```c
int main (void){

    printf("총 임대요금 : %d\n",Prob1(7));
    printf("1~1000 사이 3의 배수의 합: %d\n",Prob2());
    printf("1~1000 사이 4나 6 나눌때 나머지 1인 수 합: %d\n",Prob3());
    printf("7의 배수인 마지막 항(10항의 경우) : %d\n",Prob4(10));
    Prob5();



    return 0;
}
```

```
sunghwan@HWAN:~/Documents$ ./assignmentw1
총 임대요금 : 232500
1~1000 사이 3의 배수의 합: 166833
1~1000 사이 4나 6 나눌때 나머지 1인 수 합: 41916
7의 배수인 마지막 항(10항의 경우) : 70
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
4 * 1 = 4
```

1. 스키장에서 스키 장비를 임대하는데 37500 원이 든다.

또 3 일 이상 이용할 경우 20%를 할인 해준다.

일주일간 이용할 경우 임대 요금은 얼마일까 ?

(연산 과정은 모두 함수로 돌린다)

```c
int Prob1(int days){
    int fee=37500;
    int sum=0;
    for(int i=1;i<=days;i++){
        if(i>3)
            sum+=(fee*0.8);
        else
            sum+=fee;
    }
    return sum;
}
```

3. 1 ~ 1000 사이에 3 의 배수의 합을 구하시오.

```
int Prob2(void){
    int sum=0;
    for(int i=1;i<=1000;i++){
        if(i%3==0)
            sum+=i;
    }
    return sum;
}
```

4. 1 ~ 1000 사이에 4 나 6 으로 나눠도 나머지가 1 인 수의 합을 출력하라.

```
int Prob3(void){
    int sum=0;
    for(int i=1;i<=1000;i++){
        if(i%4==1 && i%6==1)
            sum+=i;
    }
    return sum;
}
```

5. 7 의 배수로 이루어진 값들이 나열되어 있정한다.

함수의 인자(input)로 항의 갯수를 받아서 마지막 항의 값을 구하는 프로그램을 작성하라.

```
int Prob4(int size){
    if(size==1)
        return 7;
    else
        return Prob4(size-1)+7;
}
```

7. C 로 함수를 만들 때, Stack 이란 구조가 생성된다.

이 구조가 어떻게 동작하는지 Assembly Language 를 해석하며 기술해보시오.

esp, ebp, eip 등의 Register 에 어떤 값이 어떻게 들어가는지 등등

메모리에 어떤 값들이 들어가는지 등을 자세히 기술하시오.

```
Int mult2(int num){
        return num * 2;
}

int main(void){
        int i, sum = 0, result;
        for(i = 0; i < 5; i++)
                sum += i;
        result = mult2(sum);
        return 0;
}
<main>
  0x00000000004004e4 <+0>:     push   %rbp // *(rsp)=rbp, rsp--
  0x00000000004004e5 <+1>:     mov    %rsp,%rbp // rbp=rsp;
// rsp 감소 후 해당 주소에 rbp 값 입력 후 rsp 와 rbp 를 같게 하여 새로운 base 생성(rbp1)
  0x00000000004004e8 <+4>:     sub    $0x10,%rsp // rsp=rsp-16;
// rsp 주소 값을 16 만큼 감소시킴
  0x00000000004004ec <+8>:     movl   $0x0,-0x8(%rbp) // *(rbp-8)=0;
  0x00000000004004f3 <+15>:    movl   $0x0,-0xc(%rbp) // *(rbp-12)=0;
// rpb 기준으로 8, 12 감소시킨 곳을 0 으로 만듦
  0x00000000004004fa <+22>:    jmp    0x400506 <main+34> // ip=0x400506
// ip 값을 400506 으로 바꾸어 다음 실행 될 행으로 이동
  0x00000000004004fc <+24>:    mov    -0xc(%rbp),%eax // eax = *(rbp-12);
  0x00000000004004ff <+27>:    add    %eax,-0x8(%rbp) // *(rbp-8) = eax;
  0x0000000000400502 <+30>:    addl   $0x1,-0xc(%rbp) // *(rbp-12)+=1;
  0x0000000000400506 <+34>:    cmpl   $0x4,-0xc(%rbp) // *(rbp-12) < 4 compare
```

0x000000000040050a <+38>: jle 0x4004fc <main+24> // *(rbp-12) < 4 →ip=0x4004fc

// rbp1-12 값을 4 와 비교하여 4 보다 작다면 4004fc 행을 수행하러 이동

// eax 에 rbp1 -12 값을 대입하고 rbp -8 위치에 eax 대입함.

// 그리고 rbp1 -12 위치에 +1 씩 해준다. 그리고 계속 반복 수행

// 즉 -8 위치가 sum 으로 보면 되고 -12 위치를 i 로 보면 된다.

0x000000000040050c <+40>: mov -0x8(%rbp),%eax // eax = *(rbp-a)

0x000000000040050f <+43>: mov %eax,%edi // edi = eax

0x0000000000400511 <+45>: callq 0x4004d6 <mult2> // push and jump 0x4004d6

// sum 인 rbp1-8 의 값을 eax 에 복사.

//edi 에 eax 값을 복사하여 매개변수 전달을 할 준비를 한다.

//rsp 값 감소 시키고 실행 값 전달 및 ip 에 다음 실행주소를 전달(0x4004d6)한다.

0x0000000000400516 <+50>: mov %eax,-0x4(%rbp) // *(rbp-4)=eax

0x0000000000400519 <+53>: mov $0x0,%eax // eax =0

0x000000000040051e <+58>: leaveq //

0x000000000040051f <+59>: retq // ip = *(rsp)

// eax 를 rbp1-4 에 저장 한 후에 0 으로 초기화 하고 rsp0 으로 돌아가게 된다.

<mult2 function>

0x00000000004004d6 <+0>: push %rbp // *(rsp)=rbp, rsp--

0x00000000004004d7 <+1>: mov %rsp,%rbp // rbp = rsp

// rsp 감소 후 해당 주소에 rbp 값 입력 후 rsp 와 rbp 를 같게 하여 새로운 base 생성 (rbp2)

0x00000000004004da <+4>: mov %edi,-0x4(%rbp) // *(rbp-4)=edi

0x00000000004004dd <+7>: mov -0x4(%rbp),%eax // eax = *(rbp-4)

0x00000000004004e0 <+10>: add b%eax,%eax // eax+=eax

// 매개변수 역할 인 edi 를 rbp2-4 값에 대입한다.

// rbp-4 의 값을 eax 로 옮긴다.

// eax=eax+eax 를 수행하여 2*eax 값을 만든다.

0x00000000004004e2 <+12>: pop %rbp // rbp=*(rsp), rsp++

0x00000000004004e3 <+13>: retq // ip = *(rsp)

// rsp 의 값에 rsp 가 가르키던 rbp1 의 값을 전해주고 rsp 는 증가한다.

// 다음 실행 ip 에 복귀주소를 전달한다.

10. 구구단을 만들어보시오.

```
void Prob5(void){
    for(int i=1;i<10;i++){
        for(int j=1;j<10;j++){
            printf("%d * %d = %d\n",i,j,i*j);
        }
    }
}
```

12. Visual Studio 에서 Debugging 하는 방법에 대해 기술해보시오.

Break Point 는 어떻게 잡으며, 조사식, 메모리, 레지스터등의 디버그 창은

각각 어떤 역할을 하고 무엇을 알고자 할 때 유용한지 기술하시오.

Gcc -g -O0 -o 파일이름 c 파일이름 으로 디버깅 파일 생성

gcb 파일이름을 통하여 디버거 모드로 들어간다.

B main 으로 메인 함수를 정지점으로 만들어서 r 로 실행 시킨다.

B *주소값을 통하여 원하는 곳을 정지점으로 만들 수 있다.