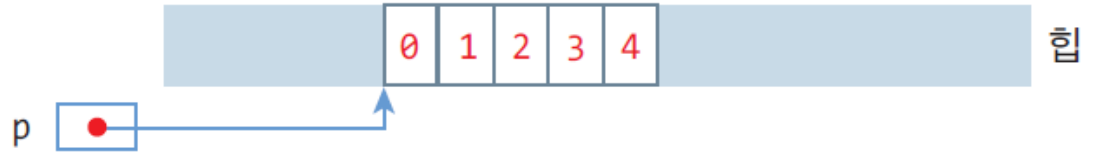

```
#include <iostream>
using namespace std;
int main() {
	int* p = new int[5];                    // 배열 메모리 할당
	if (!p) {
		cout << "메모리 부족";              // 에러메시지 출력
		return -1;
	}
	for (int i = 0; i < 5; i++) p[i] = i;   // *(p+i) = i;

	delete [ ] p;
	return 0;
}
```

예제 : 배열의 동적 할당

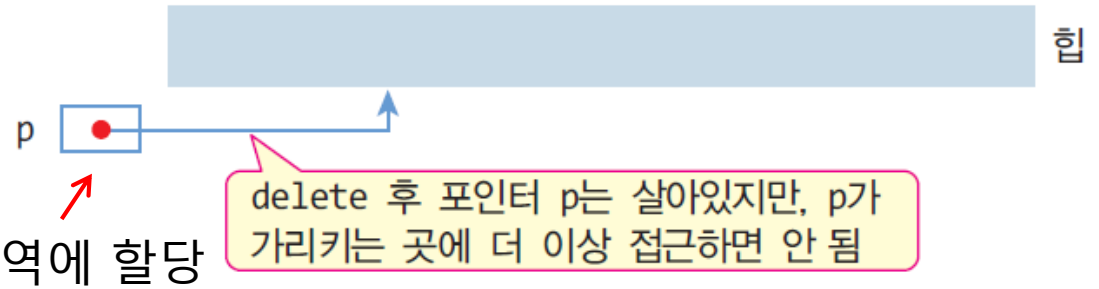

```
(1) int *p = new int [5];
```




```
(2) for(int i=0; i<5; i++)
        p[i] = i;
```




```
(3) delete [] p;
```



지역변수는 스택영역에 할당

예제 4-6 정수형 배열의 동적 할당 및 반환

- 사용자로부터 입력할 정수의 개수를 입력 받아 배열을 동적 할당 받고, 하나씩 정수를 입력 받은 후 합을 출력

```
#include <iostream>
using namespace std;
int main() {
	cout << "입력할 정수의 개수는?";
	int n;
	cin >> n;                                  // 정수의 개수 입력
	if (n <= 0) return -1;
	int* p = new int[n];                       // n 개의 정수 배열 동적 할당
	if (!p) {
		cout << "메모리를 할당할 수 없습니다.";
		return -1;
	}
```

예제 4-6 정수형 배열의 동적 할당 및 반환


```
    for (int i = 0; i < n; i++) {
        cout << i + 1 << "번째 정수: ";
        cin >> p[i];
    }

    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += p[i];
    cout << "평균 = " << sum / n << endl;

    delete [ ] p;                    // 배열 메모리 반환
}
```

```
입력할 정수의 개수는?4 <엔터>
1번째 정수: 4 <엔터>
2번째 정수: 20 <엔터>
3번째 정수: -5 <엔터>
4번째 정수: 9 <엔터>
평균 = 7
```

동적 할당 메모리 초기화 및 delete 시 유의 사항

- 동적 할당 시 배열의 초기화

```
int* pArray = new int[4]{ 1,2,3,4 };        // 정상
int* pArray = new int[10](20);              // 컴파일 오류
int* pArray = new int(20)[10];              // 컴파일 오류
```

- delete시 [ ] 생략
  - 컴파일 오류는 아니지만 비정상적인 반환

```
int* p = new int[10];
delete p;                   // 비정상 반환. delete [] p;로 하여야 함.

int* q = new int;
delete [] q;                // 비정상 반환. delete q;로 하여야 함.
```

실습과제1

- 다음 빈칸을 채우시오. 정적변수는 인터넷을 검색하여 조사해볼 것

변수의 종류	메모리영역	메모리 할당시점	메모리 해제시점
지역변수			
전역변수			
정적변수			
동적메모리			

실습과제2

- 먼저 저장할 정수의 개수를 입력 받아 동적 할당을 이용하여 메모리를 할당하고 아래 결과처럼 수행하는 프로그램을 작성하시오.(예제4-6과 유사함)

```
입력할 정수의 개수를 입력하세요: 6<엔터>
6개의 정수를 입력 하시오.
10<엔터>
20<엔터>
-3<엔터>
3<엔터>
-10<엔터>
-20<엔터>
평균값은 0입니다.
```

실습과제3

- 먼저 저장할 실수의 개수를 입력 받아 동적 할당을 이용하여 메모리를 할당하고 아래 결과처럼 수행하는 프로그램을 작성하시오.(예제4-6과 유사함)

입력할 실수의 개수를 입력하세요: 5<엔터>
5개의 실수를 입력 하시오.
10.5<엔터>
20.78<엔터>
-3.14<엔터>
3.14<엔터>
-8.74<엔터>
최대값은 20.78입니다.

실습과제4

- 먼저 저장할 문자열의 크기를 입력 받아 동적 할당을 이용하여 메모리를 할당하고 아래 결과처럼 수행하는 프로그램을 작성하시오. 동적 할당할 크기는 널문자를 저장할 공간까지 고려해야한다.(예제4-6과 유사함)

저장할 문자열의 크기를 입력하세요: ***11****<엔터>*
문자열을 입력하시오: ***I am a boy.****<엔터>*
입력한 문자열은 ***I am a boy.*** *입니다.*

과제제출방법

- 소스코드, 라인단위의 주석, 실행결과를 포함하는 pdf파일을 작성한 후 eclass 과제 게시판에 업로드, **반드시 하나의 pdf파일로 업로드할 것**
- 기한 : 과제 게시판에 마감시간 참조
- 실행결과를 캡쳐할 때 글자를 알아보기 쉽게 확대해서 캡쳐할 것.
- 소스코드의 첫 부분은 아래처럼 제목,날짜,작성자(학번,이름)를 작성할 것

```
// ************************************************
//    제   목 :  정수 4개의 평균을 구하는 프로그램
//    날   짜 :  2023년 9월10일
//    작성자 :  15010101 홍길동
// ************************************************

// 소스코드 작성
```