| CODING TEST SOLUTIONS |
| --- |
| An examiner : Son Hyegang<br>The copyright on the problem of the coding test belongs to the problem writer. Do not allow external leakage. The list of references is excluded. |

1. 197 은 circular prime 이라고 불리는데, 그 이유는 이 숫자가 prime 이면서 이것을 rotate 한 값, 즉 971, 719 도 역시 prime 이기 때문이다. 어떤 수가 circular prime 인지 확인할 수 있도록 다음 함수를 완성하시오. 필요한 경우 추가로 함수를 정의할 수 있다.

197 is called a circular prime because the number is a prime and its rotate values, i.e., 971 and 719 are also prime numbers. Complete the following function to test whether a given number is a circular prime. You can define extra functions if needed.

/* define extra functions here if needed */

```
int is_prime(int n){
        int i;
        for(i=2;i<n/2+1;i++){
                if(n%i==0)      //if(!(n%i))
                        return 0;
        }
        return 1;
}
```

int is_circular_prime(int n) {

```
        int num=n,k=0,final=num%10;
        while(num){
                k++;
                num/=10;
        }
        int rotate=n/10+pow(10,k-1)*final;
        while(rotate!=n){
                printf("rotate : %d₩n",rotate);
                if(!is_prime(rotate))
                        return 0;
                num=rotate;
                final=num%10;
                rotate=num/10+pow(10,k-1)*final;
        }
        return 1;
```

}

```
int main() {
        int num;
        printf("Enter number: ");
        scanf("%d", &num);
        if (is_circular_prime(num))
                printf("%d is a circular prime₩n", num);
        else
                printf("%d is not circular prime₩n", num);
}
```

2. ‘소수’ 는 1 보다 크고 1 과 자기 자신만으로 나누어지는 수이다. 예를 들어 5, 11, 23 등이 여기에 속한다. 양의 정수가 주어져 있을 때 이 숫자가 소수인지 혹은 소수에 얼마나 가까운 지를 알려주는 프로그램을 작성하시오.

A “prime” number is an integer greater than 1 with only two divisors: 1 and itself; examples include 5, 11 and 23. Given a positive integer, you are to print a message indicating whether the number is a prime or how close it is to a prime.

다음은 위의 프로그램을 수행한 결과를 보여준다.
The following shows an execution of the above program.

```
$ ./a.out
Enter number: 37
Would you believe it; 37 is a prime!
$ ./a.out
Enter number: 33
33 is not prime; you missed it by that much (2)!
```

```
#include <stdio.h>
int is_prime(int num) {
        int i;
        for(i=2;i<num/2+1;i++){
                if(!(num%i))
                        return 0;
        }
        return 1;
}
int closest_prime(int n) {
        if (is_prime(n))
                return 0;
        else {
                int next_prime, prev_prime;
                next_prime = 0;
                prev_prime = 0;
                for(int i=1;!next_prime;i++){
                        if(is_prime(n+i))
                                next_prime=n+i;
                }
                for(int i=1;!prev_prime;i++){
                        if(is_prime(n-i))
                                prev_prime=n-i;
                }
                return next_prime-n<n-prev_prime?next_prime-n:n-prev_prime;
        }
}
void main(void)
{
        int n = 37;
        int p;
        printf("Enter number: ");
        scanf("%d", &n);
        p = closest_prime(n);
        if (p == 0)
                printf("Would you believe it; %d is a prime!₩n", n);
        else
                printf("%d is not prime; you missed it by that much (%d)!₩n₩n", n, p);
}
```

3. 일반적인 주사위는 1 에서 6 의 값을 출력한다. 반면에 int dice100() 은 이를 확장하여 1 에서 100 사이의 값을 출력한다. 이 주사위를 던져서 30 개의 고유한 숫자를 생성하도록 다음 프로그램을 완성하시오.

A dice returns a value between 1 and 6. On the contrary, function int dice100() randomly generates a value between 1 and 100. Complete the following program so that it would fill the array with 30 numbers unique. It is important that the numbers are not duplicated.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int dice100() {
        return rand()%100 + 1;
}
int unique(int x, int output[], int n) {
        int i;
          for(i=0;i<n;i++){
                  if(x==output[i])
                          return 0;
          }
        return 1;
}
int main() {
        int i, j, p=0;
        int output[30];
        for(i = 0; i < 30; i++) {
                int x = dice100();
        while(!unique(x,output,i))
                x = dice100();
                output[p++] = x;
        }
        for(i = 0; i < 30; i++) {
                printf("%d₩n", output[i]);
        }
        return 0;
}
```

4. 다음과 같이 주어진 문자열에서 숫자만 골라 합산하는 함수 sum_digit()을 작성하시오. 예를 들어 다음 문자열에서 "Korea University, 2017" 숫자는 '2','0','1','7' 이다. 이들을 정수로 변환한 뒤 모두 더하면 10 이 된다. 다른 예로 "Pyeong Chang 2018 Olympic" 에서는 '2','0','1','8' 이 숫자이고 이들을 변환한 후에 모두 더하면 11 이 된다.

Write the function sum_digit() which converts and adds all of the numeric characters in a string. For example, "Korea University, 2017"has four numeric characters '2','0','1', and '7'. Adding them up after converting each character into the corresponding integer yields 10. As another example, "Pyeong Chang 2018 Olympic" contains the numeric characters of '2','0','1', and '8' and its sum is 11.

| Hint: 문자 '0' 의 ASCII code 는 48 이다. 그러나 이 사실을 몰라도 이 C 함수를 완성할 수는 있다. ASCII code of the character '0' is 48. However, you could write the C function without having to know this value. |
|---|

```
int sum_digit(char str[]) {
        int i;
        int sum = 0;
          for(i=0;str[i];i++){
                  if(str[i]>='0'&&str[i]<='9')
                          sum+=str[i]-48;
          }
        return sum;
}
```

5. “pair isogram” 이라고 불리는 단어는 단어를 구성하는 각각의 문자가 더도 덜도 아니고 정확하게 두 번씩 나타난다. 예를 들어 ‘teammate’ 와 같은 단어가 pair isogram 인데 여기서 각각의 문자가 더도 덜도 아니고 정확하게 두번씩 나타나기 때문이다. 그러나 ‘dad’의 경우는 pair isogram 이 아닌데, 그 이유는 문자 ‘a’ 가 두 번 나오지 않기 때문이다. 단어가 주어져있을 때, 그 단어가 pair isogram 인지를 판별하는 프로그램을 작성하시오. 단 단어는 영문자로 구성되어 있고 모두 소문자로 가정한다.

A word is considered to be “pair isogram” if each letter in the word appears exactly twice (not less, not more) in the word. For example, the word teammate is pair isograms since each letter in the word appears exactly twice (not less, not more), but the word dad is not since the letter “a” of the world doesn’t appear twice. Given a word, you are to determine wheter or not it is pair isograms. Note that all characters of the word is lower case alphabets.

다음은 위의 프로그램을 수행한 결과를 보여준다.
The following shows an execution of the above program.

```
$ ./a.out
Enter word: teammate
teammate is a pair isogram
$ ./a.out
Enter word: dad
dad is NOT a pair isogram
```

```
#include <stdio.h>
int is_pair_isogram(char str[]) {
        int count[26]={0};
          for(int i=0;str[i];i++){
                    count[str[i]-'a']++;
          }
          for(int i=0;i<26;i++){
                    if(count[i]%2)
                              return 0;
          }
        return 1;
}
void main()
{
        char str[100];
        printf("Enter word: ");
        scanf("%s", str);
        if (is_pair_isogram(str))
                printf("%s is a pair isogram₩n", str);
        else
                printf("%s is N a pair isogram₩n", str);
}
```

6. 보안상 특별한 방에 입실할 때 id 를 입력하면 id 별로 입실 횟수를 기록하는 시스템을 개발하였다. 이 시스템은 예를 들어 최근 6 회의 입실기록을 다음과 같이 보여준다. 여기서 -1 은 입실기록의 끝을 나타낸다.

We developed a system that stores the entry history to a room per user ID for security reasons. The system prints the number of entries of each user in an increasing order in terms of user ID. The following output shows an example of the program for the recent six entries, where -1indicates the end of history.

```
$ ./a.out
Enter id: 6927238
Enter id: 3428529
Enter id: 2800555
Enter id: 2900123
Enter id: 3428529
Enter id: 6927238
Enter id: -1
id: 2800555 (1)
id: 2900123 (1)
id: 3428529 (2)
id: 6927238 (2)
```

이 프로그램은 다음 그림 1 과 같이 linked list 구조를 이용하여 출입을 관리하고, linked list는 id 를 등록할 때 항상 정렬된 상태를 유지한다.
This program manages the data using the following linked list structure as shown in Figure 1.
The order of data in the linked list remains sorted when we register an id.

```
#include<stdio.h>
#include<stdlib.h>
struct Node {
        int id;
        int count;
        struct Node* next;
};
typedef struct Node* NodePtr;
NodePtr head=NULL;
void print_report() {
        /*
        for(NodePtr p = head; p != NULL; p = p->next) {
                printf("id: %d (%d)₩n", p->id, p->count);
        }
        */
        struct Node *temp=head;
        while(temp!=NULL){
                printf("id: %d (%d)₩n", temp->id, temp->count);
                temp=temp->next;
        }
}
void main() {
        int id;
        for(;;) {
                printf("Enter id: ");
                scanf("%d", &id);
                if (id < 0) {
                        break;
        }
                record_entry(id);
        }
        print_report();
}
```

예제와 같은 결과를 보이도록 다음 함수를 완성하시오.
Complete the following function to produce the output as shown in the example.

```
void record_entry(int id) {
    struct Node *temp=head;
    struct Node *prev=NULL;
    struct Node *ptr;

    ptr=(struct Node*)malloc(sizeof(struct Node));
```

```
        ptr->id=id;
        ptr->count=1;
        ptr->next=NULL;

        if(temp==NULL){
            //Executes when linked list is empty
            ptr->next=NULL;
            head=ptr;
            return;
        }

        if(id<temp->id){
            //Executes if given id is less than id in first Node of linked list
            ptr->next=head;
            head=ptr;
            return;
        }
        else{
            while(temp!=NULL){
                                if(id==temp->id){
                                        temp->count++;
                                        return;
                                }
                    if(id>temp->id){
                    //Traverse to location we want to insert the Node + 1 Node
                    prev=temp;
                    temp=temp->next;
                    continue;
                }
                else{
                    //Insert the Node
                    prev->next=ptr;
                    ptr->next=temp;

                                return;
                }
            }
            prev->next=ptr;
            //Insert Node at last
        }
}
```

7. 영문자로만 이루어져 있고 반복된 문자가 많이 나오는 문자열의 경우 문자의 반복횟수를 표시하여 문자열의 길이를 줄일 수 있다. 예를 들어 다음과 같이 변환할 수 있다.

We can compress strings composed of alphabet characters as follows.

- "zzzzzzzz"→ "8z"
- "bbbbcceeeee"→ "4b2c5e"

한 문자의 반복 횟수는 최대 9 회이고, 문자가 반복되지 않는 경우는 반복횟수를 표시하지 않는다. 또 이 알고리즘은 대문자와 소문자를 구분하고, 공백 문자열은 그대로 둔다.

We assume that the maximum number of repetitions of the same alphabet is 9. If different characters are consecutive, it is not compressed. Empty string is compressed as an empty string. The program is case-sensitive.

- "abcde"→ "abcde"
- "aaaaaAAAA"→ prints "5a4A"
- ""→ ""

(a) 주어진 문자열을 압축하는 함수 strcompress 를 작성하시오.
Write strcompress function that is supposed to compress given strings.

```
void strcompress(char* input, char* output) {
        char c = input[0];
        int cnt = 1;
        int p = 0;
        for(int i = 1; input[i]; i++) {
                if (input[i] == c)
                cnt++;
                else if (cnt == 1) {
                        output[p++] = c;
                        c = input[i];
                }
                else {
                        output[p++] = cnt + '0';
                        output[p++] = c;
                        c = input[i];
                        cnt = 1;
                }
        }
        if (cnt == 1)
                output[p++] = c;
        else {
                output[p++] = cnt + '0';
                output[p++] = c;
        }
        output[p] = 0;
}
int main() {
        char str[] = "zzzzzzzzaaabbcdddeee";
        char comp[BUFSIZ];
        strcompress(str, comp);
        printf("Compress: %s₩n", comp);
        /* Compress: 8z3a2bc3d3e */
}
```

(b) 압축된 문자열을 복원하는 함수 strdecompress 를 작성하시오. Write the strdecompress function that is supposed to decompress given strings.

```
void strdecompress(char* input, char* output) {
        int p = 0;
        int cnt = 1;
        for(int i = 0; input[i]; i++) {
```

```
                char c = input[i];
                if (c >= '0' && c <= '9')
                        cnt = c - '0';
                else {
                        for(int j = 0; j < cnt; j++)
                                output[p++] = c;
                        cnt = 1;
                }
        }
        output[0] = 0;
}
int main() {
        char comp[] = "8z3a2bc3d3e";
        char decomp[BUFSIZ];
        strdecompress(comp, decomp);
        printf("Decompress: %s₩n", decomp);
        /* Decompress: zzzzzzzzaaabbcdddeee */
}
```

8. 컴퓨터 프로그램의 버전 정보는 일반적으로 다음과 같은 형태를 갖는다
A computer program has a version information denoted as follows.
1.0.1

아주 작은 변화가 있다면 마지막 숫자를 하나 늘이고 상대적으로 큰 변화가 있을 때는 중간 숫자를 하나 늘인다. 그리고 설계상의 중대한 변화가 있을 때는 가장 앞자리 숫자를 하나 늘인다. 경우에 따라서는 다음과 같이 마지막 숫자를 표시하지 않는 경우도 있다.
For trivial changes, the last part of the version counter is incremented. For intermediate changes, the number in the middle is updated. The first number is updated when significant changes are made in design. For some cases, version number may have only two parts:
2.0

두 개의 version 정보가 주어져있을 때, 함수 compare_version() 는 두 개의 version 정 보를 비교하여 왼쪽이 더 최신 정보인 경우 양의정수를, 반대인 경우는 음의 정수를, 그렇지 않은 경우는 0 을 return 한다. 예를 들어 다음과 같다.
Given two strings representing version information, the function compare_version() compares them and returns a positive integer if the first argument is more recent. In the opposite case, i.e., the second argument is more recent, it returns a negative number. If the version informations are the same, it returns zero. The following are some examples.
compare_version("0.0.2", "0.0.2") == 0
compare_version("1.0.3", "1.0.10") < 0
compare_version("1.2.0", "1.1.99") > 0
compare_version("2.3", "3.1.6") < 0

다음 함수를 완성하시오.
Complete the following function.

```
int compare_version(char* left, char* right) {
        char* p = left;
        char* q = right;
        int x = 0;
        int y = 0;
        while (*p && *q) {
                for(; *p != '.' && *p != '₩0'; p++) {
                        x = x*10 + *p - '0';
                }
                for(; *q != '.' && *q != '₩0'; q++) {
                        y = y*10 + *q - '0';
                }
                if (*p == '₩0' && *q == '₩0') {
                        break;
                }
                else if (*p == '.' && *q == '.') {
                        if (x == y) {
                                p++, q++;
                                x = y = 0;
                        }
                        else {
                                return x - y;
                        }
                }
        }
        return x - y;
}
```

9. 뒤집어서 더하기 연산은 어떤 양의 정수가 주어졌을 때 그 수를 뒤집어서 더한 값을 출력하는 것이다. C 언어로 구현한 함수 flipadd() 는 이 연산을 수행하며 다음과 같은 결과를 출력한다.
The operator flipadd adds a given positive integer and its flipped value as shown below:
flipadd(123) = 123 + 321 = 444
flipadd(195) = 195 + 591 = 786

이 함수를 몇 차례 반복 적용하면 앞으로 읽으나 뒤로 읽으나 같은 값이 되는 회문수 (palindrome number) 가 된다. 예를 들어 195를 시작으로 flipadd() 를 몇 차례 적용하면 회문 수인 9339 가 출력된다.
With repeated application, a number often eventually becomes a palindrome number. For example, applying flipadd() to the number 195 becomes 9339, which is a palindrome number as follows:
flipadd(195) = 195 + 591 = 786
flipadd(786) = 786 + 687 = 1473
flipadd(1473) = 1473 + 3741 = 5214
flipadd(5214) = 5214 + 4125 = 9339

대부분의 정수는 이 단계를 몇 단계만 반복하면 회문 수가 된다. 그렇지만 어떤 수는 1,000번 정도 반복해도 회문수가 되지 않는다. n개의 숫자를 입력으로 받아 입력 값 그 숫자가 회문 수가 되는데 소요되는 반복횟수, 그리고 마지막으로 계산된 회문 수를 출력하는 프로그램을 작성하시오. 입력파일의 첫 줄은 총 몇 개의 숫자가 있는지를 나타낸다. 만일 1,000 회 적용했는데도 회문이 되지 않는 경우 반복횟수에 1001 을 출력한다. 마지막 필드의 값은 그 때 까지 계산한 flipadd 의 값이 된다.
Most integers becomes a palindrome number after applying the function several times. On the contrary, there are numbers which do not become a palindrome number with even more than 1,000 iterations. Write a program which outputs the initial value (e.g., input), number of iterations necessary for it to become a palindrome number, and the palindrome number. The first line states how many numbers are to be tested. If an input value does not yield a palindrome value after more than 1,000 iterations, the output should be 1001 and the value computed at the final iteration.

Sample input
4
195
265
196
750

Output
195 4 9339
265 5 45254
196 1001 -1403241137
750 3 6666

```
#include <stdio.h>
/* flipadd and other functions are defined here */
int flip(int n) {
        int rev = 0;
        while(n > 0) {
                rev = rev*10 + n%10;
                n = n / 10;
        }
        return rev;
}
int is_palin(int n) {
        return n == flip(n);
}
int flipadd(int n) {
        return n + flip(n);
}
int solve(int n, int* count) {
        int i;
        for(i = 0; i < 1000; i++) {
                n = flipadd(n);
                if (is_palin(n)) {
                        *count = i + 1;
                        return n;
                }
        }
        *count = 1001;
        return n;
}
void main() {
        FILE* fp;
        int num;
        for(int i = 0; i < num; i++) {
                int n, res;
                int count;
                fscanf(fp, "%d", &n);
                res = solve(n, &count);
                printf("%d %d %d₩n", n, count, res);
        }
}
```