

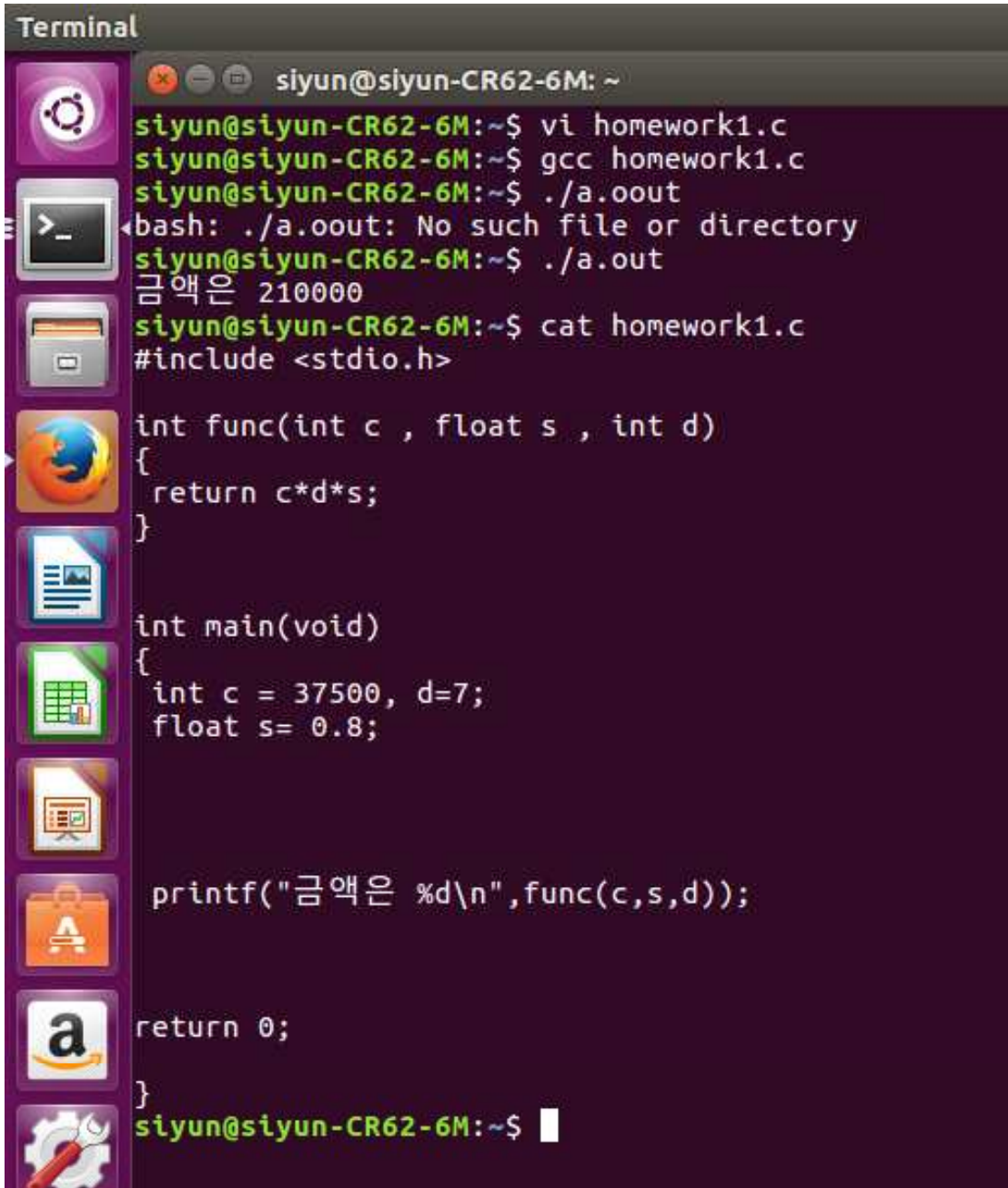
# Embedded Class

## Homework#3

### -목차

1. 스키장에서 스키 장비를 임대하는데 37500원이 든다.  
또 3일 이상 이용할 경우 20%를 할인 해준다.  
일주일간 이용할 경우 임대 요금은 얼마일까 ?  
(연산 과정은 모두 함수로 돌린다)
2. 1 ~ 1000사이에 3의 배수의 합을 구하시오.
3. 1 ~ 1000사이에 4나 6으로 나눠도 나머지가 1인 수의 합을 출력하라.
4. 7의 배수로 이루어진 값들이 나열되어 있다고 가정한다.  
함수의 인자(input)로 항의 갯수를 받아서 마지막 항의 값을 구하는 프로그램을 작성하라.
5. 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.
6. 구구단을 만들어보시오.
7. 리눅스에서 디버깅 하는 방법을 정리

1.

A terminal window titled "Terminal" with a dark background and light text. The window shows a series of commands and their outputs. The user is at the prompt "siyun@siyun-CR62-6M: ~". The commands and outputs are: "vi homework1.c" (no output), "gcc homework1.c" (no output), "./a.oout" (error: "bash: ./a.oout: No such file or directory"), and "./a.out" (output: "금액은 210000"). Then, the user runs "cat homework1.c", which displays the following C code: 

```
#include <stdio.h>

int func(int c , float s , int d)
{
    return c*d*s;
}

int main(void)
{
    int c = 37500, d=7;
    float s= 0.8;

    printf("금액은 %d\n",func(c,s,d));

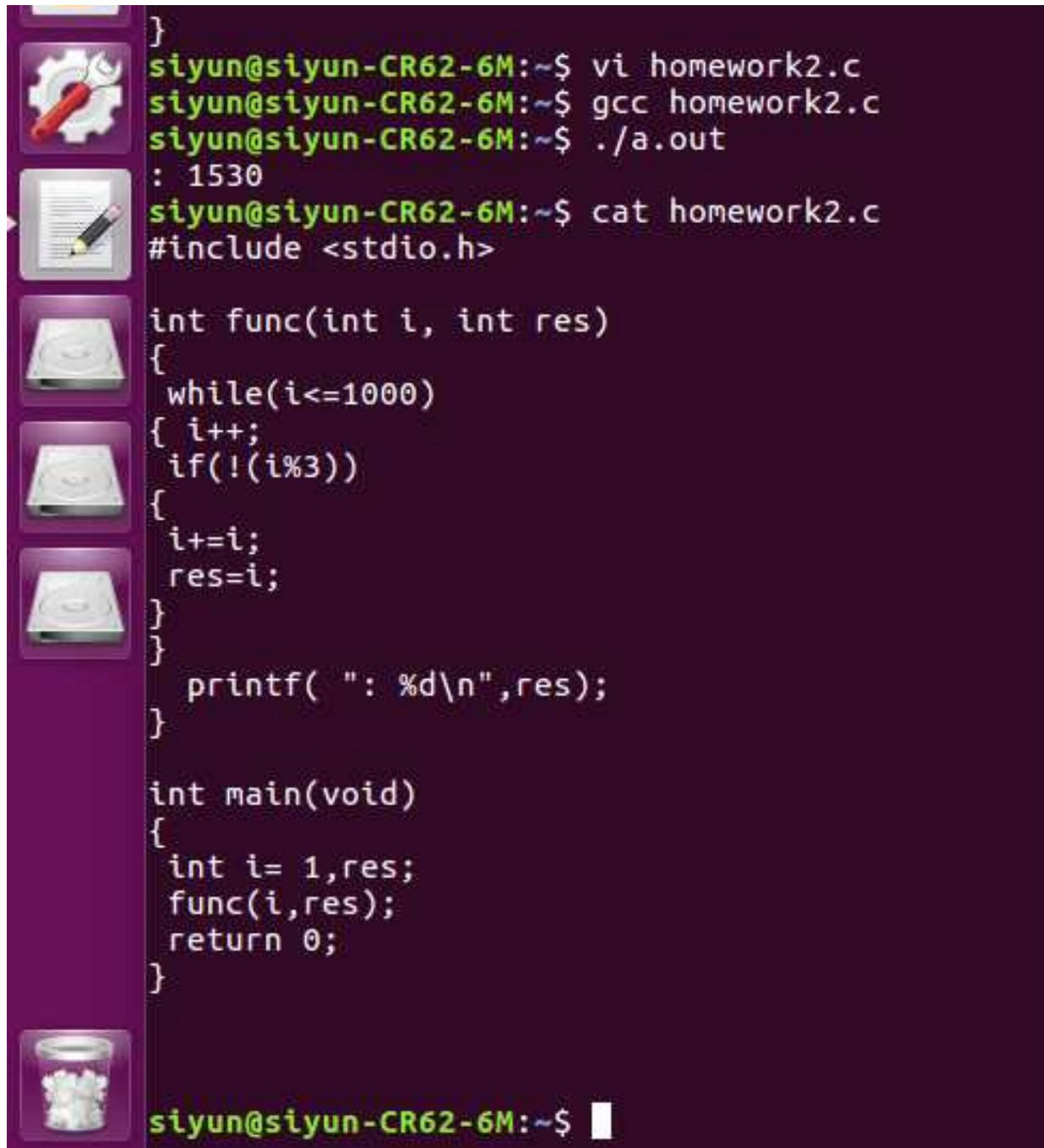
    return 0;
}
```

 The terminal window has a sidebar on the left with various application icons like a terminal, file manager, web browser, and office applications.

[그림1]

금액은 210,000원으로 계산결과가 나왔다 소스코드는 [그림1]과 같다.

2.



A terminal window with a dark purple background and a vertical toolbar on the left. The toolbar contains icons for a gear, a document with a pencil, a floppy disk, and a trash can. The terminal text shows the execution of a C program and its source code.

```
}
siyun@siyun-CR62-6M:~$ vi homework2.c
siyun@siyun-CR62-6M:~$ gcc homework2.c
siyun@siyun-CR62-6M:~$ ./a.out
: 1530
siyun@siyun-CR62-6M:~$ cat homework2.c
#include <stdio.h>

int func(int i, int res)
{
    while(i<=1000)
    { i++;
      if(!(i%3))
      { i+=i;
        res=i;
      }
    }
    printf( ": %d\n",res);
}

int main(void)
{
    int i= 1,res;
    func(i,res);
    return 0;
}

siyun@siyun-CR62-6M:~$
```

[그림2]

1부터 1000까지 3의 배수합을 계산하였다.  
함수로 모든 소스코드를 작성한후  
메인에서 불러왔다.

3.

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int i=1,x=1,res,res1;
```

```
while(i<=1000)
```

```
{i++;
```

```
    if(i%4==1)
```

```
    {res = i;}
```

```
}
```

```
while(x<=1000)
```

```
{x++;
```

```
    if(x%6==1)
```

```
    {res1 =x;}
```

```
}
```

```
    if(res==res1)
```

```
    {res1+=res1;}
```

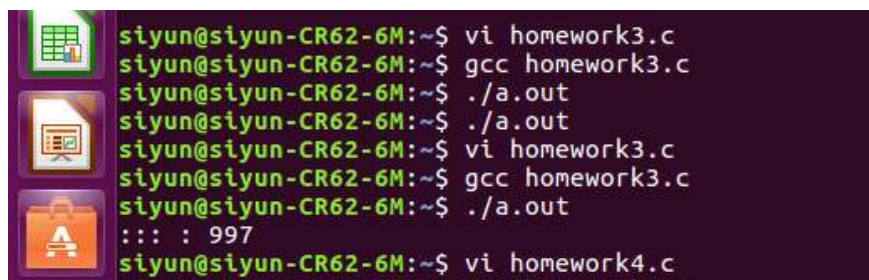
```
printf("::: : %d\n",res1);
```

```
    return 0;
```

```
}
```

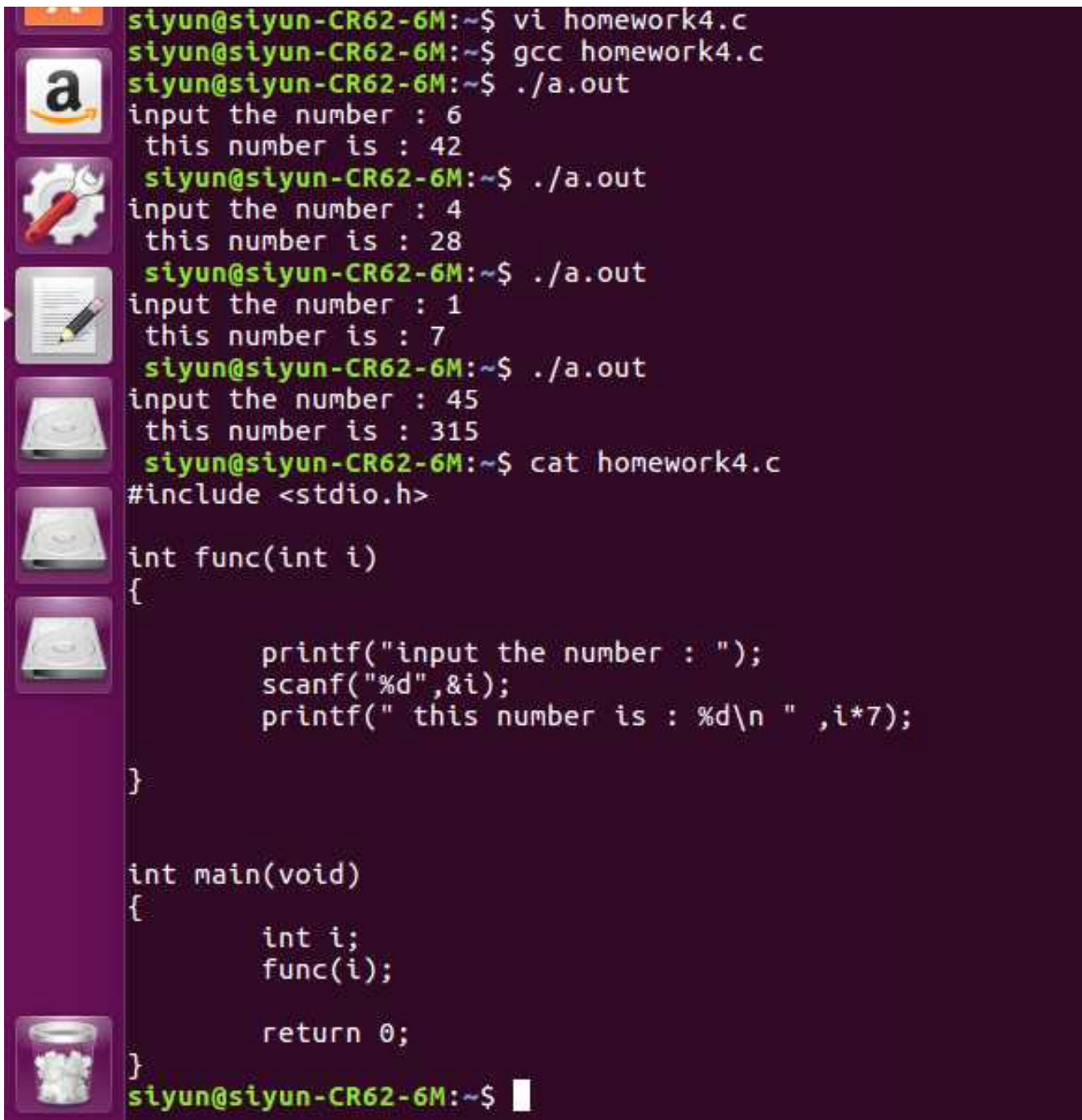
소스코드 입력해서 동작하는거 확인했습니다.

확인한 사진파일이 일부분밖에 안나와 소스코드 올립니다.



```
siyun@siyun-CR62-6M:~$ vi homework3.c
siyun@siyun-CR62-6M:~$ gcc homework3.c
siyun@siyun-CR62-6M:~$ ./a.out
siyun@siyun-CR62-6M:~$ ./a.out
siyun@siyun-CR62-6M:~$ vi homework3.c
siyun@siyun-CR62-6M:~$ gcc homework3.c
siyun@siyun-CR62-6M:~$ ./a.out
::: : 997
siyun@siyun-CR62-6M:~$ vi homework4.c
```

4.

A terminal window with a dark purple background and a sidebar on the left containing various icons (Amazon, tools, documents, etc.). The terminal shows the execution of a C program named 'homework4.c'. The program takes an input number and outputs its value multiplied by 7. The execution results are: input 6 outputs 42, input 4 outputs 28, input 1 outputs 7, and input 45 outputs 315. Finally, the source code of 'homework4.c' is displayed, showing a function 'func' that calculates the result and a 'main' function that calls it.

```
siyun@siyun-CR62-6M:~$ vi homework4.c
siyun@siyun-CR62-6M:~$ gcc homework4.c
siyun@siyun-CR62-6M:~$ ./a.out
input the number : 6
this number is : 42
siyun@siyun-CR62-6M:~$ ./a.out
input the number : 4
this number is : 28
siyun@siyun-CR62-6M:~$ ./a.out
input the number : 1
this number is : 7
siyun@siyun-CR62-6M:~$ ./a.out
input the number : 45
this number is : 315
siyun@siyun-CR62-6M:~$ cat homework4.c
#include <stdio.h>

int func(int i)
{
    printf("input the number : ");
    scanf("%d",&i);
    printf(" this number is : %d\n " ,i*7);
}

int main(void)
{
    int i;
    func(i);

    return 0;
}
siyun@siyun-CR62-6M:~$
```

7의 배수 항을 입력했을 경우 출력되는 7의 배수를 보았다.  
항의 첫 번째 항을 1항이라고 가정했을경우의 소스코딩이다.



5.

```
siyun@siyun-CR62-6M: ~
Breakpoint 2, main () at test3.c:15
15 {
(gdb) disas
Dump of assembler code for function main:
=> 0x00000000004004e4 <+0>:      push    %rbp
    0x00000000004004e5 <+1>:      mov     %rsp,%rbp
    0x00000000004004e8 <+4>:      sub     $0x10,%rsp
    0x00000000004004ec <+8>:      movl    $0x0,-0x8(%rbp)
    0x00000000004004f3 <+15>:     movl    $0x0,-0xc(%rbp)
    0x00000000004004fa <+22>:     jmp     0x400506 <main+34>
    0x00000000004004fc <+24>:     mov     -0xc(%rbp),%eax
    0x00000000004004ff <+27>:     add     %eax,-0x8(%rbp)
    0x0000000000400502 <+30>:     addl    $0x1,-0xc(%rbp)
    0x0000000000400506 <+34>:     cmpl    $0x4,-0xc(%rbp)
    0x000000000040050a <+38>:     jle     0x4004fc <main+24>
    0x000000000040050c <+40>:     mov     -0x8(%rbp),%eax
    0x000000000040050f <+43>:     mov     %eax,%edi
    0x0000000000400511 <+45>:     callq   0x4004d6 <mult2>
    0x0000000000400516 <+50>:     mov     %eax,-0x4(%rbp)
    0x0000000000400519 <+53>:     mov     $0x0,%eax
    0x000000000040051e <+58>:     leaveq  %eax
    0x000000000040051f <+59>:     retq
End of assembler dump.
(gdb) █
```

push 와 mov 는 stack frame 설정이다.

```
siyun@siyun-CR62-6M: ~
0x00000000004004fc <+24>:     mov     -0xc(%rbp),%eax
0x00000000004004ff <+27>:     add     %eax,-0x8(%rbp)
0x0000000000400502 <+30>:     addl    $0x1,-0xc(%rbp)
0x0000000000400506 <+34>:     cmpl    $0x4,-0xc(%rbp)
0x000000000040050a <+38>:     jle     0x4004fc <main+24>
0x000000000040050c <+40>:     mov     -0x8(%rbp),%eax
0x000000000040050f <+43>:     mov     %eax,%edi
0x0000000000400511 <+45>:     callq   0x4004d6 <mult2>
0x0000000000400516 <+50>:     mov     %eax,-0x4(%rbp)
0x0000000000400519 <+53>:     mov     $0x0,%eax
0x000000000040051e <+58>:     leaveq  %eax
0x000000000040051f <+59>:     retq
End of assembler dump.
(gdb) p/x $rbp
$1 = 0x400520
(gdb) p/x $rsp
$2 = 0x7fffffffdfcf8
(gdb) x $rbp
0x400520 <__libc_csu_init>:    0x56415741
(gdb) x $rsp
0x7fffffffdfcf8: 0xf7a2d830
(gdb) ls
Undefined command: "ls".  Try "help".
(gdb) █
```

```
siyun@siyun-CR62-6M: ~
0x00000000004004e4 <+0>:      push    %rbp
0x00000000004004e5 <+1>:      mov     %rsp,%rbp
=> 0x00000000004004e8 <+4>:      sub     $0x10,%rsp
0x00000000004004ec <+8>:      movl    $0x0,-0x8(%rbp)
0x00000000004004f3 <+15>:     movl    $0x0,-0xc(%rbp)
0x00000000004004fa <+22>:     jmp     0x400506 <main+34>
0x00000000004004fc <+24>:     mov     -0xc(%rbp),%eax
0x00000000004004ff <+27>:     add     %eax,-0x8(%rbp)
0x0000000000400502 <+30>:     addl    $0x1,-0xc(%rbp)
0x0000000000400506 <+34>:     cmpl    $0x4,-0xc(%rbp)
0x000000000040050a <+38>:     jle     0x4004fc <main+24>
0x000000000040050c <+40>:     mov     -0x8(%rbp),%eax
0x000000000040050f <+43>:     mov     %eax,%edi
0x0000000000400511 <+45>:     callq   0x4004d6 <mult2>
0x0000000000400516 <+50>:     mov     %eax,-0x4(%rbp)
0x0000000000400519 <+53>:     mov     $0x0,%eax
0x000000000040051e <+58>:     leaveq  %eax
0x000000000040051f <+59>:     retq
End of assembler dump.
(gdb) x $rsp
0x7fffffffddcf0: 0x00400520
(gdb) x $rbp
0x7fffffffddcf0: 0x00400520
(gdb) si
```

여기서 sup를 행하게 되면 0x10 만큼 즉 16바이트 만큼의 공을 만든후 8바이트에다 rbp를 저장한다.

```
siyun@siyun-CR62-6M: ~
0x00000000004004e5 <+1>:      mov     %rsp,%rbp
0x00000000004004e8 <+4>:      sub     $0x10,%rsp
=> 0x00000000004004ec <+8>:      movl    $0x0,-0x8(%rbp)
0x00000000004004f3 <+15>:     movl    $0x0,-0xc(%rbp)
0x00000000004004fa <+22>:     jmp     0x400506 <main+34>
0x00000000004004fc <+24>:     mov     -0xc(%rbp),%eax
0x00000000004004ff <+27>:     add     %eax,-0x8(%rbp)
0x0000000000400502 <+30>:     addl    $0x1,-0xc(%rbp)
0x0000000000400506 <+34>:     cmpl    $0x4,-0xc(%rbp)
0x000000000040050a <+38>:     jle     0x4004fc <main+24>
0x000000000040050c <+40>:     mov     -0x8(%rbp),%eax
0x000000000040050f <+43>:     mov     %eax,%edi
0x0000000000400511 <+45>:     callq   0x4004d6 <mult2>
0x0000000000400516 <+50>:     mov     %eax,-0x4(%rbp)
0x0000000000400519 <+53>:     mov     $0x0,%eax
0x000000000040051e <+58>:     leaveq  %eax
0x000000000040051f <+59>:     retq
End of assembler dump.
(gdb) x $rsp
0x7fffffffddce0: 0xffffddd0
(gdb) x $rbp
0x7fffffffddcf0: 0x00400520
(gdb) x $rsp
0x7fffffffddce0: 0xffffddd0
```

확인결과 rsp가 16바이트 크기만큼 줄어든걸 알수 있었다.

```
siyun@siyun-CR62-6M: ~
Dump of assembler code for function main:
0x00000000004004e4 <+0>:    push    %rbp
0x00000000004004e5 <+1>:    mov     %rsp,%rbp
0x00000000004004e8 <+4>:    sub     $0x10,%rsp
0x00000000004004ec <+8>:    movl    $0x0,-0x8(%rbp)
0x00000000004004f3 <+15>:   movl    $0x0,-0xc(%rbp)
=> 0x00000000004004fa <+22>:   jmp     0x400506 <main+34>
0x00000000004004fc <+24>:   mov     -0xc(%rbp),%eax
0x00000000004004ff <+27>:   add     %eax,-0x8(%rbp)
0x0000000000400502 <+30>:   addl    $0x1,-0xc(%rbp)
0x0000000000400506 <+34>:   cmpl    $0x4,-0xc(%rbp)
0x000000000040050a <+38>:   jle     0x4004fc <main+24>
0x000000000040050c <+40>:   mov     -0x8(%rbp),%eax
0x000000000040050f <+43>:   mov     %eax,%edi
0x0000000000400511 <+45>:   callq   0x4004d6 <mult2>
0x0000000000400516 <+50>:   mov     %eax,-0x4(%rbp)
0x0000000000400519 <+53>:   mov     $0x0,%eax
0x000000000040051e <+58>:   leaveq  %eax
0x000000000040051f <+59>:   retq
End of assembler dump.
(gdb) x $rvp
Value can't be converted to integer.
(gdb) x $rbp
0x7fffffffdfcf0: 0x00400520
```

여기서 점프해서 34번째줄로 갔다가 다시 jle를 만나게되면 위로 올라온다  
소스코드에 쓰여있는 for 문 때문에 반복문이 동작하고있었다.  
조건이 만족하게 되면 jle를 탈출하여 밑에 mov 를실행시키며  
eax 레지스터에 값을 저장하고 edi 로 옮긴후  
eax를 초기화 시킨다.



6.

```
siyun@siyun-CR62-6M: ~  
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  
4 * 2 = 8  
4 * 3 = 12  
4 * 4 = 16  
4 * 5 = 20  
4 * 6 = 24  
4 * 7 = 28  
4 * 8 = 32  
4 * 9 = 36  
5 * 1 = 5  
5 * 2 = 10  
5 * 3 = 15  
5 * 4 = 20  
5 * 5 = 25  
5 * 6 = 30  
5 * 7 = 35  
5 * 8 = 40  
5 * 9 = 45  
6 * 1 = 6  
6 * 2 = 12  
6 * 3 = 18  
6 * 4 = 24  
6 * 5 = 30  
6 * 6 = 36  
6 * 7 = 42  
6 * 8 = 48  
6 * 9 = 54  
7 * 1 = 7  
7 * 2 = 14  
7 * 3 = 21  
7 * 4 = 28  
7 * 5 = 35  
7 * 6 = 42
```

우선 구구단의 소스코드를 돌려보면 위와 같은 결과를 갖는다.

```
siyun@siyun-CR62-6M: ~  
#include <stdio.h>  
  
int main(void)  
{  
int num=1,i;  
while(num<10)  
{  
    i=1;  
    while(i <10){  
        printf("%d * %d = %d \n",num,i,num*i);  
        i++;  
    }  
    num++;  
}  
return 0;  
}  
~  
~  
~  
~  
~  
1,1 All
```

처음에 switch case로 해보다가 실패해서  
중첩 while문을 사용하여 소스코딩하였다..

함수로 하고싶었는데. 도저히 복잡해서 할 수가 없었다..

7.

Linux 디버깅에 대해서 설명하시오.

어셈블리어 디버깅은 컴퓨터의 메모리의 이동경로를 볼 수 있다.

레지스터가 저장할 수 있는 공간이 한계가 있어 이걸 임시 메모리 stack 에 저장했다가 원하는 때 빼내는 메모리 이동경로를 볼 수 있다.

어셈블리어로 디버깅을하면 맨처음 첫 동작은 stack frame을 만든다.

여태까지 했던 디버깅들은

rsp 에 16바이트 공간을 만들고 8바이트를 레지스터에 보냈다.

만약 함수가 있을 경우에는 복귀주소를 만들고

함수의 스택프레임을 다시 만든다

그리고 함수로 점프해 그 함수를 실행한 후 다시 복귀주소로 돌아온다.

C: 연산 결과가 캐리(Carry)를 가질 때 Set '1'

메모리 접근 명령(Memory Accesss Instruction)

Syntax: ldr<cond><B> Rd, label

ldr<cond><B><T> Rd, [Rn]

ldr<cond><B> Rd, [Rn, FlexOffset]<!> ;Pre-Indexed<Auto-Indexing>

ldr<cond><B><T> Rd, [Rn], FlexOffset ;Post-Indexed

<B>: B Suffix가 있을 경우 8-bit Unsigned byte 단위로 Access, 없을 경우 32-bit word로 Access

<T>: T suffix가 있을 경우 Processor가 User mode에서 memory access 처리

FlexOffset:

┐.#Immediate: -4095 부터 -4096사이의 상수 값

└.{-}Rm{, shift연산}: Rm은 음의 부호를 가질 수 있으며, Rm의 Shift 연산도 가능함