1. 스키장에서 스키 장비를 임대하는데 37500원이 든다.

또 3일 이상 이용할 경우 20%를 할인 해준다.

일주일간 이용할 경우 임대 요금은 얼마일까 ?

```
#include <stdio.h>

 int rental(int day , int rental_fee){

      int result;
     if(day >= 3){
          result = rental_fee * day * 0.8;
     }else{
          result = rental_fee * day;
     }
     return result;
}
 int main(void){
     printf("총 값은? %d\n", rental(7,37500));
     return 0;
}
```

3. 1 ~ 1000사이에 3의 배수의 합을 구하시오.

```
#include<stdio.h>

int main(void){
 int i=1 , result =0;
while(i<=1000){
if(i%3 ==0){
result = i+result;
}
```

```
i++;
}

printf("1~1000 3배 수 합 : %d\n",result);
return 0;
}
```

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

```
#include<stdio.h>
int divide(){
int a=1 ,b= 0
while(a<=1000){
if((a%4==1) || (a%6 ==1)){
b= b+a;
}
a++;
}
return b;
}

int main(void){
printf("1~1000사이 4나 6으로 나눠도 나머지가 1인수의합 : %d" , divide());
return 0;
}
```

5. 7의 배수로 이루어진 값들이 나열되어 있다고 가정한다.

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

```
#include<stdio.h>
int main(void){
int num;
printf("정수를 입력하세요 : ");
scanf("%d\n" , &num);

printf("%d\n" , num*7);
return 0 ;
}
```

7. C로 함수를 만들 때, Stack이란 구조가 생성된다.

이 구조가 어떻게 동작하는지 Assembly Language를 해석하며 기술해보시오.

esp, ebp, eip등의 Register에 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;
}
```

10. 구구단을 만들어보시오.

```
#include <stdio.h>
int main (void)
int i= 2 , j=1;
```

```
while(i<10){
j=1;
while(j<10){
printf("%d * %d = %d , i, j ,i*j);
}
printf("\n");
i++;
}
return =0;
}
```

12. Visual Studio에서 Debugging하는 방법에 대해 기술해보시오.

Break Point는 어떻게 잡으며, 조사식, 메모리, 레지스터등의 디버그 창은

각각 어떤 역할을 하고 무엇을 알고자 할 때 유용한지 기술하시오.

:
우선 사용할 저장소를 다운받는다 그전에 디렉토리를 먼저 만든다.
mkdir my_proj
cd my_proj
이후 git으로 저장소를 받는다
만약 이미 저장소를 clone 받았다면
cd homework
git pull origin master

위와같이 입력하면 갱신(업데이트)된 내용을 받아 볼 수 있다
pwd를 입력하여 현재 위치를 확인하고 아래 위치로 이동한다
cd ~/my_roj/homework/sanghoonlee

디버깅 옵션을 집어넣어서 컴파일 한다.

gcc -g -o debug func1.c
정상적으로 컴파일 되었다면 ls 입력시 debug가 보일 것이다
./debug

위와 같이 입력하면 6이 잘 출력되는지 확인한다.
잘 출력되었다면 프로그램에 이상이 없으므로 정상적인 어셈블리 분석이 가능함을 의미한다.

*instruction scheduling(명령어 스케쥴림)
요즘gcc(컴파일러가) 너무 □Q똑해져서
최적화 옵션을 주지 않아도 알아서 최적화해버리는데
강제로 최적화를 못하게 할 수 있다.
-o0 옵션을 추가하면 된다: 영문자o 와 숫자 0이다.
즉위의 컴파일 옵션을 아래와 같이 바꾼다
gcc -g -o0 -o debug func1.c
모든 최적화를 방지하고 디버깅 옵션을 집어넣었음을 의미한다

*왜 이 작업이 필요한가?

디버깅 작업시에 가장 큰 문제가 c언어 소스코드와 실제 cpu의 동작 흐름이 일치하지 않으면 분석을 하기가 굉장히 어렵다.
명령어를 스케쥴링 한다는 것은 아래와 같은 의미다.
어떤 명령어를 처리하는데 10 clock이 필요한 것이 있고 어떤 것은 5clock 이 걸리고 어떤것은 1clock이 걸린다
이 배치를 효율적으로 진행해서 가장 빠르게 실행될 수 있게 만들어 주는 작업을 명령어 스케쥴링이라고 한다.
그래서 컴파일러의 최적화 옵션이 활성화되면
프로그램이 여기 갔다 저기 갔다 왔다 갔다 한다.
(정신이 없으니 분석하기가 매우 힘들다)

이러한 이류 □□문에 디버깅시에는 반드시-00옵션을 주도록 한다.

*이제 디버거를 켜보자!
gdb debug

이와 같이 입력하면 디버거가 켜지면서 (gdb) 창이 보일 것이다.
어제 작업했뜻이 main 함수에 breakpoint(정지)를 걸어야 한다.

b main

위와 같이 입력하면 main 함수에서 멈춰달라는 뜻이다.
r을 눌러서 프로그램을 구동하면 main 함수에 걸릴 것이다.
단 c 언어 기반으로 움직여서 선두의 어셈블리어를 지나치게 됨
우선 현재 어느 위치에 있는지 파악하기 위해 디스어셈블리를 수행하도록 한다.

disas 이 명령어를 입력하면 현재 프로그램에 대한 디스어셈블리된 어셈블리어 코드가 보인다.
그리고 화살표가 보일 것이다
여기서 p/x $rip라고 입력해보도록 한다.

아주 흥미롭게도 '=>'가 나타내는 주소값과 p/x $rip가 출력하는 값이 같을 것이다.
앞서서 레지스터를 설명할 때
up 값은 다음에 실행할 명령어의 주소를 가르킨다고 했었다 (이 내용이 증명 되었다)
어쨋든 실행을 맨 처음으로 돌려야 하므로
disas 해서 맨 첫 라인의 주소를 복사한다
그리고 아래와 같이 breakpoint를 걸어준다.

b*복사한 주소
b 하고 띄어쓴 다음에 * 하고 복사한 주소를 붙여쓴다
그리고 다시 r을 눌러서 실행하면
'=>'가 맨 처음에 배치되어 있음을 볼 수 있을 것이다.
(이거 볼 땐 disas 를 입력함)

조사식,메모식의 디버그창은 어떤 역할을 하며 어디서 유용한지
:
어떤 소스에 많은 정수 배열이 할당된 포인터 변수가 있다.
그 많은 창을 보기 위해 조사식 창이 필요하다.
조사식에 나온 주소를 메모리 주소에 입력해도 되고 주소 대신 변수를 입력해도 자동 변환되어 그 메모리를 볼 수 있다.
메모리창에서 10진수 정수로 볼 수 있다.

레지스터의 디버그창은 어떤 역할을 하며 어디서 유용한지
레지스터 창을 열어 두면 코드 실행에 따라 레지스터 값이 변경되는 것을 볼 수 있다.
레지스터 값이 변하는 것을 분석하면 메모리가 어떤식으로 운영하는지 알 수 있다.