LAB #2

경희대학교
KYUNG HEE UNIVERSITY

실습 목표 – 포인터와 배열

1. 배열에 대한 이해를 바탕으로 배열과 포인터의 관계 이해
2. 매개 변수로의 배열의 활용
3. 포인터를 이용한 1,2차원 배열의 활용
4. 동적할당을 통한 포인터 배열의 활용

- 실습 문제 1 <Section 9-12>

목표 : 배열의 주소 이해 - 배열의 원소값 및 주소값 출력

- 아래 main( ) 함수코드를 활용하여 배열을 초기화 하고 배열의 원소와 주소값을 출력하여 그 의미를 이해하라.

```
Int main(){
     int Ary[5] = {10, 20, 30, 40, 50};            // 배열의 선언과 초기화

     cout << setw(20) << "Print Element";
     for( int i=0; i<5; i++ )
          cout << setw(10) << (코드 작성) ;        // 배열의 원소 출력(배열 첨자 연산자)
     cout << endl;

     cout <<  setw(20) << "Address of Element";
     for( int i=0; i<5; i++ )
          cout << setw(10) << (코드 작성);             // 배열의 원소 주소 출력(배열 첨자 연산자, 주소 연산자)

     cout <<  setw(30) << "Size of Memory";
     for( int i=0; i<5; i++ )
          cout << setw(10) << (코드 작성);             // 배열 각 원소의 메모리 사이즈 출력(sizeof() 함수)
     cout << endl;

     char *answer = "의미를 여기에 기록";
     cout << answer << endl;
     return 0;
}
```

output

```
      Print Element        10        20        30        40        50
    Address of Element  0040F7D8  0040F7DC  0040F7E0  0040F7E4  0040F7E8
     Size of Memory         4         4         4         4         4
배열 원소의 주소 공간의 크기 및 주소 간 간격에 대한 의미를 여기에 설명
```

• 실습 문제 2 <Section 9-12>

목표 : 배열명과 포인터 – 배열 이름의 역참조와 포인터로서의 배열의 이름 이해

- 실습 1번 문제에 아래 코드를 참고하여 '배열명'과 배열의 '첫번째 원소'의 주소를 출력하라.
- 정수형 포인터를 배열의 주소로 초기화 하여 포인터 값과 포인터가 가리키는 값을 출력하라.

```
// 실습 1의 코드 생략
cout << setw(10) << "Ary = " << (코드작성) << endl;      // '배열명'의 출력
cout << setw(10) << "&Ary[0] = " << (코드작성) << endl; // '첫 번째 원소'의 주소 출력

int* pAry = (코드작성);                                  // 포인터를 배열의 이름으로 초기화
cout << setw(10) << "pAry = " << (코드작성) << endl;     // 포인터로서의 배열 이름 출력
cout << setw(10) << "*pAry = " << (코드작성) << endl;    // 간접 연산자(*)를 통해 배열의 첫 번째 요소 접근
cout << setw(10) << "*Ary = " << (코드작성) << endl;     // 배열명에 간접 연산자(*)을 통한 역참조
cout << setw(10) << "pAry[0] = " << (코드작성) << endl; // 포인터를 이용한 배열 첨자 연산자([])
```

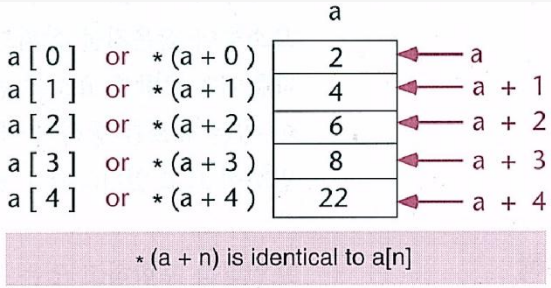
output

```
       Print Element         10         20         30         40         50
  Address of Element   0015F780   0015F784   0015F788   0015F78C   0015F790
     Ary = 0015F780
&Ary[0] = 0015F780
    pAry = 0015F780
   *pAry = 10
    *Ary = 10
pAry[0] = 10
```

- ## 실습 문제 3 <Section 9-13>

포인터의 산술 연산과 배열 – 오프셋 연산을 통한 배열 접근

- 실습 문제2에 아래 그림을 참조하여 오프셋 연산을 통해 배열의 원소와 각 원소의 주소를 출력하고 그 결과를 확인하라.




```
int Ary[5] = {10, 20, 30, 40, 50};
int* pAry = Ary;

// 포인터 산술 연산 표현을 활용한 출력
cout << setw(23) << "포인터 산술연산";
for( int k=0; k<5; k++ )
     cout << setw(9) << "pAry+" << k;
cout << endl;

cout <<  setw(23) << "포인터 산술연산 결과";
// (코드 작성) 포인터 산술연산 결과 출력
```

```
cout <<  setw(23) << "배열 포인터 역참조";
// (코드 작성)배열 포인터 역참조를 통한 배열 원소 출력

cout <<  setw(23) << "포인터에 배열 첨자 연산";
// (코드 작성)포인터를 이용한 배열 첨자 연산
```

output

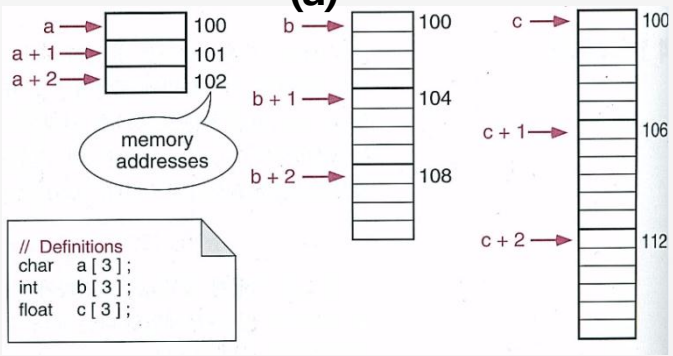
포인터 산술연산	pAry+0	pAry+1	pAry+2	pAry+3	pAry+4
포인터 산술연산 결과	003CFBC4	003CFBC8	003CFBCC	003CFBD0	003CFBD4
배열 포인터 역참조	10	20	30	40	50
포인터의 배열 첨자 연산	10	20	30	40	50

- 쉬어가는 코너 <Section 9-13>

타입에 따른 오프셋 산술 연산 결과

- 아래 그림(a)은 타입에 따라 오프셋 연산 결과가 달라짐을 보여주고 있다.(char 1byte, int 4byte, float는 6byte라 가정) 샘플 코드(b)의 디버깅을 통해 메모리 할당 모습을 출력해 보면 (c)와 같다. 타입에 따른 오프셋 연산 결과와 주소 증가(감소)를 살펴보자.

(a)



(b)

```
// 샘플 코드
int a[3];
short int b[3];
char c[3];
float d[3];
```

```
int* pa = a;
short int* pb = b;
char* pc = &c[1]; // 원소[1]의 주소로 초기화
float* pd = d;
```

(c)

pa = a

pb = b

pc = &c[1]

pd = d

Name	Value	Type
a	0x0038fd34	int [3]
&a[1]	0x0038fd38	int *
&a[2]	0x0038fd3c	int *
b	0x0038fd18	short [3]
&b[1]	0x0038fd1a	short *
&b[2]	0x0038fd1c	short *
c	0x0038fd00	char [3]
&c[1]	0x0038fd01	char *
&c[2]	0x0038fd02	char *
d	0x0038fce0	float [3]
&d[1]	0x0038fce4	float *
&d[2]	0x0038fce8	float *

Name	Value	Type
pa	0x0038fd34	int *
pa+1	0x0038fd38	int *
pa+2	0x0038fd3c	int *
pb	0x0038fd18	short *
pb+1	0x0038fd1a	short *
pb+2	0x0038fd1c	short *
pc-1	0x0038fd00	char *
pc	0x0038fd01	char *
pc+1	0x0038fd02	char *
pd	0x0038fce0	float *
pd+1	0x0038fce4	float *
pd+2	0x0038fce8	float *

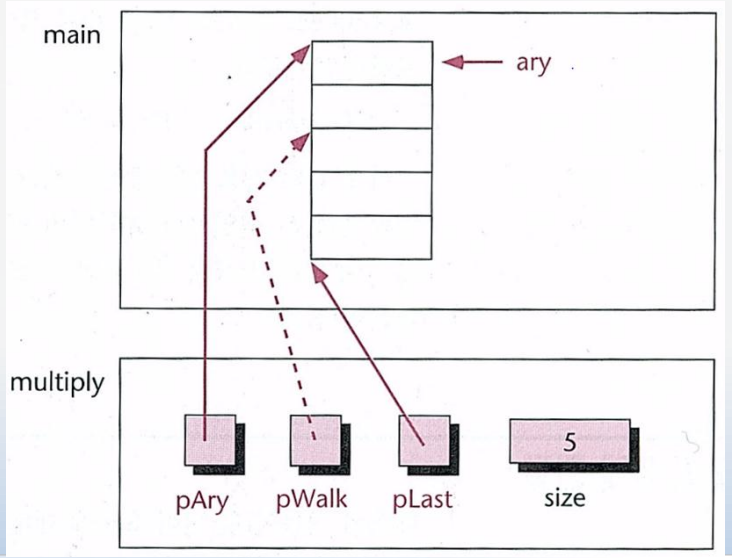
Name	Value	Type
a	0x0038fd34	int [3]
a+1	0x0038fd38	int [3]
a+2	0x0038fd3c	int [3]
b	0x0038fd18	short [3]
b+1	0x0038fd1a	short [3]
b+2	0x0038fd1c	short [3]
c	0x0038fd00	char [3]
c+1	0x0038fd01	char [3]
c+2	0x0038fd02	char [3]
d	0x0038fce0	float [3]
d+1	0x0038fce4	float [3]
d+2	0x0038fce8	float [3]

Float는 테스트 운영체제에서 4byte로 할당 되었음.

• 실습 문제 4(1) <Section 9-14>

목표 : 함수 매개 변수로서의 배열 – 매개 변수로서 배열의 전달 방법과 매개변수로서 배열 표기법의 실제 의미를 이해한다.

- 매개 변수로 배열(포인터 사용)을 전달받아 각 배열의 원소에 2를 곱하는 multiplyPtr( ) 함수 작성하라.
  - 배열의 접근은 아래 그림을 참고하여 pAry, pWalk, pLast, size를 이용하라.
  - pAry : 배열 시작 주소 포인터, pWalk : 배열 탐색 포인터, pLast : 배열 마지막 원소 주소 포인터
  - ex) pLast = pAry + size -1; // 포인터 오프셋 연산 활용
- main( ) 에서 multiplyPrt( )함수 호출 후 포인터를 이용하여 배열 원소의 출력 코드를 작성하라.
- checkSize( ) 함수를 통해 배열이 매개 변수로 전달되는 방법에 따라 매개변수 크기의 차이 여부를 확인하라.



- 실습 문제 4(2) <Section 9-14>

```
// function : 입력받은 배열의 각 원소에 2를 곱하는 함수
// input : 배열의 주소 (포인터 사용), 배열 길이
void multiplyPtr(int* pAry, int size);

// function : 배열을 입력 받아 sizeof()함수를 이용한 배열명, 포인터의 사이즈 출력
// input : 배열의 주소, 배열 길이
void checkSize(int* pAry, int Ary[], int size);

int main(){
     int Ary[ ] = { 10, 20, 30, 40, 50};
     int* pAry = Ary;
     int arySize = sizeof(Ary)/sizeof(Ary[0]);

     multiplyPtr(Ary, arySize);          // 배열의 각 원소에 2를 곱하는 함수 구현

     // (출력 코드 작성)포인터를 이용한 배열의 출력코드

     checkSize(pAry, Ary, arySize);
     cout << setw(35) << "main() : sizeof(배열명 Ary) = " << sizeof(Ary) << endl;
     char* answer = "Answer : 크기 차이 여부를 확인하고 그 이유를 여기에 작성";
     cout << answer << endl;
     return 0;
}
void multiplyPtr(int* pAry, int size){ (함수 구현) }
void checkSize(int* pAry, int Ary[], int size){
     cout <<  "checkSize() : sizeof(포인터 pAry) = " << sizeof(pAry) << endl;
     cout <<  "checkSize() : sizeof(배열명 Ary) = " << sizeof(Ary) << endl;
}
```

output

```
  20  40  60  80 100
checkSize() : sizeof(포인터 pAry) = 4
checkSize() : sizeof(배열명 Ary) = 4
     main() : sizeof(배열명 Ary) = 20
Answer : 크기 차이 여부를 확인하고 그 이유를 여기에 작성
```

• 실습 문제 5 <Section 9-15>

/ 메모리 할당 함수 – 동적 메모리 할당 방법에 대하여 이해하고 활용하자. **/**

- 배열의 길이를 입력 받아 동적 할당하고 배열의 원소를 피보나치 수열로 입력 받는 코드를 작성하여라.
  - 배열의 원소를 채우는 코드는 포인터 역참조(*) 방식을 활용하라.
  - 배열의 원소를 출력하는 코드는 배열 첨자 연산자([ ])를 활용하라.

```
int main(){
     int Dsize;
     cout << "Enter a size of the array : ";
     cin >> Dsize;
     int* Dary = new int [Dsize];     // (코드 작성) 동적 배열 할당

     // (코드 작성) 배열의 0,1번째 원소를 입력 받음
     // (코드 작성) 배열 채우기(포인터 간접 연산자 활용)

     // (코드 작성) 배열 출력하기(포인터에 배열 첨자 연산자 활용)

     delete [] Dary;                  // (코드 작성) 동적 배열 메모리 해제

     return 0;
}
```

output

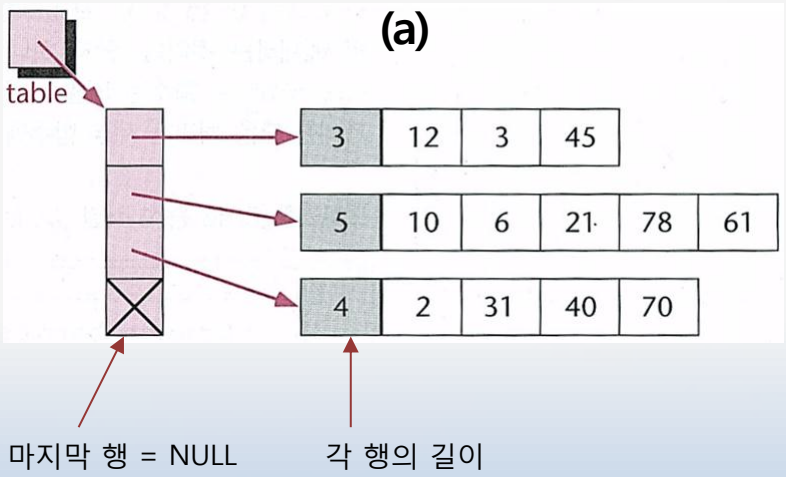
```
Enter a size of the array : 10
Enter a first element : 1
Enter a second element : 1
   1   1   2   3   5   8  13  21  34  55
계속하려면 아무 키나 누르십시오 . . .
```

• 실습 문제 6(1) <Section 9-17>

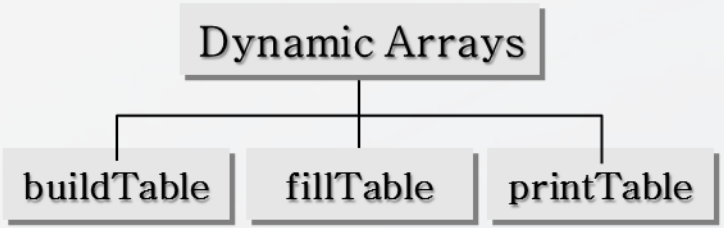
/ 목표 : 포인터를 이용한 2차원 동적 배열의 할당 방법과 그 활용법을 익힌다. **/**

- 불규칙한 크기의 배열을 저장할 수 있는 동적 테이블 생성 프로그램
  - 아래 그림(a)과 같은 불규칙한 배열을 갖는다.
  - 프로그램은 아래 그림(b)와 같은 구조는 갖는다.
  - buildTable( ) 함수는 동적 할당된 2차원 배열에 대한 이중포인터를 반환한다.
    - 각 행의 동적할당 시 행의 첫 번째 원소에 행의 길이 값의 넣는다.
  - fillTable( ) 함수는 포인터 배열을 받아 테이블을 채운다.
  - printTable( ) 함수는 포인터 배열을 받아 테이블의 내용을 출력한다.

(a)



(b)



- ## 실습 문제 6(2) <Section 9-17>

```
/** 각 함수에 대한 자세한 설명은 다음 '실습 문제 6(3), 6(4)' 참조 **/

// pre          -
// post    테이블에 대한 포인팅 포인터
// return 테이블에 대한 포인팅 포인터
int** buildTable(int rowNum, int columnSizeAry[]) ;

// pre    포인터의 배열
// post   채워진 테이블
void fillTable(int** table);

// pre    포인터의 배열
// post   테이블 원소 출력
void printTable (int** table);

int main()
{
       int nRows = 5;
       int columnSizeAry[] = { 3, 4, 5, 7, …];
       int** table = buildTable(nRows, columnSizeAry);    // Table에 대한 포인팅 포인터

       fillTable(table);        // Table 채우기
       printTable(table);            // Table 출력하기
       for(int i=0; i<columnSizeAry[i]+1; i++)
{
     delete [] table[i];
}
delete [] table;
       return 0;
}
```

output

```
Size of column in row 1: 2
Size of column in row 2: 5
Size of column in row 3: 1
For each row enter a eumber
 row 1 (2 integers) ========>11 12

 row 2 (5 integers) ========>21 22 23 24 25

 row 3 (1 integers) ========>31

 Print Array =>
        11          12
        21          22          23          24          25
        31
```

- ## 실습 문제 6(3) <Section 9-17>

```
// function : 2차원 배열에 대한 행과 열의 개수에 대한 정보를 입력 받아 포인터의 배열을 동적할당하여 포인터의 배열
              을 반환하는 함수
// input : 행의 개수, 각 행의 열 개수가 저장된 배열, 배열의 길이
// return : 불규칙한 열을 가진 2차원 포인터 배열
int** buildTable(int rowNum, int columnSizeAry[])
{
     (코드 작성);              // 2차원 동적 배열을 위한 포인터 선언

     table = (코드 작성);      // (행의 개수+1)만큼 배열 할당

     int row, col;
     for(row = 0; row < rowNum; row++)   // 각 행에 동적 배열 할당위한 반복문
     {
          int tmpColSize = columnSizeAry[row];
          cout << "Size of column in row " << row+1 << ": " << tmpColSize << endl;

          (코드 작성);              // 각 행에 (열의 개수+1)만큼 배열 할당
          (코드 작성);              // 각 행의 첫 원소에 열의 개수 저장
     }

     (코드 작성);                   // NULL 로 초기화(마지막 행 구분)
     return table;
}
```

- ## 실습 문제 6(4) <Section 9-17>

```
// function : 포인터의 배열을 입력받아 각 행의 원소를 채우는 함수
// input : 포인터의 배열
void fillTable(int** table){

     cout << "For each row enter a eumber";

     int row=0;
     while(table[row] != NULL){
          cout <<"₩n row " << row+1 <<" (" << table[row][0] << " integers) ========>";

          (코드 작성);         // 각 행의 원소값을 입력 받음
     }
     return;
}

//function : 포인터의 배열을 입력 받아 출력하는 함수(포인터 오프셋 연산 사용)
// input : 포인터의 배열
void printTable(int ** table)
{
     cout << "₩n Print Array => " << endl;

     (코드 작성);              // 포인터 오프셋연산을 이용한 배열 출력
}
```