2007년 2학기 객체지향프로그래밍 중간고사

1. 1 번 문제에 대한 가능한 답 중 하나는 다음과 같다.

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

const int DoSum(const int from, const int to)
{
  if (from>to) return 0;
  else if (from==to)
  {
    cout << from << endl;
    return from;
  }
  else
  {
    cout << from << " ";
    return from+DoSum(from+1,to);
  }
}

int main()
{
  int x,y;
  cout << "두 수를 입력하세요: ";
  cin >> x >> y;
  int sum = (x<=y) ? DoSum(x,y) : DoSum(y,x);
  cout << "합 = " << sum << endl;
}
```

첨언하자면, 이 문제는 미국 회사들의 프로그래머 인터뷰에서 가끔 나온다고 알려져 있는 문제로 인터넷에서 찾아볼 수 있다. 결론부터 말하면, 이렇게 루프를 사용하지 말라고 할 경우, 가능한 답은 재귀 호출을 사용하는 것이다.

인터넷에 떠도는 원래의 문제는 루프를 사용하지 말고 1에서 100까지 더하라는 문제인데 아래와 같이 더 쉽게 구성될 수 있다.

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

const int Sum(const int number)
{
  if (number < 1) return 0; else return number+Sum(number-1);
}

int main()
{
  cout << "합 = " << Sum(100) << endl;
```

```
  return 0;
}
```

그런데 이런 경우, 1에서 n까지의 합이 $\frac{n(n+1)}{2}$ 임을 이용해서 프로그램을 작성하는 사람도 있을 수 있을 것이다.

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

int main()
{
  cout << "Enter a number: ";
  int x=0;
  cin >> x;
  cout << "Sum = " << ((double)x*((double)x+1.0)/2.0) << endl;
  return 0;
}
```

틀린 건 아니지만 문제에서 원하는 바는 아니므로 이런 경우가 나오는 걸 막기 위해, 두 수를 입력받아 차례로 출력하게 문제를 바꿨다.

내친 김에 더 사족으로, 어느 경험이 많은 일본인 C 프로그래머가 쓴 책에서 재귀 호출에 대한 불평을 본 적이 있다. 예를 들어 계승(factorial)을 작성하라고 하면 무조건 재귀 호출을 쓰는 신참 프로그래머가 있다는 것이다. 그 이유는 대부분의 C나 자바 교재에서 재귀 호출을 가르치면서 계승을 예로 들기 때문이란다. 물론 계승을 구하는 가장 좋은 방법은 단순히 루프를 쓰는 것이다. 계승은 재귀 호출의 좋은 예는 아니며, 하노이의 탑이나 퀵소트 같은 알고리즘이 오히려 좋은 예이다.

재귀 호출은 알고리즘을 다루는 사람들이나 이론 전산학자들은 좋아하지만, 사실 피할 수 있으면 피하는 게 좋다. 함수 호출에서 발생하는 스택에 대한 오버헤드가 크기 때문이다. 이진 검색이나 트리를 다루는 알고리즘들을 보면 대부분 재귀 호출을 쓰는 데, 웹 데이터나 정보 검색같이 큰 비정형 데이터의 트리를 다루게 되면, 다뤄야 할 트리의 크기나 깊이가 너무 커서 재귀 호출은 거의 무용지물이 되고 만다. 실제로 필자가 Microsoft MSN 검색 팀에 면접을 갔을 때, 트리를 재귀 호출을 사용하지 않고 다루는 방법에 대한 질문을 받은 적이 있다.

재귀 호출이 그다지 실용적이지 않긴 하지만, 어떤 경우 더 알고리즘이 보기 좋고 명확해질 수도 있고, 위에서 얘기한 바와 같이 어떤 문제들엔 루프에 비해 더 적합하기도 하다. 또한 모든 재귀 호출은 일반 루프로 바꿀 수 있고, 반대로 모든 루프도 재귀 호출로 바꿀 수 있다. 이러한 재귀 호출에 대한 지식과 프로그래밍 기술을 테스트하기 위해 본 문제를 출제한 것이다. 나머지 14 문제는 전부 프로그래밍 문제가 아닌 일반 연습 문제이다.

마지막으로 위의 재귀 호출로 만든 프로그램의 경우 1에서 100000 (십만)까지의 덧셈을 시키자 제대로 못하고 에러를 발생한다. 비쥬얼 스튜디오를 실행한 상태에서 어느 컴퓨터에선 10000 (만)까지 덧셈을 시켰는 데 제대로 못하고 에러를 발생했다. 둘 다 스택 오버헤드로 인해 오버플로우가 발생한 것이다. 위와 같은 알고리즘에선 10000까지 더하면 스택에 만 개의 스택 프레임이 쌓이고, 100000 까지 더하면 십만 개가 쌓이는 안좋

은 구조가 된다. 이 문제는 n에 대해 $n-1$ 버전을 부르는 방법 대신, n을 공평하게 반으로 나눠서 재귀 호출하는 방법으로 해결될 수 있다. 물론 이렇게 할 경우 n이 $\frac{n}{2}$로 나눠지므로, 숫자를 차례로 출력할 수는 없게 된다. 대신 트리의 깊이는 $\log \frac{n}{2}$ 로 줄어들어 훨씬 큰 값들을 더할 수 있게 된다. 아무튼, 그렇게 구현된 프로그램은 다음과 같다.

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

const long long int
  DoSum(const long long int from, const long long int to)
{
  if (from>to) return 0L;
  else if (from==to) return from;
  else
  {
    long long int mid = (from+to)/2L;
    return DoSum(from,mid)+DoSum(mid+1L,to);
  }
}

int main()
{
  cout << "두 수를 입력하세요: ";
  long long int x,y;
  cin >> x >> y;
  cout << "합 = " << ((x<=y) ? DoSum(x,y) : DoSum(y,x)) << endl;
  return 0;
}
```

값이 커지면 오버플로우가 발생하므로, 정수형의 오버플로우를 막기 위해 가장 큰 정수형인 `long long int`를 사용하였고, 필자의 컴퓨터에서 GNU G++로 컴파일해서 실행해 본 결과로는 100000 (십만) 은 물론 1000000000 (십억) 까지의 덧셈도 무리없이 수행하였다. 그 이상이 궁금한 사람은 알아서 테스트해보기 바란다.

2. `int judge(int (*)(const char *));`
3. 함수 템플릿은 다음과 같다.

```
template <class T> const T& Max(const T& a, const T& b)
{
  return a>b?a:b;
}
```

`Box`라는 구조체에 대한 템플릿 특수화는 다음과 같다.

```
template <>
const Box& Max<Box>(const Box& a, const Box& b)
{
  return a.volume>b.volume?a:b;
}
```

4.
```
void showBox(const Box& box)
{
  cout << "Maker = " << box.maker << endl;
  cout << "Height = " << box.height << endl;
  cout << "Width = " << box.width << endl;
  cout << "Length = " << box.length << endl;
  cout << "Volume = " << box.volume << endl;
}

void setVolume(Box& box)
{
  box.volume = box.height*box.width*box.length;
}
```

5. (a) 두번째 전달인자에 default 값을 매겨서 처리할 수 있다.

```
double mass(double density, double volume=1.0);
```

또는 함수 오버로딩으로도 처리할 수 있다.

```
double mass(double density, double volume);
double mass(double density);
```

(b) 디폴트 값은 오른쪽에서 왼쪽으로 제공하므로 사용할 수 없으며, 함수 오버로딩을 다음과 같이 사용할 수 있다.

```
void repeat(int count, const char * str);
void repeat(const char * str);
```

(c) 함수 오버로딩을 사용할 수 있다.

```
int average(int a, int b);
double average(double a, double b);
```

(d) 두 함수가 같은 시그너춰를 사용하므로, 함수 오버로딩은 불가능하다.

6. `int` 형의 배열의 첫 번째 원소를 지시하는 포인터, 그 배열의 끝 바로 다음 원소를 지시하는 포인터, 하나의 `int` 형 값을 세 전달 인자로 취하여, 그 배열의 모든 원소를 세 번째 전달 인자인 `int` 형 값으로 설정하는 함수는 다음과 같다.

```
void setArray(int* startArr, int* endArr,int val)
{
  for (int* p = startArr ; p<endArr ; p++) *p = val;
}
```

`double` 형의 배열의 이름과 배열의 크기를 두 개의 전달 인자들로 취하여, 그 배열에서 가장 큰 값을 리턴하는 함수는 다음과 같다.

```
double biggerOne(const double arr[],int size)
{
  if (size<=0)
```

```
  {
    std::cerr << "Wrong array size: " << size << endl;
    throw size;
  }
  else
  {
    double bigger = arr[0];
    for (int i=1 ; i<size ; i++) if (bigger<arr[i]) bigger = arr[i];
    return bigger;
  }
}
```

7. (a) `void igor(void);`
   (b) `float tofu(int);`
   (c) `double mpg(double, double);`
   (d) `long summation(long[], int);`
   (e) `double doctor(const char*);`
   (f) `void ofcourse(Boss);`
   (g) `char* plot(Map*);`

8. 프로그램의 if 문에서 `(ch == '$')` 라고 입력하려고 했으나 `(ch = '$')` 라고 잘못 입력되었다. 이런 경우 `(ch = '$')` 문은 `$` 로 평가되어, if 문의 결과는 항상 참이 된다. 따라서 `ct2` 에서 `$` 의 개수만 세려고 했던 원래의 의도와는 달리, `ct1` 과 `ct2` 는 같이 증가한다. 결국 다음과 같이 출력된다.

```
Hi!<ENTER>
H$i$!$
$Send $10 or $20 now!<ENTER>
S$e$n$d$ $ct1 = 9, ct2 = 9
```

9. 다음의 명령문은 적합하다.

```
int x = 0;
x = (1,024);
```

   괄호 안에 든 표현식 `(1,024)` 은 콤마 연산자에 의해 처음에는 1로 평가된 후 그 다음에는 024로 평가된다. 즉 최종적으로 024로 평가된다.

   그런데, C/C++에서는 숫자가 0으로 시작하고 각각의 숫자들이 8보다 작으면 8진수로 간주한다. 따라서, 024 는 십진수 24가 아니라 8진수 24로 간주되고, 그 값은 10진수로는 20이 된다. 따라서 위의 명령문은 x에 20을 대입한다.

   다음의 명령문도 적법하다.

```
int y;
y = 1,024;
```

이번 경우, 괄호가 없으므로 우선 순위가 낮은 콤마 연산자보다 `y = 1` 이 먼저 평가된다. 따라서 y 는 1 이 되고, `y = 1` 전체는 1로 평가된다. 그리고 나서 콤마 연산자에 의해 024 가 십진수 20으로 평가가 되지만, 이미 `y = 1` 이 먼저 수행되어 따로 대입할 변수가 없으므로 그냥 버려진다.

10. `cin >> ch` 는 화이트 스페이스, 즉 빈칸 문자, 개행 문자, 캐리지 리턴, 수평 탭, 수직 탭, 한 페이지를 넘길 때 쓰는 폼 피드(form feed) 문자를 그냥 건너 뛴다. `ch = cin.get()` 이나 `cin.get(ch)` 는 그 문자들도 읽어들인다.

11. (a) `char c=88; cout << c << endl;`
    (b) `cout.put(char(88));`
    (c) `cout << char(88) << endl;`
    (d) `cout << (char)88 << endl;`

12. ```
    cout << "Enter a number :";
    unsigned int num = 0;
    cin >> num;
    int* arr = new int [num];
    ```

13. `for (int num=1 ; num<=64 ; num*=2) cout << num << " ";`

14. ```
    Fish* newFish = new Fish;
    cout << "Enter kind :";
    cin.getline(newFish->kind, 20);
    ```

15. ```
    double* pf = treacle;
    cout << pf[0] << " and " << pf[9] << endl;
    ```
    ```