**Introduction to Computer and Lab**

**Homework #5**

**Due date: Mar 12, 2016**

**학번: 201404051**

**이름: 정 용 석**

**1. 369게임**

**1.1 Solution**

사용자에게 정수를 입력 받고, 그 수가 3,6,9 중 하나이면 박수를 치고, 5의 배수일 경우 만세를 하는 프로그램으로, 구현 또한 간단하다. 사용자에게 받은 정수를 저장할 변수와 이 수의 각 자리를 확인하기 위한 복사 변수를 만들어주고, 박수 횟수와 만세 여부를 확인할 변수를 초기화 해준다. 프로그램의 지속성을 위해 무한반복문안에서 구현을 실시하였다. 가장 처음 정수를 입력 받고, 프로그램 종료 여부를 제일 먼저 확인한다. 그 이후, 각 자리의 수를 확인하여 박수 횟수를 확인하고, 5의 배수 여부 또한 확인하고, 확인된 숫자들을 출력하고, 사용했던 변수들을 초기화해준다.

**1.2. Source code**

void problem7() //사용자에게 정수를 입력 받고, 박수의 개수와 만세 여부를 확인, 출력

{

int N;

int temp;

int clapCount = 0; //박수 칠 횟수

int hooray = 0; //만세 여부

while (1)

{

printf("Input the number: ");

fflush(stdout);

scanf("%d", &N);

if (N > 1000000) {//1000000이상일시 프로그램종료

printf("Out of range. Closing program\n");

break;

}

temp = N;

while (temp != 0)

{

//정수의 자릿수가 3,6,9일 시 박수 횟수 +1

if (temp % 10 == 3 || temp % 10 == 6 || temp % 10 == 9)

clapCount++;

temp = temp / 10;

}

if (N % 5 == 0) //정수가 5의 배수일 경우 만세

hooray = 1;

printf("clap: %d\n", clapCount);

if (hooray == 0)

printf("no hooray\n");

else

printf("hooray!\n");

//박수 횟수 만세 여부 초기화

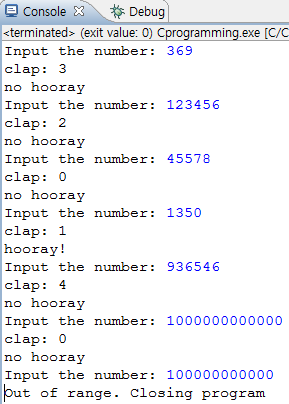
hooray = 0;

clapCount = 0;

}

}

**1.3. Result (snapshot)**



**2 거꾸로 읽은 수의 합**

**2.1. Solution**

이 문제의 핵심은 사용자에게 입력 받은 수를 역순으로 만드는 것이다. 다양한 방법이 있겠지만, 필자는 입력 받은 수의 마지막 자리부터 한 개씩 역순을 구하는 방식을 선택했다. 역순이라고 하면, 마지막 자리가 가장 앞자리가 되야 함으로, 맨 처음 수를 구한 이후로, 10을 곱하면서 자리 수를 1개씩 올려주는 방식이 되겠다. 전 문제와 동일하게 무한 반복 문을 이용하였고, 맨 처음 수를 입력 받음과 동시에 프로그램 종료 여부 확인, 역수를 구하고, 입력 받은 수와의 합을 계산, 출력 그리고 마지막으로 사용된 변수들을 다시 초기화 하는 방법을 사용하였다.

**2.2. Source code**

**void** **problem8**() //사용자에게 정수를 입력 받고, N을 반대로 한 수와 합한 결과를 출력

{

**int** N;

**int** temp;

**int** reverse = 0; //역수

**int** sum = 0; //합

**while**(1){

**printf**("Input the number: ");

**fflush**(stdout);

**scanf**("%d", &N);

**if**(N <= 0 || N >= 1000000) //프로그램 종료 여부

{

**printf**("Invalid Input Program Closes\n");

**break**;

}

temp = N;

**while** (temp != 0)

{

//가장 마지막 자리부터 10의 자리를 늘려가며 역순을 구한다.

reverse = reverse \* 10 + (temp % 10);

temp = temp / 10;

}

sum = N + reverse;

**printf**("The sum of two numbers: %d\n", sum);

//사용된 변수들 초기화

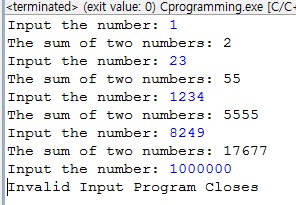
sum = 0;

reverse = 0;

}

}

**2.3. Result (snapshot)**



**3 10진법 변환**

**3.1. Solution**

이번 문제는 정수의 각 자리를 구하는 방법과 이진법에서 n번 자리의 수를 구하는 방법이 중요하다. 일단, 조건 사항 중 하나인 16자리 이진수를 위하여 입력 받을 정수와 이를 복사 저장할 변수들을 모두 unsigned long long 자료 형으로 만들어 주었다. 그리고 중요한 변수 중 하나인 add와 count 변수는 이진 수의 자리 수를 계산해 줄 것이다. 마찬가지로 무한 반복 문을 이용하였고, 시작부터 프로그램 종료 여부를 확인한다. 각 자리 수를 구함과 동시에 그 자리의 값을 계산하도록 하였는데, 일단 자리 수는 어차피 1혹은 0이기 때문에 몇 번째 자리에 있느냐가 중요하다. 반복 문은 가장 마지막 자리 수부터 구하기 때문에 마지막 자리는 2에 0제곱으로 무조건 1이기 때문에, 반복문에서도 제외 시켰다. 그 다음 자리부터 add가 n번 만큼 2를 곱하게 되는데, 이는 2의 제곱을 표현한 것이다. for문을 통과하면 decimal 변수에 그 수를 더해주는 방법으로 10진수를 구하게 된다. 전 문제들과 마찬가지로 사용된 변수들은 재사용을 위해 마지막에 초기화 해주어야한다.

**3.2. Source code**

**void** **problem9**() //사용자가 입력한 이진수를 십진수로 변환

{

**unsigned** **long** **long** N;

**unsigned** **long** **long** temp;

**int** decimal = 0; //10진법 저장 변수

**int** add = 1; //2진법 -> 10진법 변환 시 Nth 자리의 값

**int** count = 0; //N제곱승을 표현할 변수

**int** i;

**while**(1){

**printf**("Input the number: ");

**fflush**(stdout);

**scanf**("%llu", &N);

temp = N; //입력 받은 수 복사, 임시 저장

//프로그램 종료 여부 확인1. N == 0일 시 종료

**if** (N == 0) {

**printf**("No binary. Program Closes\n");

**exit**(1);

}

**while** (temp != 0)

{

//프로그램 종료 여부 확인2. 이진수가 아닐 경우

**if** (temp % 10 > 1) {//N자리의 수가 2이상일 시 이진법이 X

**printf**("No binary. Program Closes\n");

**exit**(1);

}

**else**

{

**if** (temp % 10 == 1){//자리 수가 1일 경우 2^count를 decimal에 합

**for** (i = 0; i < count; i++)

add \*= 2;

decimal += add;

}

**else** { //자리 수가 0일 경우, 0;

add = 0;

decimal += add;

}

}

//다음 자리 수를 위한 연산

add = 1;

count++; //자리수는 반복문이 한 번 돌때 마다 증가한다.

temp = temp / 10;

}

**printf**("Decimal: %d\n", decimal);

//프로그램 재사용을 위해 변수들 다시 초기화

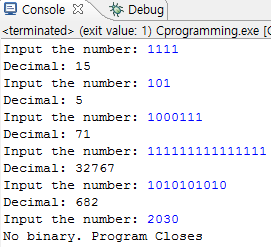
decimal = 0;

count = 0;

}

}

**3.3. Result (snapshot)**



**4. Palindrome 체크**

**4.1. Solution**

위의 문제 중 역순을 구하는 식과 동일하며, 역순을 구하여 입력 받은 수와 동일하면 palindrome이다.

**4.2. Source code**

**void** **problem10**() //사용자에게 N을 입력받고 Palindrome 확인

{

**int** N;

**int** reverse = 0;

**int** temp;

**while**(1){

**printf**("input the number: ");

**fflush**(stdout);

**scanf**("%d", &N);

//프로그램 종료 여부 확인

**if**(N < 0 || N > 1000000)

{

**printf**("Invalid Input. Program Closes\n");

**break**;

}

temp = N;

**while** (temp != 0)

{

//입력받은 수 N의 역순

reverse = reverse \* 10 + temp % 10;

temp = temp / 10;

}

**if** (reverse == N) //역순 reverse가 N과 같다면 Palindrome

**printf**("Palindrome\n");

**else**

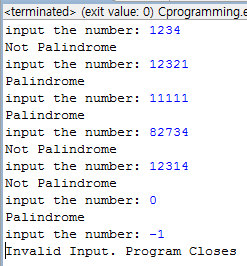
**printf**("Not Palindrome\n");

reverse = 0;

}

}

**4.3. Result (snapshot)**



**5 PC방 이용 요금 계산**

**5.1. Solution**

이 문제는 사용자에게 시간과 분을 따로 입력 받기 때문에 나는 일단 총 사용 시간을 분으로 표현하여 계산하였다. 이용 요금이 30분 당이기 때문에 그래야 계산하기 편하다. 시와 분에 대한 제약 여부는 따로 넣지 않았지만, 종료 시간이 시작 시간보다 빠르거나, 이용 시간이 0원이나 음수에 대한 프로그램 종료 문은 작성하였고, 총 이용 요금 계산 시, 사용자가 30분 이하로 이용하였을 경우 30분 이용요금으로 산정해야 하기 때문에, 이를 위하여 총 이용 시간이 30으로 나누어 떨어지지 않으면, 총 이용시간에 30을 더하고 나누었다. 이렇게 하면 1분~29분 이용한 시간을 모두 동일하게 산정할 수 있다.

**5.2. Source code**

**void** **problem11**() //사용자가 PC방 이용 시작 시간과 종료 시간, 30분당 이용 요금 입력했을 때, 시간에 대한 요금 출력

{

**int** startHour, startMin; //시작 시간, 분

**int** endHour, endMin; //종료 시간, 분

**int** totalT = 0; //총 사용 시간

**int** price; //30분당 이용 요금

**int** totalP = 0; //총 요금

**while**(1){

**printf**("Input the starting time<hour mins>: ");

**fflush**(stdout);

**scanf**("%d %d", &startHour, &startMin);

**printf**("Input the end time<hour mins>: ");

**fflush**(stdout);

**scanf**("%d %d", &endHour, &endMin);

**printf**("Price per 30 mins<won>: ");

**fflush**(stdout);

**scanf**("%d", &price);

//총 이용 시간을 분으로 계산

totalT = (60 \* endHour + endMin) - (60 \* startHour + startMin);

//프로그램 종료 여부: 요금, 시간이 0이하

**if** (totalT <= 0 || price <= 0){

**printf**("Invalid Input. Program Closes.\n");

**break**;

}

//총 시간(분)을 30분 단위로 나누어 총 이용시간을 계산한다.

// \*주의점: 30분 이하로 사용했을 시에는 30분 이용요금 선정을 위해 총 이용시간에 30분을 더하여 게산한다.

**if** (totalT % 30 == 0)

totalP = price\*(totalT / 30);

**else**

totalP = price\*((totalT+30) / 30);

**printf**("Total time spent: %d hour %d mins <%d mins>\n", totalT / 60, totalT % 60, totalT);

**printf**("Total price: %d\n\n", totalP);

//변수 다시 초기화

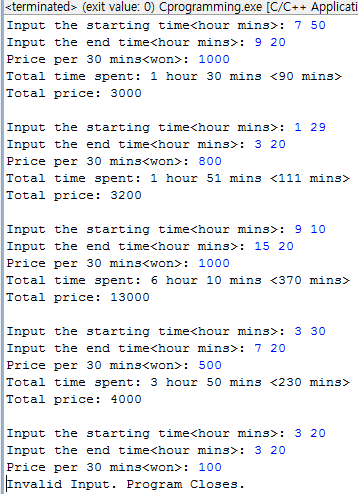
totalT = 0;

totalP = 0;

}

}

**5.3. Result (snapshot)**



**6. 해당 년도 달력 출력**

**6.1 Solution**

여러가지 방법을 시도해 봤지만, 현재 방법이 가장 쉽다고 생각한다. 가장 첫 번째로 입력 받은 해의 시작 요일을 찾는 것이다. 문제 설명에도 나와있듯이, 1월 1일은 월요일이고 매년 1일씩 요일이 밀려나고, 윤년이면 2일이 밀려난다. 일주일은 7일이기 때문에 시작 요일을 계산할 때, 서기 1년부터 입력 받은 년까지의 시작 요일을 계산할 때, 단순하게 더하고 맨 마지막에 mod를 이용하여 7을 나눈 나머지를 계산하면 그 해의 시작 요일을 구할 수 있다. 실제 구현단계에서도 문제가 되었던 것은 계산을 할 때, 올 해가 윤년인 것은 중요하지 않다는 것이다. 작년이 윤년 이어야 올 해의 시작 요일이 2일이 밀리는 것이기 때문이다. 이것만 주의하면 쉽게 구할 수 있다. 물론 정수로 표현되어있기 때문에 필자는 1을 월요일로 해서 7을 일요일까지 로 표현하기로 했다.

다음으로는 윤년 확인 여부를 넣어두었다. 윤년 확인 문 자체가 길고, 한 번만 구해 놓으면 되기 때문에, 변수를 따로 두어서 윤년이면 1, 아니면 0이 되게 해놓았다. 그리고 다음 계산하여야 하는 것이 바로 입력 받은 달의 시작 요일이다. 이미 그 해의 1월의 시작 요일을 알기 때문에, 마찬가지로 각 달의 요일 수를 합하여 이번 달의 시작 요일을 계산할 수 있다. 각 달을 직접 적어서 더하려고 하다 너무 헷갈려서 switch문을 통하여 보기 쉽게 각 달의 요일을 표시함과 동시에 다음 달의 시작 요일을 계산하였다. 마찬가지로 7로 나눈 나머지 값이 시작 요일이 되겠다. 마지막으로 지금까지 계산한 시작 요일을 바탕으로 달력을 출력해주면 된다.

**6.2 Source code**

**void** **problem12**() //사용자가 서기년도와 월을 입력 시, 해당 년도 원의 달력 출력

{

**int** year, month; //년, 월

**int** startDay = 0 , endDay; //시작 요일, 각 달의 요일 수

**int** lunarYear = 0; //윤년 여부

**int** i = 1;

**int** date = 1;

**printf**("Input the year and month: ");

**fflush**(stdout);

**scanf**("%d %d", &year, &month);

//그 해의 시작 날 계산

**for** (i = 1; i <= year; i++) //서기 1년부터 입력 받은 해까지 윤년과 아닌 해르 구분하여 시작날을 더한다.

{

//\*\*(ㅑ-1)을 한 이유: 작년이 윤년이여만 올해 시작 요일 2일 뒤로 밀린다.

**if** (((i-1) % 4 == 0 && (i-1) % 100 != 0) || ((i-1) % 400 == 0 && (i -1) != 0))

startDay += 2;

**else**

startDay += 1;

}

//일주일은 7일 임으로, 시작 요일 또한 지금 해까지의 모든 시작 요일을 계산한 값을 7로 나눈 나머지 값

startDay = startDay % 7;

//이번 년이 윤년이면 1, 아니면 0

**if** ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0)

lunarYear = 1;

**else**

lunarYear = 0;

//각 달의 시작 날 확인 및 날수 확인 및 해당 달의 시작 요일 계산

**for** (i = 2; i <= month; i++)

{

**switch** (i-1) {

//4, 6, 9, 11월은 30일

**case** 4:

**case** 6:

**case** 9:

**case** 11:

endDay = 30;

**break**;

**case** 2:

//윤년인 년의 2월은 29일, 아니면 28일

**if** (lunarYear)

endDay = 29;

**else**

endDay = 28;

**break**;

//1, 3, 5, 7, 8, 10, 12월은 31일

**case** 1:

**case** 3:

**case** 5:

**case** 7:

**case** 8:

**case** 10:

**case** 12:

endDay = 31;

**break**;

}

//이번 달의 시작 요일은 1월부터 지금까지의 일수를 7로 나눈 나머지

startDay += endDay;

}

startDay = startDay % 7;

//입력받은 달의 요일 수 체크

**switch** (month) {

**case** 4:

**case** 6:

**case** 9:

**case** 11:

endDay = 30;

**break**;

**case** 2:

**if** (lunarYear)

endDay = 29;

**else**

endDay = 28;

**break**;

**case** 1:

**case** 3:

**case** 5:

**case** 7:

**case** 8:

**case** 10:

**case** 12:

endDay = 31;

**break**;

}

//달력 출력

**printf**("Sun\tMon\tTue\tWed\tThu\tFri\tSat\n");

**for** (i = 1; i <= startDay; i++)

{

**printf**("\t");

}

**for** (i; i <= (endDay + startDay); i++)

{

**printf**("%d\t", date);

**if** (i % 7 == 0)

**printf**("\n");

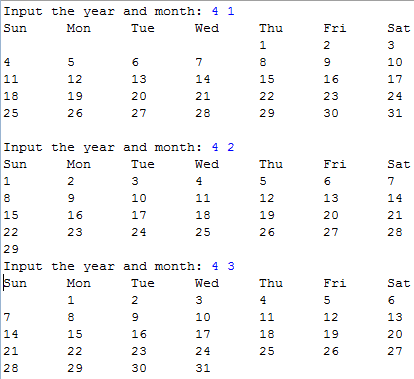
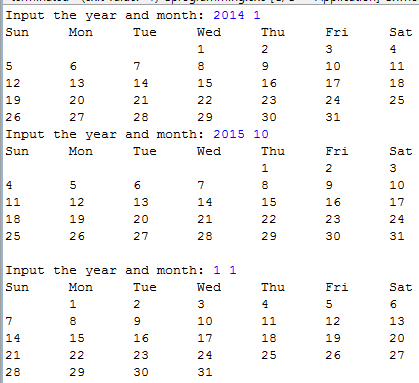
date++;

}

**printf**("\n");

}

**6.3 Result (snap shot)**



**7.Euler Project 6: 1~100까지 자연수에 대한 “합의 제곱”과 “제곱의 합”의 차이**

**7.1 Solution**

For 반복 문 2개를 이용하여 구할 수 있는 간단한 문제이다. 합의 제곱과 제곱의 합을 구하는 차이는 합의 제곱은 제곱을 구하고 더하는 과정이고, 제곱의 합은 일단 모든 수를 더하고 반복 문을 빠져나온 뒤에 제곱을 해주면 된다.

**7.2 Source Code**

**void** **Eproblem6**() //1~100 "합의 제곱"과 "제곱의 합"의 차이

{

**int** sum1 = 0, sum2 = 0;

**int** i, j;

//1~100 제곱의 합

**for** (i = 1; i <= 100; i++)

sum1 += i\*i;

//1~100 합의 제곱

**for** (j = 1; j <= 100; j++)

sum2 += j;

sum2 = sum2 \* sum2;

//결과 출력

**printf**("Answer: %d - %d = %d\n", sum2, sum1, sum2 - sum1);

}

**7.3 Result (snap shot)**



**8. Euler project 7: 10001번째 소수**

**8.1 Solution**

소수를 구하는 알고리즘을 이용하면 쉽다. 2부터 증가하는 정수에 대하여, 2부터 자기 자신까지의 수 중 나누어지는 수가 있으면 소수가 아니고, 자기 자신이 되면 소수이다. 무한 반복 문을 이용하여 소수인 경우 수를 세어서 10001번째가 되면 빠져나오게 설계한다.

**8.2 Source Code**

**void** **Eproblem7**() // 10001번째 소수

{

**int** i;

**int** count = 0;

**int** N = 2;

**while** (1)

{

//소수를 구하는 알고리즘: 2부터 현재 수까지 나눠지는 수가 있으면 break,

// 현재 수까지 도달하면 소수

**for** (i = 2; i < N; i++)

{

**if** ((N % i) == 0)

**break**;

}

//소수이면 count++;

**if** (N == i)

count++;

//10001번째 소수이면 중지

**if** (count == 10001)

**break**;

N++;

}

**printf**("Answer: %d\n", N);

}

**8.3 Result (snap shot)**



**9. Euler Project 9: a+b+c=1000의 피타고라스일 때, aXbXc = ?**

**9.1 Solution**

다양한 방법이 있지만, 필자는 2가지만 시도해보았다. 단순하게 a, b, c를 1부터 1000까지 계산하는 방법과 a, b를 1000까지 올리면서 c를 계산하는 방법이다. 3개의 반복 문과 한 반복 문에 적어도 990번 이상은 반복 해야 하는 특성 때문에 결과 값 출력 또한 생각보다 걸린다.

**9.2 Source Code**

**void** **Eproblem9**() //A+b+C = 1000의 피타고라스일 떄, AXBXC=?

{

**int** a, b = 2, c = 3;

/\*

for (a = 1; a < b ; a++)

{

for (b = 2; b < c; b++)

{

for (c = 3; c <= 997 ;c++)

{

if (a + b + c == 1000 && (a\*a + b\*b == c\*c) && a < b && b < c) {

printf("a:%d b:%d c:%d\n", a, b, c);

printf("a X b X c = %d\n", a\*b\*c);

}

}

}

}

\*/

**for** (a = 1; a > 0; a++)

{

**for** (b = 2; b < 999 - a; b++)

{

c = 1000 - b - a;

**if** (a + b + c == 1000 && (a\*a + b\*b == c\*c) && a < b && b < c) {

**printf**("a:%d b:%d c:%d\n", a, b, c);

**printf**("a X b X c = %d\n", a\*b\*c);

}

}

}

}

**9.3 Result (snap shot)**



**10. Euler project 10: 이백만 이하 소수의 합**

**10.1 Solution**

**바로 떠오르는 코드는 그리 어렵지 않다. 하지만 문제점은 답을 구하는 과정에 있다. 수가 방대할 뿐만 아니라, 수가 커짐에 따라 그 안의 반복 문도 제곱 비례하여 커지기 때문에 결과 값을 얻기 위해 많은 시간이 필요하다. 분명히 이런 시간적 문제를 줄일 수 있는 알고리즘이 있다. 하지만 현재는 반복 문까지 밖에 모르기에 단순히 2부터 수를 증가시켜 소수이면 더하는 방식으로 2백만 까지 가야하는 방법이 최선인 듯 하다.**

**10.2 Source Code**

**void** **Eproblem10**(){//2백만 이하 소수의 합

**int** i = 2 , j;

**const** **int** N = 2000000;

**long** **long** sum = 0;

**while** (i <= N)

{

**for** (j = 2; j < i; j++)

{

**if** ((i % j) == 0)

**break**;

}

**if** (i == j) {

//소수이면 더한다.

sum += i;

**printf**("Sum: %lld\n", sum);

}

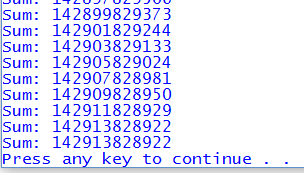
i++;

}

**printf**("Answer: %lld\n", sum);

}

**10.3 Result (snap shot)**



**11. Euler project 12: 500개 이상의 약수를 갖는 가장 작은 삼각수**

**11.1 Solution**

바로 위의 문제와 동일하게 엄청난 시간이 걸린다. 시간상으로는 지금 문제가 2배정도는 더 오래 걸린다. 반복 문안에 반복 문이 존재하면서 수가 늘어남과 동시에 제곱 비례로 그 시간 또한 증가한다. 삼각수를 구할 때 첫 반복 문을 사용하고, 구한 삼각수에 대한 약수를 구하기 위해 또 다시 반복 문을 들어가게 된다. 그렇게 약수가 500개 이상이 되면 무한 반복 문을 빠져나오게 된다. 일단 약수가 500개라는 것은 그 수가 엄청나게 크다는 걸 의미하고, 수가 100000이 넘어가면 이를 1부터 100000까지 반복 문을 돌리는 과정이 된다. 게다가 이클립스의 출력 속도는 상상할 수 없을 정도로 느리기에 Visual Studio를 이용하여 결과물을 출력하였다.

**11.2 Source Code**

**void** **Eproblem12**(){//500개 이상의 약수를 갖는 가장 작은 삼각수

**int** N = 0;

**int** count = 0;

**int** i = 1, j = 0;

**while** (1)

{

//삼각수 구하기

**for** (i = 1; i <= j; i++)

N += i;

**printf**("N: %d\n", N);

//삼각수의 약수 개수 구하기

**for** (i = 1; i <= N; i++)

**if** (N % i == 0)

count++;

**printf**("count: %d\n", count);

**if** (count >= 500)

**break**;

j++;

N = 0;

count = 0;

}

}

**11.3 Result (snap shot)**

