스스로 퀴즈 해답

CHAPTER 2

1 프로그램이 올바로 실행되면 운영체제에 리턴되는 값은 무엇인가?

Ⓐ -1 Ⓑ 1 **Ⓒ 0**
Ⓓ 프로그램은 값을 리턴하지 않는다.

2 C++ 프로그램이라면 반드시 포함해야 하는 한 가지 함수는 무엇인가?

Ⓐ start() Ⓑ system()
Ⓒ main() Ⓓ program()

3 코드 블록의 시작과 끝을 알리는 기호는 무엇인가?

Ⓐ { }
Ⓑ –>와 <–
Ⓒ BEGIN과 END
Ⓓ (와)

4 C++ 코드에서 각 행이 끝날 때 사용하는 기호는 무엇인가?

Ⓐ . **Ⓑ ;** Ⓒ : Ⓓ '

5 다음 중에서 올바른 주석은 무엇인가?

Ⓐ */ 주석 */
Ⓑ ** 주석 **
Ⓒ /* 주석 */
Ⓓ { 주석 }

6 cout에 접근하려면 반드시 사용해야 하는 헤더 파일은 무엇인가?

Ⓐ stream
Ⓑ 없다. 디폴트로 그냥 접근할 수 있다.
Ⓒ iostream
Ⓓ using namespace std;

CHAPTER 3

1 3.1415와 같은 수를 저장하려면 어떤 변수 타입을 사용해야 하는가?

Ⓐ int Ⓑ char
Ⓒ double Ⓓ string

2 두 변수를 비교하는 연산자는 어느 것인가?

Ⓐ := Ⓑ = Ⓒ equal **Ⓓ ==**

3 문자열 데이터 타입에는 어떻게 접근해야 하는가?

Ⓐ 언어 자체에 빌드되었으므로, 따로 할 일이 없다.
Ⓑ 문자열은 I/O에 사용되기 때문에 iostream 헤더 파일을 포함해야 한다.
Ⓒ string 헤더 파일을 포함해야 한다.
Ⓓ C++에서는 문자열을 지원하지 않는다.

4 다음 중에서 올바르지 않은 변수 타입은 무엇인가?

Ⓐ double **Ⓑ real** Ⓒ int Ⓓ char

5 사용자에게서 행 전체를 읽어오려면 어떻게 해야 하는가?

Ⓐ cin>>을 사용한다.
Ⓑ readline을 사용한다.
Ⓒ getline을 사용한다.
Ⓓ 쉽게 할 수 있는 방법이 없다.

6 C++에서 cout << 1234/2000 수식의 결과는 화면에 어떻게 나타나는가?

Ⓐ 0
Ⓑ .617
Ⓒ .617의 근사치. 다만, 부동소수점수로 정확하게 저장될 수는 없다.
Ⓓ 어떤 결과가 출력될지 단정 지을 수 없다.

7 **C++에 정수 타입이 있는데도 char 타입이 필요한 이유는 무엇인가?**

Ⓐ 문자와 정수는 완벽하게 다른 데이터이기 때문이다. 말 그대로 글자와 수는 다르다.
Ⓑ C와의 호환성을 염두에 둔 결과다.
Ⓒ 실제로 char는 수로 저장되지만, 수가 아닌 문자로 다루어야 쉽게 읽어 들이고 출력할 수 있어서다.
Ⓓ 프로그램을 세계화하려면 한국어나 중국어를 처리하기 위해 나타낼 수 있는 문자가 많아야 하기 때문이다.

CHAPTER 4

1 **다음 중에서 참인 것은 무엇인가?**

Ⓐ 1 Ⓑ 66 Ⓒ .1 Ⓓ −1
Ⓔ 전부 참이다

2 **다음 중에서 부울 and에 해당하는 부울 연산자는 무엇인가?**

Ⓐ & **Ⓑ &&** Ⓒ | Ⓓ |&

3 **!(true && ! (false || true))의 결과는 무엇인가?**

Ⓐ true Ⓑ false

4 **다음 중에서 if문의 올바른 문법은 무엇인가?**

Ⓐ if 수식 Ⓑ if { 수식
Ⓒ if (수식) Ⓓ 수식 if

CHAPTER 5

1 **int x; for(x=0; x<10; x++) {}라는 코드를 실행하면 x의 최종 값은 어떻게 되는가?**

Ⓐ 10 Ⓑ 9 Ⓒ 0 Ⓓ 1

2 **while(x<100) 뒤에 이어지는 코드 블록은 언제 실행되는가?**

Ⓐ x가 100보다 작을 때
Ⓑ x가 100보다 클 때
Ⓒ x가 100일 때
Ⓓ 그냥 적당할 때

3 **다음 중에서 루프 구조가 아닌 것은?**

Ⓐ for
Ⓑ do−while
Ⓒ while
Ⓓ repeat until

4 **do−while 루프는 최소한 몇 번 실행되는가?**

Ⓐ 0
Ⓑ 무한 반복
Ⓒ 1
Ⓓ 그때그때 상황에 따라 다르다

CHAPTER 6

1 **다음 원형 중에서 올바르지 않은 것은 무엇인가?**

Ⓐ int funct(char x, char y);
Ⓑ double funct(char x)
Ⓒ void funct();
Ⓓ char x();

2 **원형이 int func(char x, double v, float t);인 함수에서 다음 중 리턴 타입은 무엇인가?**

Ⓐ char **Ⓑ int** Ⓒ float Ⓓ double

3 **다음 중에서 유효한 함수 호출(함수가 종료한다고 가정한다)은 무엇인가?**

Ⓐ funct; Ⓑ funct x, y;
Ⓒ funct(); Ⓓ int funct();

4 **다음 중에서 온전한 함수는 무엇인가?**

Ⓐ int funct();
Ⓑ int funct(int x) {return x=x+1;}
Ⓒ void funct(int) {cout<<"Hello"}
Ⓓ void funct(x) {cout<<"Hello";}

CHAPTER 7

1 **다음 중에서 case문 뒤에 등장해야 하는 것은 무엇인가?**

Ⓐ : Ⓑ ; Ⓒ − Ⓓ 새로운 행

2 **어떤 case문을 실행한 뒤 다음 case문으로 넘어가지 못하게 하려면 무엇이 필요한가?**

Ⓐ end; **Ⓑ** break;
Ⓒ Stop; Ⓓ 세미콜론

3 **처리하지 못한 경우를 다루는 데 사용하는 키워드는 무엇인가?**

Ⓐ all Ⓑ contingency
Ⓒ default Ⓓ other

4 다음 코드의 실행 결과는 무엇인가?

```
int x = 0;
switch( x )
{
      case 1: cout << "One";
      case 0: cout << "Zero";
      case 2: cout << "Hello World";
}
```

Ⓐ One Ⓑ Zero
Ⓒ Hello World **Ⓓ ZeroHello World**

CHAPTER 8

1 **rand를 호출하기 전에 srand를 호출하지 않으면 어떻게 되는가?**

Ⓐ rand가 동작하지 않는다.
Ⓑ rand는 언제나 0을 리턴한다.
Ⓒ rand는 프로그램이 실행될 때마다 언제나 동일한 수들을 리턴한다.
Ⓓ 어떤 일도 일어나지 않는다.

2 **왜 현재 시간으로 srand에 시드를 제공하는가?**

Ⓐ 프로그램이 언제나 같은 방식으로 실행되도록 하기 위해서
Ⓑ 프로그램이 실행될 때마다 새로운 난수를 만들어내기 위해서
Ⓒ 컴퓨터에서 진정한 난수를 만들어내기 위해서
Ⓓ 프로그래머 편의 기능이다. 매번 같은 시드를 설정하려면 단순히 srand만 호출한다.

3 **rand는 어떤 범위의 값을 리턴하는가?**

Ⓐ 프로그래머가 원하는 범위
Ⓑ 0에서 1000까지
Ⓒ 0에서 RAND_MAX까지
Ⓓ 1에서 RAND_MAX까지

4 **11 % 3이라는 수식이 리턴하는 것은 무엇인가?**

Ⓐ 33 Ⓑ 3 Ⓒ 8 **Ⓓ 2**

5 **srand는 언제 사용해야 하는가?**

Ⓐ 난수가 필요할 때마다
Ⓑ 사용할 일이 없다. 단지 겉치레에 불과하다.
Ⓒ 프로그램이 시작할 때 한 번
Ⓓ rand를 사용하고 난 뒤 임의성을 높이기 위해 생각날 때마다 한 번씩

CHAPTER 10

1 **다음 중에서 배열을 정확하게 선언한 것은 무엇인가?**

Ⓐ int anarray[10];
Ⓑ int anarray;
Ⓒ anarray{ 10 };
Ⓓ array anarray[10];

2 **요소가 모두 29개인 배열에서 마지막 요소의 인덱스 번호는 무엇인가?**

Ⓐ 29
Ⓑ 28
Ⓒ 0
Ⓓ 프로그래머가 따로 정함

3 **다음 중에서 2차원 배열을 나타내는 것은 무엇인가?**

Ⓐ array anarray[20][20];
Ⓑ int anarray[20][20];
Ⓒ int array[20, 20];
Ⓓ char array[20];

4 **다음 중에서 100개의 요소가 담긴 foo 배열에서 7번째 요소에 올바로 접근한 것은 무엇인가?**

Ⓐ foo[6]; Ⓑ foo[7]; Ⓒ foo(7); Ⓓ foo;

5 **다음 중에서 2차원 배열을 넘겨받는 함수가 올바로 선언된 것은 무엇인가?**

Ⓐ int func (int x[][]);
Ⓑ int func (int x[10][]);
Ⓒ int func (int x[]);
Ⓓ int func (int x[][10]);

CHAPTER 11

1 **b 구조체의 변수에 접근하는 것은 무엇인가?**

Ⓐ b->var;
Ⓑ b.var;
Ⓒ b-var;
Ⓓ b>var;

2 **구조체를 올바로 정의한 것은 무엇인가?**

Ⓐ struct {int a;}
Ⓑ struct a_struct {int a};
Ⓒ struct a_struct int a;
Ⓓ struct a_struct {int a;};

3 **foo 타입의 구조체 변수인 my_foo를 올바로 선언한 것은 무엇인가?**

Ⓐ my_foo as struct foo;
Ⓑ foo my_foo;
Ⓒ my_foo;
Ⓓ int my_foo;

4 **이 코드를 실행하면 얻을 수 있는 최종 값은 무엇인가?**

```
#include <iostream>

using namespace std;

struct MyStruct
{
      int x;
};

void updateStruct (MyStruct my_struct)
{
      my_struct.x = 10;
}

int main ()
{
      MyStruct my_struct;
      my_struct.x = 5;
      updateStruct( my_struct );
      cout << my_struct.x << '\n';
}
```

Ⓐ 5
Ⓑ 10
Ⓒ 이 코드는 컴파일되지 않는다.

CHAPTER 12

1 **다음 중에서 포인터를 사용하는 이유로 적절하지 않은 것은?**

Ⓐ 함수로 인수가 전달되도록 하기 위해서
Ⓑ 크기가 큰 변수를 복사하지 않고 공간을 절약하기 위해서
Ⓒ 운영체제로부터 더 많은 메모리를 가져오기 위해서
Ⓓ 변수에 더 빨리 접근하기 위해서

2 **포인터가 저장하는 것은 무엇인가?**

Ⓐ 다른 변수의 이름
Ⓑ 정수 값
Ⓒ 다른 변수의 메모리 주소
Ⓓ 메모리 주소, 그러나 다른 변수일 필요는 없음

3 **프로그램이 실행되는 동안 메모리를 더 많이 가져올 수 있는 곳은 어디인가?**

Ⓐ 메모리를 더 많이 가져올 수 없다.
Ⓑ 스택
Ⓒ 자유 저장소(힙)
Ⓓ 다른 변수를 선언함으로써

4 **다음 중에서 포인터를 사용할 때 잘못될 수 있는 것은?**

Ⓐ 사용할 수 없는 메모리에 접근하면 프로그램은 예기치 않게 종료된다.
Ⓑ 잘못된 메모리 주소에 접근하면 데이터가 손상된다.
Ⓒ 잊어버리고 메모리를 OS에 반환하지 않으면 메모리 부족 현상이 일어난다.
Ⓓ 위의 경우 모두

5 **함수에서 선언된 일반적인 변수에 할당되는 메모리는 어느 곳에서 마련되는가?**

Ⓐ 자유 저장소(힙)
Ⓑ 스택
Ⓒ 일반적인 변수는 메모리를 사용하지 않는다.
Ⓓ 프로그램의 바이너리 자체(EXE의 크기가 큰 이유가 바로 이 때문임!)

6 **메모리를 할당하면 메모리로 무엇을 해야 하는가?**

Ⓐ 없다. 메모리 영구 소유권이 여러분에게 있다.
Ⓑ 메모리를 사용한 뒤에는 운영체제에 반환한다.
Ⓒ 포인터가 가리키는 값을 0으로 설정한다.
Ⓓ 0을 포인터에 저장한다.

CHAPTER 13

1 **다음 중에서 올바른 포인터 선언은 무엇인가?**

Ⓐ int x; Ⓑ int &x; Ⓒ ptr x; **Ⓓ int *x;**

2 **다음 중에서 정수 변수인 a의 메모리 주소를 나타내는 것은 무엇인가?**

Ⓐ *a; Ⓑ a; **Ⓒ &a;** Ⓓ address(a);

3 **다음 중에서 포인터 p_a가 가리키는 변수의 메모리 주소를 나타내는 것은 무엇인가?**

Ⓐ p_a; Ⓑ *p_a;
Ⓒ &p_a; Ⓓ address(p_a);

4 **다음 중에서 포인터 p_a가 가리키는 주소에 저장된 값을 나타내는 것은 무엇인가?**

Ⓐ p_a; Ⓑ val(p_a); **Ⓒ *p_a;** Ⓓ &p_a;

5 **다음 중에서 레퍼런스를 올바로 선언한 것은 무엇인가?**

Ⓐ int *p_int;
Ⓑ int &my_ref;
Ⓒ int &my_ref = & my_orig_val;
Ⓓ int &my_ref = my_orig_val;

6 다음 중에서 레퍼런스를 사용하기에 적절하지 않은 상황은 무엇인가?

Ⓐ 자유 저장소(힙)에서 동적으로 할당된 주소를 저장할 때
Ⓑ 큰 값을 함수로 전달할 때 이를 복사하지 않기 위해
Ⓒ 함수의 파라미터가 NULL이 아님을 분명히 할 때
Ⓓ 함수가 자신에게로 넘어온 원래 변수에 포인터를 사용하지 않고 접근해야 할 때

CHAPTER 14

1 C++에서 메모리를 할당하기 위한 올바른 키워드는 무엇인가?

Ⓐ new
Ⓑ malloc
Ⓒ create
Ⓓ value

2 C++에서 메모리를 반환하기 위한 올바른 키워드는 무엇인가?

* 인정할 것은 인정하자! 이 두 문제의 정답으로 malloc과 free를 골랐다면 딴죽을 걸고 싶지는 않지만 이는 이 장을 읽지 않았다는 방증이다. 그 둘은 C에 등장하는 키워드이기 때문이다.

Ⓐ free **Ⓑ** delete
Ⓒ clear Ⓓ remove

3 참인 문장은 어느 것인가?

Ⓐ 배열과 포인터는 똑같다.
Ⓑ 배열은 포인터로 대입될 수 없다.
Ⓒ 포인터는 배열처럼 취급될 수 있지만, 그렇다고 해서 배열은 아니다.
Ⓓ 포인터를 배열처럼 사용할 수는 있지만, 포인터를 배열처럼 할당할 수는 없다.

4 다음 코드에서 x, p_int, p_p_int의 최종 값은 무엇인가?

❗ 정수와 포인터는 타입이 서로 다르기 때문에 컴파일러는 이 코드를 그대로 인식하지 않는다. 다만, 이 퀴즈는 포인터가 여럿일 때 그 처리 과정을 종이에 직접 써 가며 이해하는 데는 유용하다.

```
int x = 0;
int *p_int = & x;
int **p_p_int = & p_int;
*p_int = 12;
**p_p_int = 25;
p_int = 12;
*p_p_int = 3;
p_p_int = 27;
```

Ⓐ x = 0, p_p_int = 27, p_int = 12
Ⓑ x = 25, p_p_int = 27, p_int = 12
Ⓒ x = 25, p_p_int = 27, p_int = 3
Ⓓ x = 3, p_p_int = 27, p_int = 12

5 포인터가 가리키는 값이 유효하지 않다는 것을 어떻게 알 수 있나?

Ⓐ 포인터를 음수로 설정한다.
Ⓑ 포인터를 NULL로 설정한다.
Ⓒ 포인터와 연결된 메모리를 반환한다.
Ⓓ 포인터를 거짓으로 설정한다.

CHAPTER 15

1 배열에 비해 연결 리스트가 가지는 장점은 무엇인가?

Ⓐ 연결 리스트가 요소당 공간을 덜 차지한다.
Ⓑ 연결 리스트가 동적으로 커질 수 있으므로 기존 요소를 복사하지 않고도 새로운 요소를 담을 수 있다.
Ⓒ 연결 리스트가 더 빠르게 특정 요소를 찾을 수 있다.
Ⓓ 연결 리스트가 구조체를 요소로 담을 수 있다.

2 다음 중에서 참인 문장은 무엇인가?

Ⓐ 만에 하나라도 배열을 사용할 이유는 없다.
Ⓑ 연결 리스트와 배열은 같은 성능을 보인다.
Ⓒ 연결 리스트와 배열은 인덱스별로 요소에 접근하는 시간을 일정하게 유지한다.
Ⓓ 중간에 요소를 삽입할 때는 배열일 때보다 연결 리스트일 때가 더 빠르다.

3 언제 연결 리스트를 사용하게 되는가?

Ⓐ 하나의 항목만을 저장해야 할 때
Ⓑ 저장할 항목의 개수를 컴파일 타임에 알게 될 때
Ⓒ 항목을 동적으로 추가하고 제거해야 할 때
Ⓓ 정렬된 리스트에서 어떤 항목에 반복 구조 없이 곧바로 접근해야 할 때

4 리스트 항목을 가리키는 레퍼런스로 연결 리스트를 선언해도 되는 이유는 무엇인가?
(struct Node { Node *p_next; };)

Ⓐ 허용되지 않는 선언 방식이다.
Ⓑ 항목을 자체 참조하기 위한 메모리가 필요 없다는 사실을 컴파일러가 파악할 수 있기 때문에
Ⓒ 타입이 포인터이므로 포인터 하나를 담을 공간만 필요하며, 따라서 실제 다음 노드에 필요한 메모리는 나중에 할당되기 때문에
Ⓓ 다른 구조체를 가리키기 위해 p_next를 대입하지만 않는다면 허용되는 선언 방식이다.

5 왜 연결 리스트의 끝에 NULL이 있어야 하는가?

Ⓐ NULL은 리스트의 끝을 알리고, 초기화되지 않은 메모리에 접근하지 못하도록 막아준다.
Ⓑ NULL은 리스트가 순환 레퍼런스가 되지 않도록 막아준다.
Ⓒ NULL은 디버깅 도우미다. 리스트 안으로 너무 깊이 내려가면 프로그램은 예기치 않게 종료된다.
Ⓓ NULL을 저장하지 않으면 리스트는 자체 레퍼런스 때문에 메모리가 무한대로 필요하게 된다.

6 배열과 연결 리스트의 닮은 점은 무엇인가?

Ⓐ 둘 다 현재 리스트의 중간에 새로운 요소를 빠르게 추가할 수 있도록 한다.
Ⓑ 둘 다 데이터를 순차적으로 저장하여 해당 데이터에 순차적으로 접근할 수 있도록 한다.
Ⓒ 둘 다 요소들이 삽입되면 크기가 쉽게 커진다.
Ⓓ 둘 다 모든 요소에 빠르게 접근할 수 있도록 한다.

CHAPTER 16

1 꼬리 되부름이란 무엇인가?

Ⓐ 강아지를 부를 때
Ⓑ 함수가 자신을 부를 때
Ⓒ 되부름 함수가 돌아가기 전에 마지막으로 하는 것으로 자신을 부를 때
Ⓓ 되부름 알고리즘을 루프로 작성할 수 있을 때

2 되부름을 언제 사용하는 것이 좋은가?

Ⓐ 루프로 알고리즘을 작성할 수 없을 때
Ⓑ 알고리즘을 루프의 관점이 아니라 하위 문제의 관점에서 더 자연스럽게 표현할 수 있을 때
Ⓒ 사용할 일이 없다. 너무 어렵다. ☹
Ⓓ 배열, 연결 리스트를 함께 처리할 때

3 되부름 알고리즘의 필수 요소는 무엇인가?

Ⓐ 탈출 조건, 되부름 호출
Ⓑ 탈출 조건, 문제를 더 작은 문제로 쪼갤 수 있는 방법
Ⓒ 원래 문제의 더 작아진 버전을 다시 결합할 수 있는 방법
Ⓓ 위의 경우 모두

4 탈출 조건이 만족되지 않으면 어떤 일이 일어나는가?

Ⓐ 알고리즘이 일찍 끝날 수 있다.
Ⓑ 컴파일러가 이를 감지하고 관련 메시지를 쏟아낸다.
Ⓒ 문제가 되는 상황이 아니다.
Ⓓ 스택 오버플로가 생길 수 있다.

CHAPTER 17

1 바이너리 트리의 기본적인 장점은 무엇인가?

Ⓐ 바이너리 트리는 포인터를 사용한다.
Ⓑ 바이너리 트리는 데이터의 양에 상관없이 저장한다.
Ⓒ 바이너리 트리는 데이터의 빠른 탐색을 가능하게 한다.
Ⓓ 바이너리 트리에서 제거하는 연산이 쉽다.

2 바이너리 트리 대신 연결 리스트를 사용하는 것이 바람직할 때는 언제인가?

Ⓐ 빠른 탐색을 가능하게 하는 데이터가 필요할 때
Ⓑ 정렬된 순서로 요소에 접근할 수 있어야 할 때
Ⓒ 중간에는 항목들에 접근할 일이 없고 앞이나 뒤에 빠르게 추가할 수 있어야 할 때
Ⓓ 사용하고 있는 메모리를 반환할 필요가 없을 때

3 다음 중에서 참인 문장은 어느 것인가?

Ⓐ 항목을 바이너리 트리에 추가하는 순서는 트리 구조를 변경할 수 있다.
Ⓑ 바이너리 트리는 항목이 정렬된 순서로 추가되어야 최고의 구조를 제공할 수 있다.
Ⓒ 바이너리 트리에 요소들이 무작위로 삽입되었다면 요소를 찾는 데 걸리는 시간은 연결 리스트가 훨씬 빠르다.
Ⓓ 바이너리 트리는 아무리 줄어들어도 연결 리스트와 같은 구조가 될 수 없다.

4 다음 중에서 바이너리 트리의 노드 탐색 시간이 빠른 이유를 가장 나타낸 것은?

Ⓐ 빠르지 않다. 포인터가 두 개 있다는 것은 트리 운행 연산도 더 많다는 의미다.
Ⓑ 트리 아래로 한 단계씩 내려갈 때마다 탐색해야 하는 노드의 수가 대략 반으로 줄어들기 때문이다.
Ⓒ 연결 리스트보다 나은 점이 하나도 없다.
Ⓓ 바이너리 트리의 되부름 호출은 연결 리스트의 루프 반복보다 빠르다.

CHAPTER 18

1 벡터를 사용하기에 적절한 때는 언제인가?

Ⓐ 키와 값의 짝을 저장해야 할 때
Ⓑ 항목들을 변경하면서 성능을 최대한으로 끌어올려야 할 때
Ⓒ 데이터 구조를 업데이트하면서 세부적인 내용을 신경 쓰고 싶지 않을 때
Ⓓ 구직 면접 때 입는 양복처럼 벡터 또한 항상 적절하게 사용할 수 있다.

2 맵에서 모든 항목을 동시에 제거하려면 어떻게 해야 하는가?

Ⓐ 항목을 빈 문자열로 설정한다.
Ⓑ erase를 호출한다.
Ⓒ empty를 호출한다.
Ⓓ clear를 호출한다.

3 데이터 구조를 직접 구현해야 할 때는 언제인가?

Ⓐ 정말로 빠른 것이 필요할 때
Ⓑ 더욱 강력한 구조가 필요할 때
Ⓒ 원시 데이터 구조의 장점을 활용해야 할 때, 가령 수식 트리를 만들 때
Ⓓ 사실 내키지 않는데도 데이터 구조를 직접 구현할 이유는 없다.

4 다음 중에서 **vector<int>**와 함께 사용할 수 있는 반복자를 올바로 선언한 것은?

Ⓐ iterator<int> itr;
Ⓑ vector::iterator itr;
Ⓒ vector<int>::iterator itr;
Ⓓ vector<int>::iterator<int> itr;

5 다음 중에서 맵의 현재 반복자가 가리키는 요소의 키에 올바로 접근하는 것은?

Ⓐ itr.first
Ⓑ itr->first
Ⓒ itr->key
Ⓓ itr.key

6 반복자가 사용될 수 있는지 어떻게 구별하는가?

Ⓐ NULL과 비교한다.
Ⓑ 반복 처리하고 있는 컨테이너에 end()를 호출한 결과와 비교한다.
Ⓒ 0과 비교한다.
Ⓓ 반복 처리하고 있는 컨테이너에 begin()을 호출한 결과와 비교한다.

CHAPTER 19

1 다음 중에서 유효한 코드는?

Ⓐ const int& x;
Ⓑ const int x = 3; int *p_int = & x;
Ⓒ const int x = 12; const int *p_int = & x;
Ⓓ int x = 3; const int y = x; int& z = y;

2 다음 함수 선언 중 아래 코드를 올바로 컴파일할 수 있는 것은?

```
const int x = 3; fun( x );
```

Ⓐ void fun (int x);
Ⓑ void fun (int& x);
Ⓒ void fun (const int& x);
Ⓓ A와 C

3 다음 중에서 문자열 검색이 실패했는지 알려줄 수 있는 최고의 방법은?

Ⓐ 결과 위치를 0과 비교한다.
Ⓑ 결과 위치를 –1과 비교한다.
Ⓒ 결과 위치를 string::npos와 비교한다.
Ⓓ 결과 위치가 문자열의 길이보다 큰지 확인한다.

4 const STL 컨테이너에 사용할 반복자는 어떻게 만들어야 하는가?

Ⓐ 반복자를 const로 선언한다.
Ⓑ 반복자보다는 인덱스를 사용하여 반복 처리한다.
Ⓒ const_iterator를 사용한다.
Ⓓ 템플릿 타입을 const로 선언한다.

CHAPTER 21

1 다음 중에서 C++ 빌드 프로세스의 세부 단계가 아닌 것은 무엇인가?

Ⓐ 링크 Ⓑ 컴파일 Ⓒ 전처리 **Ⓓ 후처리**

2 정의되지 않은 함수와 관련된 오류는 어느 곳에서 발생하는가?

Ⓐ 링크 단계 Ⓑ 컴파일 단계
Ⓒ 프로그램 시작 Ⓓ 함수 호출 시

3 다음 중에서 헤더 파일을 여러 번 포함하면 일어날 수 있는 일은 무엇인가?

Ⓐ 다중 선언과 관련된 오류
Ⓑ 없다. 헤더 파일은 언제나 한 번만 로드된다.
Ⓒ 헤더 파일의 구현 방식에 따라 다르다.
Ⓓ 헤더 파일은 한 번에 소스 파일 하나에만 포함되므로 문제될 것이 없다.

4 개별 컴파일 및 링크 단계를 적용하면 얻을 수 있는 장점은 무엇인가?

Ⓐ 없다. 개별 컴파일 및 링크 단계는 혼동만 일으키고, 프로그램이 여럿 실행되기 때문에 더 느려지기만 한다.

Ⓑ 오류가 링커에서 비롯되었는지, 컴파일러에서 비롯되었는지 알 수 있기 때문에 오류를 진단하기가 훨씬 더 수월해진다.
Ⓒ 변경된 파일만 재컴파일할 수 있어 컴파일 및 링크 시간을 절약할 수 있다.
Ⓓ 변경된 파일만 재컴파일할 수 있어 컴파일 시간을 절약할 수 있다.

CHAPTER 22

1 데이터에 직접 접근하지 않고 함수를 사용하여 얻을 수 있는 장점은 무엇인가?

Ⓐ 함수는 더 빠른 접근을 제공하기 위해 컴파일러가 최적화할 수 있다.
Ⓑ 함수는 그 구현 방식을 콜러에게서 숨길 수 있어서 자신의 콜러를 쉽게 변경할 수 있다.
Ⓒ 함수는 동일한 데이터 구조를 여러 소스 파일에서 사용할 수 있는 유일한 방법이다.
Ⓓ 장점이 없다.

2 언제 코드를 공용 함수에 두어야 하는가?

Ⓐ 호출해야 할 때마다
Ⓑ 여러 곳에서 동일한 코드를 호출하기 시작했을 때
Ⓒ 함수가 너무 커 컴파일할 수 없다며 컴파일러가 투덜거리기 시작할 때
Ⓓ B와 C의 경우에

3 데이터 구조의 표현 방식을 숨겨야 하는 이유는 무엇인가?

Ⓐ 데이터 구조를 더욱더 쉽게 교체하기 위해
Ⓑ 데이터 구조를 사용하는 코드를 더욱더 쉽게 이해하기 위해
Ⓒ 데이터 구조를 새로운 코드에서 더욱더 쉽게 사용하기 위해
Ⓓ 위의 경우 모두

CHAPTER 23

1 구조체의 필드를 직접 사용하지 않고 메소드를 사용하는 이유는 무엇인가?

Ⓐ 메소드를 읽기가 더 쉽기 때문에
Ⓑ 메소드가 더 빠르기 때문에
Ⓒ 메소드를 사용하지 않는 편이 더 낫다. 구조체의 필드를 직접 사용해야 한다.
Ⓓ 데이터의 표현 방식을 변경할 수 있기 때문에

2 다음 중에서 struct MyStruct { int func(); }; 구조체와 연결된 메소드를 정의하는 것은 무엇인가?

Ⓐ int func() { return 1; }
Ⓑ MyStruct::int func() { return 1; }
Ⓒ int MyStruct::func() { return 1; }
Ⓓ int MyStruct func () { return 1; }

3 클래스에 메소드 정의를 포함하는 이유는 무엇인가?

Ⓐ 클래스 사용자가 그 동작 방식을 확인할 수 있기 때문에
Ⓑ 코드를 더 빨리 실행할 수 있기 때문에
Ⓒ 포함하면 안 된다! 세부적인 구현 방식이 유출된다.
Ⓓ 포함하면 안 된다! 프로그램의 실행 속도가 느려진다.

CHAPTER 24

1 비공개 데이터를 사용하는 이유는 무엇인가?

Ⓐ 해커로부터 데이터를 안전하게 지키기 위해
Ⓑ 다른 프로그래머가 데이터를 만지지 못하도록 하기 위해
Ⓒ 어떤 데이터만이 클래스 구현에 사용되는지 명확히 하기 위해
Ⓓ 사용할 이유가 없다. 프로그램만 이해하기가 어려워진다.

2 클래스와 구조체는 무엇이 다른가?

Ⓐ 전혀 다르지 않다.
Ⓑ 클래스는 모든 것이 공개로 설정되는 것이 디폴트다.
Ⓒ 클래스는 모든 것이 비공개로 설정되는 것이 디폴트다.
Ⓓ 클래스는 필드의 공개 여부를 알려주지만, 구조체는 그렇게 하지 못한다.

3 클래스의 데이터 필드에 무엇을 해야 하는가?

Ⓐ 디폴트로 공개 설정을 한다.
Ⓑ 디폴트로 비공개 설정을 한다. 다만, 필요할 때 공개로 전환한다.
Ⓒ 절대로 공개 설정을 하지 않는다.
Ⓓ 클래스는 데이터를 담지 않는 경우가 대부분이다. 만일 담는다면 골치가 아플 것이다.

4 메소드가 공개로 설정되어야 하는지 어떻게 결정하는가?

Ⓐ 메소드는 절대로 공개 설정을 하지 않는다.
Ⓑ 메소드는 항상 공개 설정을 해야 한다.
Ⓒ 클래스의 주 기능을 사용해야 한다면 메소드를 공개 설정한다. 그렇지 않을 때는 비공개로 설정한다.
Ⓓ 누군가 메소드를 사용할 가능성이 있다면 공개로 설정한다.

CHAPTER 25

1 클래스에 사용할 생성자는 언제 작성해야 하는가?

Ⓐ 항상 작성해야 한다. 단, 사용할 수 없는 생성자가 클래스에 없을 때만.

Ⓑ 클래스를 디폴트가 아닌 값으로 초기화해야 할 때마다

Ⓒ 작성할 이유가 전혀 없다. 컴파일러는 생성자를 항상 제공한다.

Ⓓ 파괴자도 있어야 할 때만.

2 파괴자와 대입 연산자는 어떤 관계를 보이는가?

Ⓐ 전혀 관계가 없다.

Ⓑ 클래스의 파괴자는 대입 연산자를 실행하기 전에 호출된다.

Ⓒ 대입 연산자는 파괴자가 어떤 메모리를 삭제해야 하는지 지정해야 한다.

Ⓓ 대입 연산자는 복사된 클래스와 새 클래스의 파괴자 둘 다 실행하는 것이 안전하다고 확신시켜야 한다.

3 언제 초기화 리스트를 사용해야 하는가?

Ⓐ 생성자의 효율성을 최대화하고 빈 객체들을 구성하지 못하도록 금지할 때

Ⓑ 상수 값을 초기화할 때

Ⓒ 클래스에서 필드의 비디폴트 생성자를 실행할 때

Ⓓ 위의 경우 모두

4 다음 코드에서 두 번째 행에서는 어떤 함수가 실행되는가?

```
string str1;
string str2 = str1;
```

Ⓐ str2의 생성자와 str1의 대입 연산자

Ⓑ str2의 생성자와 str2의 대입 연산자

Ⓒ str2의 복사 생성자

Ⓓ str2의 대입 연산자

5 다음 코드에서는 어떤 함수가 호출되는가? 그리고 어떤 순서로 호출되는가?

```
{
      string str1;
      string str2;
}
```

Ⓐ str1의 생성자가 먼저, str2의 생성자가 그 다음

Ⓑ str1의 파괴자가 먼저, str2의 생성자가 그 다음

Ⓒ str1의 생성자가 가장 먼저, str2의 생성자가 그 다음, 이어서 str1의 파괴자, 마지막으로 str2의 파괴자

Ⓓ str1의 생성자가 가장 먼저, str2의 생성자가 그 다음, 이어서 str2의 파괴자, 마지막으로 str1의 파괴자

6 클래스에 비디폴트 복사 생성자가 있을 때 다음 중에서 대입 연산자에 관한 설명 중 올바른 것은?

Ⓐ 디폴트 대입 연산자를 가져야 한다.

Ⓑ 비디폴트 대입 연산자를 가져야 한다.

Ⓒ 선언은 되었지만, 아직 구현되지 않은 대입 연산자를 가져야 한다.

Ⓓ B든 C든 어느 하나는 맞는 말이다.

CHAPTER 26

1 슈퍼클래스의 파괴자는 언제 실행되는가?

Ⓐ 슈퍼클래스를 가리키는 포인터에 delete를 호출하여 객체가 파괴되는 경우에만

Ⓑ 서브클래스의 파괴자가 호출되기 전

Ⓒ 서브클래스의 파괴자가 호출된 뒤

Ⓓ 서브클래스의 파괴자가 호출되는 동안

2 다음 클래스 계층 구조에서는 Cat의 생성자로 무엇을 해야 하는가?

```
class Mammal {
public:
      Mammal (const string& species_name);
};

class Cat : public Mammal
{
public:
       Cat();
};
```

Ⓐ 특별히 해야 할 것은 없다.

Ⓑ 이니셜라이저 리스트를 사용하여 'cat'이라는 인수로 Mammal의 생성자를 호출한다.

Ⓒ Mammal의 생성자를 Cat 생성자 안에서 'cat'이라는 인수로 호출한다.

Ⓓ Cat 생성자를 제거하고 디폴트 생성자를 사용해야 한다. 그래야 이 문제를 해결할 수 있다.

3 다음 클래스 정의 중에서 잘못된 것은 무엇인가?

```
class Nameable
{
      virtual string getName();
};
```

Ⓐ getName 메소드를 '공개'로 설정하지 않았다.

Ⓑ 가상 파괴자가 없다.

Ⓒ getName을 구현하지 않았는데도 getName을 '순수 가상'으로 설정하지 않았다.

Ⓓ 위의 내용 모두 잘못되었다.

4 인터페이스 클래스에서 가상 메소드를 선언하면, 이 인터페이스 메소드를 사용하여 서브클래스에 메소드를 호출하기 위해 무엇을 해야 하는가?

Ⓐ 인터페이스를 포인터(또는 레퍼런스)로 받는다.
Ⓑ 할 일이 없다. 객체를 복사만 할 수 있다.
Ⓒ 메소드를 실행할 서브클래스의 이름을 알아야 한다.
Ⓓ 무엇을 해야 할지 막막하다. 가상 메소드가 도대체 무엇인가?

5 재사용은 어떻게 상속을 통해 개선되는가?

Ⓐ 슈퍼클래스의 메소드를 상속할 수 있는 코드를 작성하여
Ⓑ 슈퍼클래스에서 서브클래스의 가상 메소드를 구현할 수 있도록 하여
Ⓒ 구체적인 클래스보다는 인터페이스를 처리하는 코드를 작성하여. 이때, 새로운 클래스에서는 인터페이스를 구현하고 해당 코드를 사용할 수 있다.
Ⓓ 새로운 클래스에서 가상 메소드와 사용될 수 있는 구체적인 클래스의 특징을 상속하도록 하여

6 다음 중에서 클래스 접근 수준에 대한 설명 중 올바른 것은 무엇인가?

Ⓐ 서브클래스는 오직 부모 클래스의 '공개' 메소드와 데이터에만 접근할 수 있다.
Ⓑ 서브클래스는 부모 클래스의 '전용' 메소드와 데이터에 접근할 수 있다.
Ⓒ 서브클래스는 오직 부모 클래스의 '보호' 메소드와 데이터에만 접근할 수 있다.
Ⓓ 서브클래스는 부모 클래스의 '보호' 또는 '공개' 메소드와 데이터에 접근할 수 있다.

CHAPTER 27

1 using namespace 지시자는 언제 사용해야 하는가?

Ⓐ 모든 헤더 파일에서 include 바로 뒤에
Ⓑ 사용할 일이 없다. 위험하기만 한 일종의 목발과도 같다.
Ⓒ 네임스페이스가 서로 충돌하지 않는 모든 cpp 파일에서 맨 위에
Ⓓ 해당 네임스페이스의 변수를 사용하기 바로 전에

2 네임스페이스를 사용해야 하는 이유는 무엇인가?

Ⓐ 컴파일러를 작성하는 사람에게 흥미로운 일거리를 제공하기 위해
Ⓑ 코드의 더 강도 높은 캡슐화를 제공하기 위해
Ⓒ 대형 코드베이스에서 이름 충돌을 막기 위해
Ⓓ 클래스의 용도를 명확히 하기 위해

3 언제 네임스페이스에 코드를 넣어야 하는가?

Ⓐ 항상
Ⓑ 파일이 수십 개 이상일 만큼 충분히 큰 프로그램을 개발할 때
Ⓒ 다른 사람과 공유할 라이브러리를 개발할 때
Ⓓ B와 C

4 using namespace 선언을 헤더 파일에 두면 안 되는 이유는 무엇인가?

Ⓐ 문법에 맞지 않아서
Ⓑ 두지 않을 이유가 없다. using 선언문은 헤더 파일 안에서만 유효하다.
Ⓒ 헤더 파일을 포함하는 모든 사람은 설령 충돌이 일어나더라도 using 선언문을 사용해야 한다.
Ⓓ 여러 헤더 파일이 using 선언문을 포함한다면 충돌이 일어날 수도 있어서

CHAPTER 28

1 파일에서 읽으려면 어떤 타입을 사용할 수 있는가?

Ⓐ ifstream Ⓑ ofstream Ⓒ fstream **Ⓓ A와 C**

2 다음 중에서 참인 내용은 어느 것인가?

Ⓐ 텍스트 파일은 바이너리 파일보다 공간을 덜 차지한다.
Ⓑ 바이너리 파일은 디버깅하기가 더 쉽다.
Ⓒ 바이너리 파일은 텍스트 파일보다 공간을 더 효율적으로 사용한다.
Ⓓ 텍스트 파일은 실제 프로그램에서 사용하지 못할 만큼 느리다.

3 바이너리 파일에 쓸 때는 왜 포인터를 문자열 객체에 전달할 수 없는가?

Ⓐ write 메소드에는 반드시 char*를 전달해야 하기 때문에
Ⓑ 문자열 객체는 메모리에 담길 수 없기 때문에
Ⓒ 문자열 객체의 배치도를 알지 못하기 때문에. 문자열 객체에는 포인터가 포함되기도 한다.
Ⓓ 문자열이 너무 큰데다 하나하나 써야 하기 때문에

4 다음 중에서 파일 형식에 관한 내용 중 참인 것은 무엇인가?

Ⓐ 파일 형식은 다른 입력만큼 변경하기가 쉽다.
Ⓑ 파일 형식을 변경하려면 구 버전의 프로그램에서 새 버전의 파일을 읽을 때 어떤 일이 일어날지 고려해야 한다.
Ⓒ 파일 형식을 디자인하려면 새 버전의 프로그램에서 구 버전의 파일을 열 때 어떤 일이 일어날지 고려해야 한다.
Ⓓ B와 C

CHAPTER 29

1 언제 템플릿을 사용해야 하는가?

Ⓐ 시간을 절약해야 할 때
Ⓑ 코드가 더 빨리 실행되도록 해야 할 때
Ⓒ 같은 코드를 타입만 달리 하여 여러 번 작성해야 할 때
Ⓓ 나중에 다시 코드를 재사용하도록 확실히 해두어야 할 때

2 언제 템플릿 파라미터에 타입을 제공해야 하는가?

Ⓐ 항상
Ⓑ 템플릿 클래스의 인스턴스를 선언할 때만
Ⓒ 타입을 추론할 수 없을 때에만
Ⓓ 템플릿 함수에서는 타입을 추론할 수 없을 때에만. 템플릿 클래스에서는 항상.

3 어떻게 컴파일러는 템플릿 파라미터가 어떤 템플릿과 사용될 수 있는지 구분하는가?

Ⓐ 컴파일러는 특정 C++ 인터페이스를 구현한다.
Ⓑ 템플릿을 선언할 때 제한 조건을 지정해야 한다.
Ⓒ 컴파일러는 템플릿 파라미터를 사용하려고 한다. 필요한 모든 연산을 해당 타입이 지원하면 컴파일러는 이 타입을 받아들인다.
Ⓓ 템플릿을 선언할 때는 유효한 모든 템플릿 타입을 나열해야 한다.

4 템플릿 클래스를 헤더 파일에 두면 일반적인 클래스를 헤더 파일에 둘 때와 어떻게 달라지는가?

Ⓐ 전혀 차이가 없다.
Ⓑ 일반적인 클래스는 헤더 파일에 자신의 메소드를 정의해 둘 수 없다.
Ⓒ 템플릿 클래스는 헤더 파일에 자신의 메소드를 모두 정의해 두어야 한다.
Ⓓ 템플릿 클래스는 해당 .cpp 파일을 필요로 하지 않지만, 일반적인 클래스는 .cpp 파일이 필요하다.

5 언제 함수를 템플릿 함수로 작성해야 하는가?

Ⓐ 처음부터 작성해야 한다. 타입은 달라도 로직이 같은 코드를 언제 필요로 할지 알 수 없으므로 항상 템플릿 메소드를 작성해 두어야 한다.
Ⓑ 함수에 현재 필요한 타입으로 변환할 수 없을 때에만
Ⓒ 처음 함수와 로직은 거의 같지만 사용하는 타입이 다르고 속성이 비슷한 함수를 작성할 때마다
Ⓓ 두 함수가 '대략' 같은 일을 수행하고, 몇 가지 부울 파라미터로 로직을 세밀하게 다듬을 수 있을 때마다

6 템플릿 코드의 오류는 대개 언제 알게 되는가?

Ⓐ 템플릿을 컴파일하자마자
Ⓑ 링크 단계가 진행되는 동안
Ⓒ 프로그램을 실행할 때
Ⓓ 템플릿을 인스턴스화하는 코드를 처음 컴파일할 때