제8장 포인터와 동적 할당

이번 장에서 학습할 내용

- 포인터
  - 포인터의 개념, 간접참조 연산자
  - 포인터 연산, const, void
- 포인터와 배열
- 포인터와 함수
  - 값/참조/포인터에 의한 호출
- 동적 메모리 할당
  - 예제: 벡터 클래스
- 2차원 배열의 동적 할당
  - 예제: 행렬 클래스
- 객체의 동적 생성
- 복사 생성자, 대입 연산자, 소멸자
  - 얕은 복사/깊은 복사
  - 예제: 벡터 클래스 V2

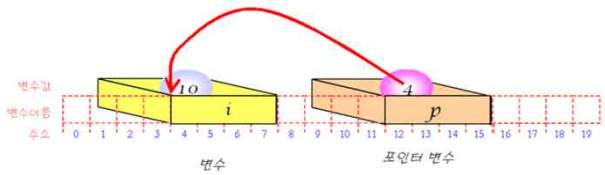
8.1 포인터

- *포인터(pointer):* 주소를 가지고 있는 변수

포인터의 선언

- 포인터: 변수의 주소를 가지고 있는 변수

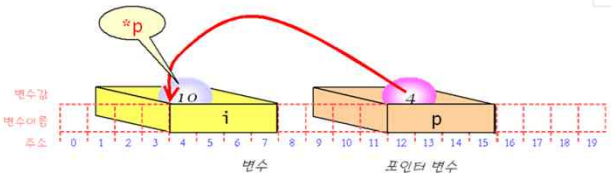
```
int i = 10;                // 정수형 변수 i 선언
int *p = &i;               // 변수 i의 주소가 포인터 p로 대입
```



간접 참조 연산자

- 간접 참조 연산자 *: 포인터가 가리키는 값을 가져오는 연산자

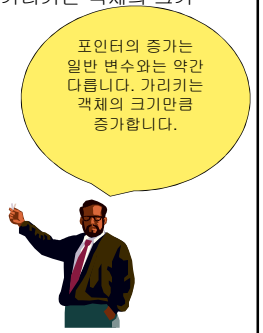
```
int i = 10;
int *p = &i;

cout << *p;        // 10이 출력된다.

*p = 20;
cout << *p;        // 20이 출력된다.
```



포인터 연산

- 가능한 연산: 증가, 감소, 덧셈, 뺄셈 연산
- 증가 연산의 경우 증가되는 값은 포인터가 가리키는 객체의 크기

포인터 타입	++연산후 증가되는값
char	1
short	2
int	4
float	4
double	8



증가 연산 예제

```
void main() {
   char *pc;
   int *pi;
   double *pd;

   pc = (char *)10000;
   pi = (int *)10000;
   pd = (double *)10000;
   cout << "증가 전 pc = " << (void *)pc << " pi = "
        << pi << " pd = " << pd << endl;
   pc++;
   pi++;
   pd++;
   cout << "증가 후 pc = " <<  (void *)pc << " pi = "
        << pi << " pd = " << pd << endl;
}
```

```
증가 전 pc = 00002710 pi = 00002710 pd = 00002710
증가 후 pc = 00002711 pi = 00002714 pd = 00002718
```


const와 포인터

- const 객체에 대한 포인터

```
const double *p;
double d = 1.23;
p = &d;          // 가능!
*p = 3.14;       // 컴파일 오류!
```

- 객체를 가리키는 const 포인터

```
double d = 1.23;
double *const p=&d;
*p = 3.14;       // 가능!
p = p + 1;       // 컴파일 오류! p는 변경될 수 없다.
```

const 포인터

- const int *p1;
- p1은 const int에 대한 포인터이다. 즉 p1이 가리키는 내용이 상수가 된다.
- *p1 = 100;(X)

- int * const p2;
- 이번에는 정수를 가리키는 p2가 상수라는 의미이다. 즉 p2의 내용이 변경될 수 없다.
- p2 = p1; (X)

void 포인터

- C++에서는 묵시적인 포인터 변환은 허용되지 않는다.

```
int *pi;
double *pd;
void* vp;

vp = pi;
pd = vp;
```

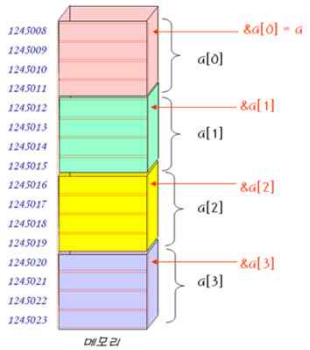
```
1>c:\sources\test\test\test.cpp(18) : error C2440: '=' : 'void *'에서 'double
*'(으)로 변환할 수 없습니다.
1> 'void*'에서 'void'가 아닌 포인터로 변환하려면 명시적 캐스트가 필요합니다.
1>빌드 로그가 "file://c:\sources\test\test\Debug\BuildLog.htm"에 저장되었습니다
.
1>test - 오류: 1개, 경고: 0개
========== 빌드: 성공 0, 실패 1, 최신 0, 생략 0 ==========
```


중간 점검 문제

1. 배열의 첫 번째 원소의 주소를 계산하는 2가지 방법을 설명하라.
2. 배열 a[]에서 *a의 의미는 무엇인가?
3. 배열의 이름에 다른 변수의 주소를 대입할 수 있는가?
4. 포인터를 이용하여 배열의 원소들을 참조할 수 있는가?
5. 포인터를 배열의 이름처럼 사용할 수 있는가?
6. 함수의 매개 변수로 전달된 배열을 변경하면 원본 배열이 변경되는가?
7. 배열을 전달받은 함수가 배열을 변경하지 못하게 하려면 어떻게 하여야 하는가?

8.2 포인터와 배열

- 배열 이름은 첫 번째 배열 원소의 주소
- 포인터를 사용하여서 배열 원소를 처리 가능



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

int main(void)
{
        const int STUDENTS = 5;
        int grade[STUDENTS] = { 10, 20, 30, 40, 50 };

        for(int *p=grade, *pend=grade+STUDENTS; p != pend; p++)
                cout << *p << " ";
        return 0;
}
```

```
10 20 30 40 50
```


8.3 포인터와 함수

- C++에서의 인수 전달 방법
  - 값에 의한 호출 (call-by-value)
    - 함수로 복사본이 전달된다.
    - 객체 전달 : 객체를 복사 ➔ 복사생성자
    - 객체의 주소 전달: 주소값 복사
  - 참조에 의한 호출 (call-by-reference)
    - 함수로 원본이 전달된다.
    - 객체의 별명 전달 : 복사되는 내용이 없음

swap() 함수 3가지

```
void swap(int x, int y)
{
    int tmp;

    tmp = x;
    x = y;
    y = tmp;
}
```

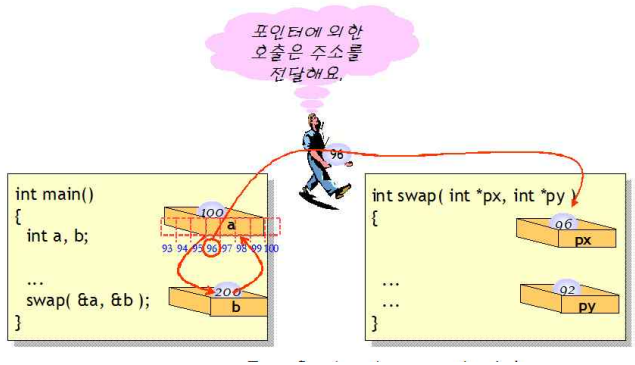
```
void swap(int &rx, int &ry)
{
    int tmp;

    tmp = rx;
    rx = ry;
    ry = tmp;
}
```

```
void swap(int* px, int* py)
{
    int tmp;

    tmp = *px;
    *px = *py;
    *py = tmp;
}
```

```
void main()
{
   int a = 100, b = 200;
   swap(a, b);
}
```

```
void main()
{
   int a = 100, b = 200;
   swap(&a, &b);
}
```


포인터를 이용한 호출



포인터 vs 참조자

- 일반적으로 참조자를 사용하는 편이 쉽다.
- 만약 참조하는 대상이 수시로 변경되는 경우에는 포인터를 사용
- NULL이 될 가능성이 있는 경우에도 포인터를 사용

```
int *p = new int;
if( p != NULL )
{
        int &ref = *p;
        ref = 100;
}
```


중간 점검 문제

1. 포인터와 참조자의 차이점을 설명하라.
2. 참조자보다 포인터를 사용하여야 하는 경우는?
3. 함수가 참조자를 반환할 수 있는가?
4. 함수에 매개 변수로 변수의 복사본이 전달되는 것을 _____라고 한다.
5. 함수에 매개 변수로 변수의 원본이 전달되는 것을 _____라고 한다.
6. 배열을 함수의 매개 변수로 지정하는 경우, 배열의 복사가 일어나는가?

8.4 동적 메모리 할당

- **동적 메모리**
  - 실행 도중에 동적으로 메모리를 할당받는 것
  - 사용이 끝나면 시스템에 메모리를 반납
  - 필요한 만큼만 할당을 받고 메모리를 매우 효율적으로 사용
  - new와 delete 키워드 사용

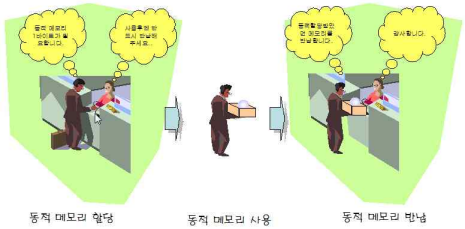


그림 3.5 동적 메모리 사용 단계

동적 메모리 할당

- 정적 메모리 할당
  - 메모리의 크기는 프로그램이 시작하기 전에 결정
  - 실행 도중에 크기를 변경할 수 없다.
  - 만약 처음에 결정된 크기보다 더 큰 입력이 들어온다면 처리하지 못할 것이고 더 작은 입력이 들어온다면 남은 메모리 공간은 낭비될 것이다.

```
int x;
int buffer[100];
char name[] = "data structure";
```

- **동적 메모리 할당**
  - 실행 도중에 메모리를 할당 받는 것
  - 필요한 만큼만 할당을 받고 반납함
  - 메모리를 매우 효율적으로 사용가능

동적 메모리 할당

- 정적메모리와 동적 메모리 할당 및 해제 코드

```
main()
{
    int x;                      // 정적으로 int 객체 할당
    int *py = new int;   // 동적으로 int 객체 할당
    ...
    delete py;           // 동적으로 int 객체 제거

    int arrA[20];               // 정적으로 배열 할당
    int *arrB = new int [20];   // 동적으로 배열 할당
    for(int i=0; i < 20;  i++) {
        arrA[i] = 0;            // 정적 메모리 사용
        arrB[i] = 0;            // 동적 메모리 사용
    }

    delete [] arrB;             // 동적으로 배열 제거
}                               // 정적 객체(x, arrA) 자동 해제
```


동적 메모리 할당 라이브러리

- new 연산자

```
data_type *pData = new data_type;
data_type *array = new data_type [size];
```

  - char *pc = new char[100] ; // char형 100개의 메모리할당
  - char *pi = new int; // int형 1개의 메모리할당
  - Book *pb = new Book; // Book객체 1개의 메모리할당

- delete 연산자

```
delete pData;
delete [] array;
```


동적 메모리 할당과 반납

```
int* pi = new int;              // 하나의 int형 공간할당
int* pia = new int[100];        // 크기가 100인 int형 동적배열할당
double* pd = new double;        // 하나의 double형 공간할당
double* pda = new double[100];  // 크기가 100인 double형 동적배열할당
```

```
delete pi;              // 동적할당int형공간반납
delete[] pia;           // 동적할당배열반납
delete pd;              // 동적할당double형공간반납
delete[] pda;           // 동적할당배열반납
```


중간 점검 문제

1. 프로그램의 실행 도중에 메모리를 할당받아서 사용하는 것을 _____이라고 한다.
2. 동적으로 메모리를 할당받을 때 사용하는 키워드는 _____이다.
3. 동적으로 할당된 메모리를 해제하는 키워드는 _____이다.

응용1: 벡터 클래스

```
class Vector
{
    int     dim;    // 벡터의 차원
    double* v;

private:
    void alloc( int d ) {
        if( d != dim ) {
            reset();
            dim = d;
            v = new double [dim];
        }
    }
    void reset() {
        if( dim > 0 ) {
            dim = 0;
            delete []v;
        }
    }
```

```
public:
    Vector(int d = 0) : dim(0), v(NULL) { alloc(d); }
    ~Vector() { reset(); }

    // 복소수 내용을 설정하는 함수 : inline
    void setRandom() {
        for( int i=0 ; i<dim ; i++ )
            v[i] = (rand() % 1000) / 10.0;
    }
    void read( char* msg = " 벡터 입력 =" ) {
        printf("  %s 차원 = ", msg);
        int d;
        scanf("%d", &d);

        alloc( d );
        printf("  벡터의 내용 입력(%d개) = ", d);
        for( int i=0 ; i<d ; i++ )
            scanf("%lf", v+i);
    }
    void print( char* msg = "벡터 = " ) {
        printf("%s\n\t[%d차원] : ", msg, dim);
        for( int i=0 ; i<dim ; i++ )
            printf(" %5.1f ", v[i]);

        printf("\n", dim);
    }
```



```
#include "Vector.h"

void main()
{
    Vector a(5), b(5), c;

    a.setRandom();
    b.setRandom();
//  a.read("VectorA");
//  b.read("VectorB");
    c.add(a,b);

    a.print ( "  A:");
    b.print ( "  B:");
    c.print ( "C.add(A,B):")
}
```

```
C:₩Windows₩system32₩cmd.exe
  A:
        [5차원] :   4.1   46.7   33.4   50.0   16.9
  B:
        [5차원] :  72.4   47.8   35.8   96.2   46.4
C.add(A,B):
        [5차원] :  76.5   94.5   69.2  146.2   63.3
계속하려면 아무 키나 누르십시오 . . .
```


8.5 2차원 배열의 동적 할당

- 이거 될까?

```
int **arr2D = new int [cols][rows];     // 잘못된 코드
...
delete [][] arr2D;                      // 잘못된 코드
```

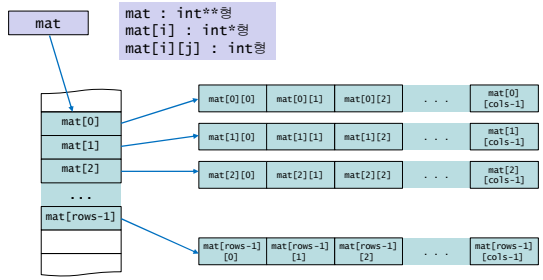
- 이상한 함수

```
int findMaxPixel( int a[][5], int h, int w );
```

- 2차원 배열 동적할당의 응용은?
  - 미로 찾기에서 임의의 크기의 맵
  - 영상처리에서 임의의 크기의 영상
  - 행렬에서 임의의 크기의 행렬 처리
  - 술에 취한 딱정벌레 문제 등

2차원 배열의 동적 할당

- 2차원 배열의 동적 할당: 보다 현명한 방법은?



```
int** alloc2DInt (int rows, int cols)
{
    if( rows <= 0 || cols <= 0 ) return NULL;
    int** mat = new int* [ rows ];
        for (int i=0 ; i<rows ; i++ )
            mat[i] = new int [cols];
    return mat;
}
```

```
void free2DInt ( int** mat, int rows, int cols=0)
{
    if( mat != NULL ) {
        for( int i=0 ; i<rows ; i++ )
                delete [] mat[i];
        delete [] mat;
    }
}
```


응용2: 행렬 클래스

```
class MatrixI {
    int     rows;   // 행의 개수
    int     cols;   // 열의 개수
    int**   mat;    // 행렬 데이터

public:
    MatrixI( int r=0, int c=0 ): mat(NULL) { alloc(r,c); }
    ~MatrixI( ) { reset(); }

    /* 행렬 원소의 접근 함수: get, set */
    int     get(int r, int c )          { return mat[r][c]; }
    void    set(int r, int c, int val ) { mat[r][c] = val; }

    /* 2차원 배열의 동적 해제 */
    void reset ( ) {
        if( mat != NULL ) {
            for( int i=0 ; i<rows ; i++ )
                delete [] mat[i];
            delete [] mat;
        }
        rows = cols = 0;
        mat = NULL;
    }
    /* 2차원 배열의 동적 할당 */
    void alloc ( int r, int c ) {
        reset();
        rows = r;
        cols = c;
        mat = new int* [ rows ];
        for (int i=0 ; i<rows ; i++ )
            mat[i] = new int [ cols ];
    }
```

```
    /* 행렬의 값을 모두 같은 값으로 설정 */
    void set ( int val = 0 ) {
        if( mat != NULL ) {
            for( int i=0 ; i<rows ; i++ )
            for( int j=0 ; j<cols ; j++ )
                mat[i][j] = val;
        }
    }
    /* 행렬의 값을 임의의 값(0~99)으로 설정 */
    void setRandom ( ) {
        if( mat != NULL ) {
            for( int i=0 ; i<rows ; i++ )
            for( int j=0 ; j<cols ; j++ )
                mat[i][j] = rand() % 100;
        }
    }
    /* 행렬의 모든 항목을 화면으로 출력 */
    void print ( char *str = "matrix" ) {
        printf("%s %d X %d\n", str, rows, cols);
        for( int i=0 ; i<rows ; i++ ) {
            printf("\t");
            for( int j=0 ; j<cols ; j++ ) {
                printf("%4d", mat[i][j] );
            }
            printf("\n");
        }
    }
};
```

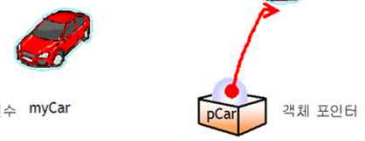


```
#include "Matrix.h"
void main()
{
    int r, c;
    printf(" 행과 열의 크기를 입력하세요: ");
    scanf("%d%d", &r, &c);

    MatrixI m(r,c);
    m.setRandom();
    m.print();
}
```

```
C:₩Windows₩system32₩cmd.exe
행과 열의 크기를 입력하세요: 4 8
matrix 4 X 8
          41  67  34   0  69  24  78  58
          62  64   5  45  81  27  61  91
          95  42  27  36  91   4   2  53
          92  82  21  16  18  95  47  26
계속하려면 아무 키나 누르십시오 . . .
```


8.6 객체의 동적 생성

- 객체도 동적으로 생성할 수 있다.
  - Car   myCar; // 정적 메모리 할당으로 객체 생성
  - Car *pCar = new Car(); // 동적 메모리 할당으로 객체 생성
- 객체 포인터를 통한 멤버 접근
  - pCar->speed = 100;
  - pCar->speedUp();



객체 변수 myCar 객체 포인터

예제

```
int main()
{
        Car myCar;

        myCar.print();

        pCar = new Car(0, 1, "blue");
        pCar->print();
        return 0;
}
```

객체 동적 생성

속도: 0 기어: 1 색상: white
속도: 0 기어: 1 색상: blue

중간 점검 문제

1. 클래스로부터 객체를 생성할 수 있는 방법을 열거하여 보라.

2. 객체 포인터로는 반드시 동적 생성된 객체만을 가리켜야 하는가?

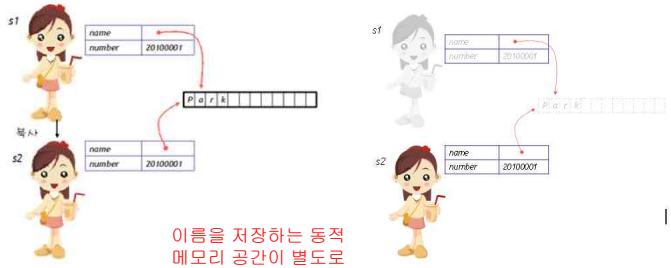
8.7 복사 생성자, 대입연산자, 소멸자

- 복사 생성자의 호출
  - 기존의 객체의 내용을 복사하여서 새로운 객체를 만드는 경우
  - 객체를 값으로 매개 변수로 전달하는 경우
  - 객체를 값으로 반환하는 경우
- 대입 연산자의 호출
  - 객체간의 대입 연산을 하는 경우
- 소멸자의 호출
  - 객체가 소멸되는 경우

- 컴파일러가 자동으로 제공함
- ➔ **반드시 구현해 주어야 하는 경우는?**

얕은 복사/깊은 복사

- 멤버의 값만 복사하면 안되는 경우가 발생한다.
- 얕은 복사(shallow copy) 문제



이름을 저장하는 동적 메모리 공간이 별도로 할당되지 않았음

예제

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

class Student {
        char *name; // 이름
        int number;
public:  Student(char *p, int n) {
                cout << "메모리 할당" << endl;
                name = new char[strlen(p)+1];
                strcpy(name, p);
                number = n;
        }
        ~Student() {
                cout << "메모리 소멸" << endl;
                delete [] name;
        }
};
int main()
{
        Student s1("Park", 20100001);
        Student s2(s1);    // 복사 생성자 호출
        return 0;
}
```

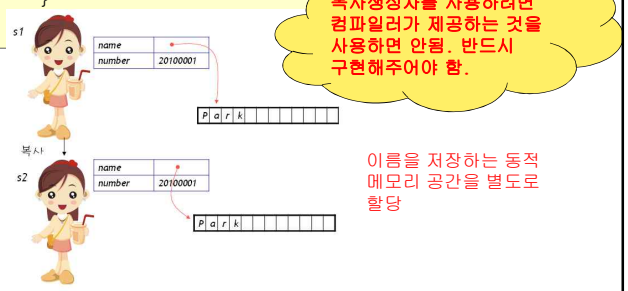
동적으로 할당한 멤버는 반드시 소멸자에서 메모리를 해제하여야 함

메모리의 소멸이 2번 호출되었음

```
메모리 할당
메모리 소멸
메모리 소멸
```


깊은 복사가 필요한 클래스

```
class Student {
        ....
        Student(const Student& s) {
                        cout << "메모리할당<< endl;
                name = new char[strlen(s.name)+1];
                strcpy(name, s.name);
                number = s.number;
        }
};
```



복사생성자를 사용하려면 컴파일러가 제공하는 것을 사용하면 안됨. 반드시 구현해주어야 함.

이름을 저장하는 동적 메모리 공간을 별도로 할당

그림 6.9 깊은 복사

대입연산자, 소멸자

- 대입 연산자도 마찬가지로 구현해주어야 한다.
- 소멸자에서 반드시 메모리를 해제하여야 한다.

중간 점검 문제

1. 복사 생성자는 언제 사용되는가?
2. 얕은 복사와 깊은 복사의 차이점은 무엇인가?

3. 복소수 클래스와 벡터 클래스의 차이는?

4. 다음과 같은 연산을 사용하기 위해서 벡터 클래스에서 해 주어야 하는 것은?

```
Vector a, b, c;
...
c.add(a,b);          // 소멸자추가, 복사생성자 추가
c = a.add(b);        // 대입연산자까지 추가
```