실습 8 - 9주차

학과 : 전자공학과 학번 : 2023104322 이름 : 현시온

- 과제는 pdf로 변환하여 제출(과제 문서 첫 줄에 학과/학번/이름 포함)
- 과제는 순서대로 작성하며, 문제와 설명을 모두 포함(형식이 맞지 않으면 감점)
- 프로그램을 작성하는 문제는 소스코드와 실행 결과를 모두 text로 붙여넣기(그림으로 포함하지 말 것)하고 코드 설명 및 결과에 대한 설명을 포함해야 함
- 문의 사항은 이메일(nize@khu.ac.kr) 또는 오픈 카톡방을 이용

1. 아래 C++ 코드의 출력을 확인하고, 동작을 설명하라.

```
#include <iostream>
void print(int* p, const int size) {
   for (int i = 0; i < size; ++i) {
      std::cout << p[i] << ", ";
   }
   std::cout << std::endl;
}
int main() {
   int a1[3] = { 1, 2, 3 };
   int a2[] = { 1, 2, 3 };
   int a3[10] = { 1, 2, 3 };
   int a4[10];
   print(a1, 3);
   print(a2, 3);
   print(a3, 10);
   print(a4, 10);
}
```

동작 설명: 파라미터로 포인터 p와 불변 정수형 size를 받는 함수 print는 포인터 p의 대상의 엘리먼트를 차례대로 나열한다. Main 함수 동작으로는 정수형 엘리먼트를 가지는, size가 3이고 엘리먼트는 1,2,3인 a1, 엘리먼트가 1,2,3이어서 size도 3인 a2, size가 10이고 처음 인덱스부터 차례대로 1,2,3이 들어가고 그 외의 엘리먼트는 0인 a3, size가 10이고 모든 엘리먼트는 정의되지 않은 a4까지 총 4개의 배열을 선언한다. 그 다음 각각의 인수를 배열, size로 하여 차례대로 print 함수를 4번 호출한다. Print 함수의 역할이 포인터의 대상(여기서는 배열이 된다)의 엘리먼트를 차례대로 나열하는 것이므로, a1, a2, a3는 방금 선언한 것과 같이 동일한 결과물이 나오지만, a4의 경우는 알 수 없는 숫자가 나오는 것을 확인할 수 있는데, 이것은 엘리먼트 값이 초기화되지 않아 a4가 메모리에 할당될 때 임의로 배정된 쓰레기 값이라고 볼 수 있다.

출력 결과:

```
1, 2, 3,
1, 2, 3,
1, 2, 3, 0, 0, 0, 0, 0, 0, 0,
-858993460, -858993460, -858993460, -858993460, -858993460, -858993460, -858993460, -858993460, -858993460, -858993460,
```

2. 아래 C++ 코드의 출력을 확인하고, 동작을 설명하라.

```
#include <iostream>
int main() {
int a1[3] = { 10, 20, 30 };
double a2[3] = { 1, 2, 3 };
std::cout << a1 << ", " << a1 + 1 << std::endl;
std::cout << a2 << ", " << a2 + 1 << std::endl;
int *p1 = a1;
std::cout << p1 << ", " << p1 + 1 << std::endl;
std::cout << (*p1)++ << std::endl;
std::cout << *p1 << std::endl;
std::cout << *p1++ << std::endl;
std::cout << *p1 << std::endl;
}
```

동작 설명: 엘리먼트가 10, 20, 30 (정수형)인 배열 a1과 1, 2, 3 (실수형)인 배열 a2를 선언한다. (size는 전부 3) 그 다음 a1(2)과 a1(2)+1을 그대로 출력하는데, 이렇게 대괄호로 인덱스를 지정하여 엘리먼트를 의미하는 경우가 아닌 배열의 이름을 그대로 사용하는 경우, 그 배열의 첫번째 엘리먼트의 주소를 의미한다. 한편 배열의 이름에 숫자를 더한 것은 배열의 첫번째 엘리먼트의 주소에 의미하는 숫자와 그 숫자의 타입의 메모리 크기의 곱만큼 더한 주소를 의미하며, 배열의 엘리먼트의 주소 간의 차이는 엘리먼트의 타입의 메모리 크기만큼이기에, a1 + 1과 a2 + 1은 배열의 첫번째 엘리먼트 주소에 각각 4 (int), 8 (double)을 더한 주소가 되며, 이 둘은 모두 두번째 엘리먼트 주소를 의미한다. 한편 p1은 정수형을 가리키는 포인터인데, 배열 a1을 할당받았다. 이 경우에도 위에서와 같이 a1은 a1의 첫번째 엘리먼트의 주소를 의미한다. 그래서 p1와 p1+1의 출력은 a1과 a1+1의 출력과 동일한 결과가 나오게 된다. 한편 (*p1)++는 ()에 의해 순서가 뒤바뀌어 *, ++ 순이므로 p1의 대상의 후위 증가가 되어버린다. 그러므로 (*p1)++의 연산 결과는 p1의 대상인 10이지만 이후 p1의 대상은 1이 증가된 11이 될 것임을 알 수 있다. 그래서 이 다음 출력인 *p1, 즉 p1의 대상은 11이 된 것을 확인할 수 있다. 한편 *p1++는 그대로 우선 순위에 따라 ++, * 순이므로 p1++의 연산 결과는 p1이지만, 모든 연산이 끝나고 p1은 1이 더해져 원래 가리키던 엘리먼트의 다음 엘리먼트를 가리키게 될 것임을 알 수 있다. 그 다음 * 연산자가 작용하여 p1의 대상이 이 연산의 최종 결과이므로 11이 출력됨을 확인할 수 있다. 이제 다음 출력인 *p1은 후위 증가의 효력에 따라 다음 엘리먼트의 주소가 된 p1이기에 다음 엘리먼트인 20이 출력됨을 확인할 수 있다.

출력 결과:

```
000000018CAFF8A8, 000000018CAFF8AC
000000018CAFF8D8, 000000018CAFF8E0
000000018CAFF8A8, 000000018CAFF8AC
10
11
11
20
```

3. 아래 C++ 코드의 출력을 확인하고, 동작을 설명하라.

```
#include <iostream>
void print(int* p, const int size) {
   for (int i = 0; i < size; ++i) {
```

```
      std::cout << p[i] << ", ";
   }
   std::cout << std::endl;
}
int main() {
   int a1[3] = { 1, 2, 3 };
   for (int i = 0; i < 3; ++i)
      *(a1 + i) += 10;
   print(a1, 3);
   int* p = a1;
   for (int i = 0; i < 3; ++i)
      *p++ += 20;
   print(a1, 3);
   p = a1;
   for (int i = 0; i < 3; ++i)
      p[i] += 30;
   print(a1, 3);
}
```

동작 설명: 파라미터로 포인터 p와 불변 정수형 size를 받는 함수 print는 포인터 p의 대상의 엘리먼트를 차례대로 나열한다. Main 함수의 작동은 먼저 엘리먼트가 1,2,3에 size가 3인 배열 a1을 선언하고, i = 0,1,2로 하여 a1+i의 대상에 10을 더하는 과정을 반복한다. 위에 문제에서 설명한 것과 같이 a1+i는 인덱스 i에 위치한 엘리먼트의 주소를 의미하기에, 여기서는 a1의 인덱스 0,1,2, 즉 모든 엘리먼트에 10을 더하게 된다. 이후 그 결과를 print 함수로 출력한다. 그 다음엔 정수형을 가리키는 포인터 p를 선언하고 a1, 즉 a1의 첫번째 엘리먼트의 주소를 할당한다. 그 다음 i = 0,1,2로 하여 *p++에 20을 더하는 과정을 반복한다. *p++ += 20;의 연산은 p의 대상인 엘리먼트에 20을 더하고 p의 주소에 1을 더하여 p의 주소가 그 다음 엘리먼트의 주소가 되도록 하는 것이고 첫번째 p의 대상은 a1의 첫번째 엘리먼트이므로, a1의 모든 엘리먼트에 20이 더해진다. 이후 그 결과를 print 함수로 출력한다. 그 다음엔 변화한 p에 다시 a1의 첫번째 엘리먼트의 주소를 할당하고, i = 0,1,2로 하여 p[i]에 30을 더하는 과정을 반복한다. 이 경우 p[i]는 p가 가리키는 배열인 a1의 인덱스 i에 위치한 엘리먼트를 의미하게 되어, 결국 모든 a1의 엘리먼트에 30을 더하게 된다. 이후 그 결과를 print 함수로 출력한다.

출력 결과:

```
11, 12, 13,
31, 32, 33,
61, 62, 63,
```

4. 아래 C++ 코드의 출력을 확인하고, 동작을 설명하라.

```
#include <iostream>
void print(int* p, const int size) {
   for (int i = 0; i < size; ++i) {
      std::cout << p[i] << ", ";
   }
   std::cout << std::endl;
```

```
}
void print(int* begin, int* end) {
   for (int* elem = begin; elem != end; elem++) {
      std::cout << *elem << ", ";
   }
   std::cout << std::endl;
}
int main() {
   int a[3] = { 1, 2, 3 };
   print(a, 3);
   print(&a[0], &a[2] + 1);
   print(a, a + 3);
}
```

동작 설명: 파라미터로 정수형 포인터 p와 불변 정수형 size를 받는 함수 print는 포인터 p의 대상의 엘리먼트를 차례대로 나열한다. 그리고 오버로드지만 파라미터로 정수형 포인터 begin과 end를 받는 함수 print는 정수형 포인터 elem이 end가 아닐 때까지만 elem에 1씩 더하고 elem의 대상을 출력한다. 이 경우 begin과 end가 같은 배열의 다른 인덱스의 엘리먼트를 가리킨다면 그 사이의 모든 엘리먼트를 반복 과정마다 elem으로 받아 출력될 것임을 예상할 수 있다. Main 함수의 작동은 엘리먼트가 1,2,3인 배열 a를 선언하고 배열 a와 정수 3을 인수로 하여 print를 호출하는데, 이 경우 파라미터와 대응하는 적절한 함수는 첫번째 print이기에 a의 엘리먼트들이 나열된다. 한편 a의 인덱스 0의 주소와 인덱스 2의 주소에 1을 더한 결과, 즉 존재하지 않는 인덱스 3의 주소를 인수로 하여 print를 호출하면 이 경우는 두번째 print에 해당한다. 즉 인덱스 0이상 3미만의 엘리먼트, 즉 모든 a의 엘리먼트를 출력하게 된다. 그리고 a와 a+3을 인수로 하여 함수를 호출하는 경우, 그냥 a는 a의 인덱스 0의 주소를 의미하고 a+3은 존재하지 않는 a의 인덱스 3의 주소를 의미하므로 두번째 print 함수를 호출하게 되고, 똑같이 a의 모든 엘리먼트를 출력한다.

출력 결과:

```
1, 2, 3,
1, 2, 3,
1, 2, 3,
```

5. 아래 C++ 코드의 출력을 확인하고, 동작을 설명하라.

```
#include <iostream>
void print(double(*p)[3], const int size) {
   for (int row = 0; row < size; row++) {
      for (int col = 0; col < 3; col++)
         std::cout << p[row][col] << ", ";
      std::cout << std::endl;
   }
}
int main() {
   double a[2][3] = { {1, 2}, {4, 5, 6} };
   print(a, 2);
```

```
}
```

동작 설명: 엘리먼트의 타입이 실수형인 size가 3인 배열을 엘리먼트로 하는 배열을 가리키는 포인터 p와 불변 정수형 size를 파라미터로 하는 print 함수는 정수 row가 0부터 size보다 작을 때까지, 그리고 정수 col이 0부터 3보다 작을 때까지 p[row][col]을 나열한다. 이는 p가 가리키는 대상의 인덱스 row에 위치한 엘리먼트인 배열의, 인덱스 col에 위치한 엘리먼트를 의미한다. 메인 함수의 작동은 먼저 엘리먼트가 size가 3인 실수형 엘리먼트를 가지는 배열인, size가 2인 배열 a에 엘리먼트로 {1,2}, {4,5,6}을 할당한다. 그런 다음 a와 정수 2를 인수로 하여 print 함수를 호출한다.

출력 결과:

```
1, 2, 0,
4, 5, 6,
```

6. 아래 C++ 코드의 출력을 확인하고, 동작을 설명하라.

```
#include <iostream>
void print(double(*p)[3], const int size) {
   for (int row = 0; row < size; row++) {
      for (int col = 0; col < 3; col++)
         std::cout << p[row][col] << ", ";
      std::cout << std::endl;
   }
   std::cout << std::endl;
}
int main() {
   double a[2][3] = { {1, 2}, {4, 5, 6} };

   double(*p1)[3] = a;
   print(p1, 2);
   double(*p2)[3] = &a[0];
   print(p2, 2);
   double* p3 = a[0];
   *p3 = 10;
   print(p2, 2);
   double* p4 = &a[0][1];
   *p4 = 20;
   print(p2, 2);
}
```

동작 설명: 동작 설명: 엘리먼트의 타입이 실수형인 size가 3인 배열을 엘리먼트로 하는 배열을 가리키는 포인터 p와 불변 정수형 size를 파라미터로 하는 print 함수는 정수 row가 0부터 size보다 작을 때까지, 그리고 정수 col이 0부터 3보다 작을 때까지 p[row][col]을 나열한다. 이는 p가 가리키는 대상의 인덱스 row에 위치한 엘리먼트인 배열의, 인덱스 col에 위치한 엘리먼트를 의미한다. 메인 함수의 작동은 먼저 엘리먼트가 size가 3인 실수형 엘리먼트를 가지는 배열인, size가 2인 배열 a에 엘리먼트로 {1,2}, {4,5,6}을 할당한다. 여기까지는 바로 위 문제와 다를 것이 없다. 그 다음 엘리먼트의 타입이 실수형인 size가 3인 배열을 엘리먼트로 하는 배열을 가리키는 포인터 p1에 a를 할당하고 p와 정수 2를 인수로 하여 print 함수를 호출

한다. 이것은 a와 정수 2가 인수인 함수 호출의 결과와 다름없다. 다음은 엘리먼트의 타입이 실수형인 size가 3인 배열을 엘리먼트로 하는 배열을 가리키는 포인터 p2에 a의 첫번째 엘리먼트의 주소를 할당한다. 이는 a를 할당한 것과 동일한 맥락이므로 역시 동일한 결과를 출력한다. 다음은 실수형 포인터 p3에 a의 첫번째 엘리먼트를 할당한다고 문맥상으로는 볼 수 있지만, 실수형 포인터에 단지 배열 이름만을 할당할 경우 배열의 첫번째 엘리먼트의 주소가 할당되었던 것을 생각해본다면 실수형 포인터 p3에는 a의 첫번째 엘리먼트인 배열의 첫번째 엘리먼트의 주소가 할당될 것이라는 것을 예측할 수 있다. 즉 p3가 가리키는 대상은 a[0][0]이고, 이 다음 문장에서 10을 할당하므로 {1, 2}에서 1이 10으로 바뀐다. 다음은 실수형 포인터 p4에 a의 인덱스 0에 위치한 배열의 인덱스 1에 위치한 엘리먼트의 주소를 할당한다. 즉 {10, 2}에서 2에 해당하는 a[0][1]의 주소를 의미한다. 그리고 그 대상에 20을 할당하므로 {10, 20}으로 변화한다.

출력 결과:

```
1, 2, 0,
4, 5, 6,

1, 2, 0,
4, 5, 6,

10, 2, 0,
4, 5, 6,

10, 20, 0,
4, 5, 6,
```

7. 아래 C++ 코드의 출력을 확인하고, 동작을 설명하라.

```
#include <iostream>
#include <cstring>
// 아래 3줄은 설명이 필요 없음, visual studio에서 strcpy를 사용하기 위해 설정
#ifdef _MSC_VER
#pragma warning(disable : 4996)
#endif
int main() {
   char str[100] = "C++ Programming";

   std::cout << str << std::endl;
   str[0] = '\0';
   std::cout << str << std::endl;
   strcpy(str, "Object oriented programming");
   std::cout << str << ", " << strlen(str) << std::endl;
}
```

동작 설명: 문자형 엘리먼트를 가지고 크기가 100인 문자열 str에 C++ Programming 문자열을 할당한다. 이 경우 문자 하나하나가 차례대로 문자열의 엘리먼트가 된다. 그런 다음 문자열의 이름인 str을 그대로 출력한다면 실수형 배열의 경우에서 첫번째 엘리먼트의 주소가 나왔던 것과는 다르게 str에 초기화된 문자를 차례대로 나열해서 문자열을 출력하는 것을 볼 수 있다.

그 다음 str의 인덱스 0에 문자 \0, 즉 null을 할당하고, str을 다시 출력한다. 하지만 문자열 출력 중 null을 만나면 출력이 멈추기 때문에 아무것도 출력되지 않는다. 다음은 strcpy 함수를 통해 문자열 str에 Object oriented programming을 복사한다. 그러므로 str의 출력은 Object oriented programming이고, 문자열의 길이를 의미하는 strlen 함수에 의해 str의 문자 수인 27도 출력된다.

출력 결과:

```
C++ Programming


Object oriented programming, 27
```

8. 아래 C++ 코드의 출력을 확인하고, 동작을 설명하라.

```
#include <iostream>
void Fn(char* str1, const char* str2) {
   while (*str2) {
      *str1++ = *str2++;
   }
   *str1 = '\0';
}
int main() {
   char str[100];


   Fn(str, "Object oriented programming");
   std::cout << str << std::endl;
}
```

동작 설명: 문자형 포인터 strl과 불변 문자형 포인터 str2를 파라미터로 하는 함수 Fn은 가리키고 있는 str2의 인덱스에 엘리먼트가 존재하는 한, strl의 대상의 엘리먼트에 str2의 대상의 엘리먼트를 할당하고 두 포인터가 가리키는 인덱스를 옆으로 옮기는 과정을 계속 반복한다. 그러고 나선 strl의 대상의 마지막 엘리먼트는 null을 할당한다. 즉 함수 Fn의 역할은 복사이다. 메인 함수의 작동은 먼저 크기가 100인 문자열 str을 선언하고, str과 문자열 Object oriented programming을 인수로 하여 Fn을 호출한다. 그러고나서 str을 출력하면 Object oriented programming가 복사되었으니 정상적으로 출력됨을 알 수 있다.

출력 결과:

```
Object oriented programming
```

9. 아래 C++ 코드의 출력을 확인하고, 동작을 설명하라.

```
#include <iostream>
int main() {
   int data[10];
   for (int i = 0; i < 10; i++)
      data[i] = i;


   int* p1, * p2;
```

```
   p1 = new int[10];
   for (int i = 0; i < 10; i++)
      p1[i] = i*i;
   for (int i = 0; i < 10; i++)
      std::cout << data[i] << ", ";
   std::cout << std::endl;
   delete[] p1;
   p2 = data;
   for (int i = 0; i < 10; i++)
      p2[i] = i * i;
   for (int i = 0; i < 10; i++)
      std::cout << data[i] << ", ";
   std::cout << std::endl;
}
```

동작 설명: 메인 함수의 동작은 먼저 크기가 10인 정수형 배열 data를 선언하고, 각각의 인덱스에 0~9까지의 정수를 할당한다. 그런 다음 정수형 포인터 p1에 정수형 엘리먼트를 10개 저장할 수 있는 주소를 할당하고 각 엘리먼트는 p1이 가리키는 배열의 인덱스의 제곱으로 초기화한다. 그리고 data의 모든 엘리먼트를 나열하여 출력한다. 그런 다음 p1에 할당된 메모리를 해제하고 (p1을 nullptr로 만들지는 않는다. 더 이상 주소가 메모리를 할당 받지 않지만, 포인터는 그 주소를 가리키고 있는 것이다.) p2에 배열 data의 주소를 할당한다. 이후 p2가 가리키는 배열인 data의 엘리먼트에 인덱스의 제곱을 할당하고, data의 모든 엘리먼트를 출력한다.

출력 결과:

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
0, 1, 4, 9, 16, 25, 36, 49, 64, 81,
```

10. 아래 C++ 코드에서, 주석과 같이 출력되도록 F 함수를 정의하라(요소가 0보다 크면 square root를 저장).

```
#include <iostream>
#include <iomanip>
#include <cmath>
int main() {
double a[4][3] = { { -1, 2, 3 }, { 13 }, { 1.5, -12, 2.5 }, { 9, 4, -8 } };
F(a, 4);
for (int row = 0; row < 4; row++) {
for (int col = 0; col < 3; col++)
std::cout << std::setw(10) << a[row][col] << ", ";
std::cout << std::endl;
}
std::cout << std::endl;
}
```

```
// 실행 결과
//        -1,    1.41421,    1.73205,
//   3.60555,          0,          0,
//   1.22474,        -12,    1.58114,
//         3,          2,         -8,
```

```
#include <iostream>
#include <iomanip>
#include <cmath>
void F(double(*p)[3], const int size) {
        for (int row = 0; row < size; row++) {
                for (int col = 0; col < 3; col++)
                        if (p[row][col] > 0) {
                                p[row][col] = sqrt(p[row][col]);
                        }
                std::cout << std::endl;
        }
        std::cout << std::endl;
}
int main() {
        double a[4][3] = { { -1, 2, 3 }, { 13 }, { 1.5, -12, 2.5 }, { 9, 4, -8 } };
        F(a, 4);
        for (int row = 0; row < 4; row++) {
                for (int col = 0; col < 3; col++)
                        std::cout << std::setw(10) << a[row][col] << ", ";
                std::cout << std::endl;
        }
        std::cout << std::endl;
}
// 실행 결과
//        -1,    1.41421,    1.73205,
//   3.60555,          0,          0,
//   1.22474,        -12,    1.58114,
//         3,          2,         -8,
```

코드 설명: 인수로 받는 2차원 배열 a에 직접적인 변화를 주어야 하므로 크기가 3인 실수형 배열을 엘리먼트로 하는 배열의 포인터 p를 파라미터로 채택한다. 어짜피 a의 변화를 주는 것이 핵심이므로 아무것도 변환한 필요는 없으니 함수의 반환 타입은 void이다. 이후 대략적인 메커니즘은 5번 문제에서 2차원 배열을 출력할 때 사용한 print 함수와 비슷하게 하면서, 단지 반복문 내부의 내용을 p[row][col]을 출력하는 것에서 p[row][col]가 0보다 클 경우 p[row][col]에 루트 p[row][col]을 할당하는 것으로 바꾸었다. 또한 출력 내용은 main 함수 영역으로 옮겨간 것을 알 수 있다.