

# HOMEWORK

이름	문지희
날짜	2018/2/23
수업일수	3일차
담당교수	이상훈

# 목차

1. 데이터 타입
2. 음수와 보수
3. 오버플로우, 언더플로우
4. 아스키코드
5. 기본 산술 연산자
6. 전위, 후위 연산
7. 비트연산자
8. 관계연산자
9. 논리 연산자
10. 쏫컷의 이점
11. 연산자 예제
12. Scanf
13. if문
14. switch문
15. while문
16. 과제

## 1. 데이터 타입(자료형)

데이터 타입(자료형) : 실수치, 정수, 불린 자료형같은 여러 종류의 데이터를 식별하는 분류로서 더 나아가 해당 자료형에 대한 가능한 값, 해당 자료형에서 수행을 마칠 수 있는 명령들, 데이터의 의미, 해당 자료형의 값을 저장하는 방식을 결정한다.

int, short, char, float, double, long double

Char : 1 byte(8 비트 =  $2^8$ 개에 해당)

Short : 2 byte(16 비트 =  $2^{16}$ 개에 해당)

Int : 4 byte(32 비트 =  $2^{32}$  개를 나타낼 수 있음)

Float : 4 byte

Double : 8 byte( $2^{64}$ )

long double : 12 or 16 byte

\* 변수를 선언하기만 했을 때에는 실제로 메모리할당이 되지 않고 초기값을 넣거나 변수에 값이 들어갔을 때 부터 메모리가 할당된다.

Ex)

```
int date;           >메모리 할당x
Date=3;            >메모리 할당o
```

## 2. 음수와 보수

\* 데이터 타입에 unsigned가 있으면 음수값이 없고, unsigned 가 없으면 음수값이 존재한다.

\* 부호비트가 0이면 양수이고, 부호비트가 1이면 음수이다.

- 음수로 바꾸는 법

부호가 다르고 절대값이 다른 수를 더하면 0이 되게 되는데 2진수 일 때에도 마찬가지 일 것이다. 아래 예시로 8비트로 표현한 1과 -1을 더하면 0이 나오게 되는데 이를 해결하기 위해 보수의 개념이 등장한다. 보수를 구할 때 빠르게 구하는 방법은 뒤에서부터 앞으로 1을 만날 때 까지 유지시키다가 1을 만난 이후부터 반전 시켜주면 음수 값을 구할 수 있다.

Ex1)

```
0000 0001    ①
1111 1111 +   ②
+0000 0000    ③
```

①+②=③ 이고, ③의 9비트인 맨 앞의 1은 버리게 되면 ①과 ②의 합이 0이 되는 것을 알 수 있다.

Ex2)

```
0000 0101
1111 1011 +
+ 0000 0000
```

### 3. 오버플로우, 언더플로우

-오버플로우 : 데이터 값이 표현할 수 있는 최대 값이 있는데 이 최대 값보다 올라가 데이터의 최대값을 벗어난 경우 맨 아래로 그 값이 저장되는 현상.

-언더플로우 : 데이터 값이 표현할 수 있는 최소 값이 있는데 이 최소 값보다 내려가 데이터의 최소값을 벗어난 경우 맨 위로 그 값이 저장되는 현상.

ex) char 타입의 수의 범위 : -128 ~ 127

$127 + 1 = -128$  (오버플로우)

$-128 - 1 = 127$  (언더플로우)

$127 + 2 = -127$  (오버플로우)

$-128 - 2 = 126$  (언더플로우)

예제)

-소스코드

```
#include <stdio.h>
int main(void)
{
    char a = 127, b = -128;
    a++;
    b--;
    printf("Overflow : %d", a);
    printf("Underflow : %d", b);
    return 0;
}
```

-결과

Overflow:-128 Underflow:127

## 4. 아스키코드

ANSI가 정의한 미국 표준 정보 교환 코드이다. 7비트의 이진수 조합으로 만들어져 총 128개의 부호를 표현한다. 숫자와 로마글자 및 도량형 기호와 문장기호를 나타낸다.

이 아스키 코드를 사용하는 이유는 문자가 숫자로 치환이 가능하기 때문에 암호화에 효율적이다. 일상에서 무선으로 정보를 주고받는데 SDR이라는 것을 활용하면 무선으로 주고받는 데이터들을 가로챌 수 있다. 따라서 데이터에 암호화가 필요하게 되고, 암호화 한 대표적인 예시가 공인인증서이다.

이전에는 로마의 시저 장군이 군사 기밀 문서를 암호화 시키기 위해 모든 문자에 +3을 하여 받은 문서를 다시 -3해 복호화 하여 내용을 확인했다. 이 기법은 세계 1차대전, 2차대전 등에서도 많이 사용되었고 현재에도 많이 사용중이다.

-Printf 안에서 %의 사용

예제)

-소스코드

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

```
int main(void)
```

```
{
```

```
    int num1=2, num2=2;
```

```
    printf("num1 %c= num2 = %d, num1 = %d\n", 37, num1 %= num2, num1);
```

```
    return 0;
```

```
}
```

-결과

```
num1 %= num2 = 0, num1 = 2
```

위의 결과를 보게 되면 printf 안의 %를 문자로 나타낼 때의 방법이 2가지가 있다.

1) 메타 문자 방식(%%)을 사용하는 방법

2) 아스키 코드를 이용하는 방법

아스키 코드를 사용 할 때에는 %를 출력하고자 할 때 %c에 37을 넣어주면 '%'를 출력 가능하다. 다른 문자들도 해당 문자의 아스키 코드를 알면 사용 가능하다.

## 5. 기본 산술 연산자

+: 덧셈  
-: 뺄셈  
\*: 곱셈  
/: 나눗셈  
%: 나머지 연산

예제)

-소스코드

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

```
int main(void)
```

```
{  
    int num1= 11, num2=5;  
    int result = num1+num2;           →덧셈  
    printf("result=%d\n",result);  
    result=num1-num2;                 →뺄셈  
    printf("result=%d\n",result);  
    result=num1*num2;                 →곱셈  
    printf("result=%d\n",result);  
    result=num1/num2;                 →나눗셈  
    printf("result=%d\n",result);  
    result=num1%num2;                 →나머지  
    printf("result=%d\n",result);  
}
```

-결과

result=16

result=6

result=55

result=2

result=1

## 6. 전위, 후위 연산

\*postfix++ , ++prefix

postfix++ , ++prefix 모두 1을 더하는데 ++a의 경우 먼저 즉각적으로 1이 증가하고 a++인 경우 다음 문장이 실행되고 1이 증가한다.

## 7. 비트연산자

AND, OR, XOR, NOT, 쉬프트 연산

AND(&) : 서로 참으로 같을 때만 참(1) 이 된다.

OR(|) : 둘 중 하나만 1이어도 참(1)이 된다..

XOR(^) : 서로의 값이 다를 때 참(1)이 된다.

NOT(~) : 수를 반전시킨다. 0(거짓)→1(참), 1(참)→0(거짓)

<<, >> : 쉬프트 연산은 정수형에서만 가능하다. 2로 나누고 소수점 자리를 버림. 쉬프트는 나눌 때 고수점 자리를 그냥 버려버린다.

Ex)

```
1010 << 1
→ 10100 =20
1010<<3
→ 1010000 =80
```

**-AND**

10 & 3

```
    1010
    0011
=    0010 → 2
```

**-OR**

10 | 16

```
    01010
    10000
=    11010 → 26
```

**-XOR**

10 ^ 5

```
    1010
    0101
=    1111 → 15
```

10000 - 1 = 16 - 1 = 15



$10 \wedge 3$

1010  
0011  
= 1001 → 9

**-NOT**

~10

000000000....1010 NOT

111111111....0101 (반전) -X

0000000000001011 (음수화시키기) 11 = X

따라서 -X = 11

예제)

```
int main(void)
{
    int num1=10;
    printf("num1<<1 = %d\n", num1<<1);
    printf("num1<<3 = %d\n", num1<<3);
    printf("num1>>2 = %d\n", num1>>2);
    printf("num1&3 = %d\n", num1&3);
    printf("num1|16 = %d\n", num1|16);
    printf("num1^5 = %d\n", num1^5);
    printf("num1^3 = %d\n", num1^3);
    printf("~num1 %d\n", ~num1);

    return 0;
}
```

-결과

num1<<1 = 20  
num1<<3 = 80  
num1>>2 = 2  
num1&3 = 2  
num1|16 = 26  
num1^5 = 15  
num1^3 = 9  
~num1 -11

## 8. 관계연산자

관계연산자는 결과가 참(1) 혹은 거짓(0)이다.

아래와 같은 관계연산자 들이 있다.

<, >, <=, >=, ==, !=

< : 오른쪽이 왼쪽보다 크다

> : 왼쪽이 오른쪽보다 크다

<= : 오른쪽이 왼쪽보다 크거나 같다

>= : 왼쪽이 오른쪽보다 크거나 같다

== : 왼쪽과 오른쪽이 같다

!= : 왼쪽과 오른쪽이 다르다

\* =연산자와 ==연산자

C언어의 코드를 분석할 때에는 오른쪽에서 왼쪽으로 해석한다.

=는 오른쪽에 있는것을 왼쪽에 대입하는 것이고 ==는 왼쪽과 오른쪽이 같다는 뜻이다.

예를들어

```
num1 = num2 = num3;
```

Num3를 num2,num1에 셋팅하는 역할을 하는 것이고 연산자 =는 어셈블리어의 mov와 동일하다.

예제)

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

```
int main(void)
```

```
{
```

```
int num1 = 10;
```

```
printf("num1 > 5 = %d\\n", num1 > 5);
```

```
printf("num1 < 7 = %d\\n", num1 < 7);
```

```
printf("num1 == 10 = %d\\n", num1 == 10);
```

```
printf("num1 != 12 = %d\\n", num1 != 12);
```

```
printf("num1 <= 11 = %d\\n", num1 <= 11);
```

```
printf("num1 >= 9 = %d\\n", num1 >= 9);
```

```
return 0;
```

```
}
```

-결과

```
n>5=1
```

```
n<7=0
```

```
n==10=1
```

```
n!=12=1
```

```
n<=11=1
```

```
n>=9=1
```

## 9. 논리 연산자

논리 연산자는 &&, ||, ! 이 존재한다. 관계연산자와 같이 결과는 참(1) 혹은 거짓(0)이다.  
! 를 사용할 때에는 값이 있으면 거짓이고 값이 없으면 참이 된다.

예제)

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

```
int main(void)
```

```
{  
    int n1=3,n2=7;  
    int r1,r2,r3;  
  
    r1=(n1==3&& n2==7);  
    r2=(n1>100||n2<100);  
    r3=!n2;  
  
    printf("r1=%d\n",r1);  
    printf("r2=%d\n",r2);  
    printf("r3=%d\n",r3);  
  
    return 0;  
}
```

-결과

r1=1

r2=1

r3=0

## 10. 숏컷(shortcut)의 이점

기본적으로 if 문을 사용하면 mov, cmp, jmp 3 개의 어셈블리어 코드가 만들어진다.  
하지만 shortcut 을 사용하면 비교하는 cmp 가 사라져 코드 최적화 용도로 사용된다.  
특히 ARM으로 구현할 때 더 많은 이득을 볼 수 있다.

## 11. 연산자 예제

1. 입력을 6 을 주고 num << 4 를 수행하는 함수를 작성하고 결과를 출력하도록 만든다.

```
#include<stdio.h>

int func1(int n)
{
    return n<<4;
}

int main(void)
{
    int n=6,r;
    r= func1(n);
    printf("r= %d\\n",r);
    return 0;
}
```

결과 r=96

2. 입력에 55 를 넣고 num >> 3 을 수행하는 함수 작성

```
#include<stdio.h>

int func2(int n)
{
    return n>>3;
}

int main(void)
{
    int n=55,r;
    r= func2(n);
    printf("r= %d\\n",r);
}
```

결과

55(10)=32+16+4+2+1 = 110111

110111

r= 6

3. 입력에 char num1 = 21, num2 = 31 을 넣고  
AND, OR, XOR 를 수행하는 함수를 각각 만든다.  
(함수가 총 3 개 만들어짐 main 포함 4 개)

```
#include<stdio.h>

int f1(char n1, char n2)
{
    return n1&n2;
}
int f2(char n1, char n2)
{
    return n1|n2;
}
int f3(char n1, char n2)
{
    return n1^n2;
}
int main(void)
{
    char n1=21,n2=31;
    int r;
    r=f1(n1,n2);
    printf("r=%d\n",r);
    r=f2(n1,n2);
    printf("r=%d\n",r);
    r=f3(n1,n2);
    printf("r=%d\n",r);

    return 0;
}
```

-결과

r=21

r=31

r=10

-계산

21(10) = 010101

31(10)=011111

AND = 010101

OR = 01111

XOR= 001010

## 12. Scanf

scanf 의 첫 번째 입력은 "%d" 혹은 "%f", "%lf" 등이 올 수 있다.

그리고 두 번째 입력은 반드시 결과를 받을 주소값을 적어야 한다.

그렇기 때문에 주소값을 의미하는 & 가 들어간 것이다.

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

```
int main(void)
```

```
{
```

```
    int n1;
```

```
    float n2;
```

```
    printf("int:");
```

```
    scanf("%d",&n1);
```

```
    printf("float");
```

```
    scanf("%f",&n2);
```

```
    printf("data: %d,%f\n",n1,n2);
```

```
    return 0;
```

```
}
```

-결과

int:10

float:2.5

data: 10,2.500000

\*multiple data with scanf

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

```
int main(void)
```

```
{
```

```
    int n1, n2;
```

```
    double r1,r2;
```

```
    printf("2개의 정수를 입력하시오:");
```

```
    scanf("%d,%d",&n1,&n2);
```

```
    printf("%d,%d\\n",n1,n2);
```

```
    printf("2개의 소수를 입력하시오:");
```

```
    scanf("%lf,%lf",&r1,&r2);
```

```
    printf("%.3lf,%.3lf\\n",r1,r2);
```

```
    return 0;
```

```
}
```

-결과

2개의 정수를 입력하시오:2,1

2,1

2개의 소수를 입력하시오:2.1,1.1

2.100,1.100

## 13. if문

if 문은 조건을 지정하고 싶을 때 사용한다.

이 안에는 관계 연산자, 조건 연산자 등등이 올 수 있다.

(복잡하게 생각하지 말고 어떤 조건을 만족하는지 안하는지를 생각하면 됨)

그리고 if(조건) 에서 조건이 만족되지 않을 경우에는

else 쪽으로 이동하게 되고

만약 여기서도 추가 조건을 비교하고자 한다면

else if(또다른조건) 의 형식으로 여러 조건을 만들 수 있다.

예제)

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

```
int main(void)
```

```
{
```

```
    int num;
```

```
    printf("정수를 입력하시오:");
```

```
    scanf("%d",&num);
```

```
    if(num>=0)
```

```
    {
```

```
        printf("양수\n");
```

```
    }
```

```
    if(num<0)
```

```
    {
```

```
        printf("음수\n");
```

```
    }
```

```
    return 0;
```

```
}
```

-결과

정수를 입력하시오:3

양수



## 14. switch문

if 문을 많이 사용하게되면 가독성이 떨어지고 코드가 무거워지게 된다. if문을 많이 사용해야 할 때에는 switch문을 사용하면 좋은데 switch는 case부분에 조건이 있어 case에 해당하는 조건만 보고 해당 부분만을 확인하면 되어 가독성이 올라간다.  
case안에서의 동작은 break를 만나기 전까지 계속된다.

예제)

```
#include <stdio.h>
int main(void)
{
    int num;
    printf("상담은 1, 사용 내역 조회는 2, 계약 해지는 3을 눌러주세요 : ");
    scanf("%d", &num);
    switch(num)
    {
        case 1:
            printf("상담 센터에 연결중입니다\n");
            break;
        case 2:
            printf("사용 내역을 조회합니다\n");
            break;
        case 3:
            printf("계약 해지를 진행합니다\n");
            break;
        default:
            printf("번호를 잘못 누르셨습니다\n");
    }
    return 0;
}
```

## 15. while문

if 나 switch 와 같이 조건을 활용하여 작업을 반복시킬 때 사용한다.  
조건이 참(1)인 동안 반복하게되고 조건이 0이면 작업을 반복하지 않는다.  
While(1)을 이용하여 무한반복 루프를 만들 수 있다.

예제1)

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

```
int main(void)
```

```
{
```

```
    int i=0, r='A';
```

```
    while(i<10)
```

```
    {
```

```
        printf("%c\\n", r);
```

```
        r++;
```

```
        i++;
```

```
    }
```

```
    return 0;
```

```
}
```

-결과

A

B

C

D

E

F

G

H

I

J

## 예제2

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

```
void print_even(int start, int end)
{
    int i= start;
    while(i<=end)
    {
        if(!(i%2))
        {
            printf("even=%d\n",i);
        }
        i++;
    }
}
```

```
int main(void)
{
    print_even(1,100);
    return 0;
}
```

-결과

```
even=2
even=4
even=6
even=8
even=10
even=12
even=14
even=16
even=18
even=20
even=22
even=24
even=26
even=28
even=30
even=32
even=34
```

even=36  
even=38  
even=40  
even=42  
even=44  
even=46  
even=48  
even=50  
even=52  
even=54  
even=56  
even=58  
even=60  
even=62  
even=64  
even=66  
even=68  
even=70  
even=72  
even=74  
even=76  
even=78  
even=80  
even=82  
even=84  
even=86  
even=88  
even=90  
even=92  
even=94  
even=96  
even=98  
even=100  
)

## 16. 과제

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)

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

```
int f1(int i)
{
    int r;
    if(3<=i)
    {
        r= i*37500*0.8;
    }else
    {
        r= i*37500;
    }
    return r;
}
```

```
int main(void)
{
    int i;
    printf("number of days:");
    scanf("%d",&i);

    printf("charge:%d₩n",f1(i));

    return 0;
}
```

3. 1 ~ 1000사이에 3의 배수의 합을 구하시오.

```
#include<stdio.h>

double f1(int s, int e)
{
    double r;

    while(s<=e)
    {
        if(s%3 == 0)
        {
            r=s+r;
        }
        s++;
    }

    return r;
}

double main(void)
{
    int s=1,e=1000;
    double r2;
    r2= f1(s,e);
    printf("vlaue:%lf",r2);

    return 0;
}
~
166833.000000
```

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

```
#include<stdio.h>

double f1(int s, int e)
{
    double r;

    while(s<=e)
    {
        if((s%4 ==1)||(s%6==1))
        {
            r=s+r;
        }
        s++;
    }

    return r;
}

double main(void)
{
    int s=1,e=1000;
    double r2;
    r2= f1(s,e);
    printf("vlaue:%lf",r2);

    return 0;
}
```

-결과

166167

5. 7의 배수로 이루어진 값들이 나열되어 있다고 가정한다.

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

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

```
void f1(int i)
{
    printf("%d", 7*i);
}
```

```
int main(void)
{
    int i;
    scanf("%d",&i);

    f1(i);
    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;
}
```

- 어셈블리어 코드

(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
0x000000000040051f <+59>:    retq
```

End of assembler dump.

(gdb) disas

Dump of assembler code for function main:

```
=> 0x0000000004004e4 <+1(1): push %rbp
0x0000000004004e5 <+1(2): mov %rsp,%rbp
0x0000000004004e8 <+4(3): sub $0x10,%rsp
0x0000000004004ec <+8(4): movl $0x0,-0x8(%rbp)
0x0000000004004f3 <+1(5): movl $0x0,-0xc(%rbp)
0x0000000004004fa <+2(6): jmp 0x400506 <main+34>
0x0000000004004fc <+2(9): mov -0xc(%rbp),%eax
0x0000000004004ff <+27(10): add %eax,-0x8(%rbp)
0x000000000400502 <+3(11): addl $0x1,-0xc(%rbp)
0x000000000400506 <+3(7): cmpl $0x4,-0xc(%rbp)
0x00000000040050a <+3(8): jle 0x4004fc <main+24>
0x00000000040050c <+4(12): mov -0x8(%rbp),%eax
0x00000000040050f <+4(13): mov %eax,%edi
0x000000000400511 <+4(14): callq 0x4004d6 <mult2>
0x000000000400516 <+5(15): mov %eax,-0x4(%rbp)
0x000000000400519 <+5(16): mov $0x0,%eax
0x00000000040051e <+5(17): leaveq
0x00000000040051f <+5(18): retq
End of assembler dump.
```

복귀주소



C P U

ax	4(9) > 4(12) > 0(16)
edi	4(13)
sp	
bp	
ip	

10. 구구단을 만들어보시오.

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

```
void f1(int s,int e)
{
    int r=0,r2;
    while(s<=e)
    {
        while(r<=9)
        {
            r2=r*s;
            printf("%d * %d = %d\\n",s,r,r2);
            r++;
        }
        s++;
        r=0;
    }
}
```

```
int main(void)
{
    f1(0,9);

    return 0;
}
```

-결과

0 \* 0 = 0  
0 \* 1 = 0  
0 \* 2 = 0  
0 \* 3 = 0  
0 \* 4 = 0  
0 \* 5 = 0  
0 \* 6 = 0  
0 \* 7 = 0  
0 \* 8 = 0  
0 \* 9 = 0  
1 \* 0 = 0  
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 \* 0 = 0  
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 \* 0 = 0  
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 \* 0 = 0  
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 \* 0 = 0  
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 \* 0 = 0  
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 \* 0 = 0  
7 \* 1 = 7  
7 \* 2 = 14  
7 \* 3 = 21  
7 \* 4 = 28  
7 \* 5 = 35  
7 \* 6 = 42  
7 \* 7 = 49  
7 \* 8 = 56  
7 \* 9 = 63  
8 \* 0 = 0  
8 \* 1 = 8  
8 \* 2 = 16  
8 \* 3 = 24  
8 \* 4 = 32  
8 \* 5 = 40  
8 \* 6 = 48  
8 \* 7 = 56  
8 \* 8 = 64  
8 \* 9 = 72  
9 \* 0 = 0  
9 \* 1 = 9  
9 \* 2 = 18  
9 \* 3 = 27  
9 \* 4 = 36  
9 \* 5 = 45  
9 \* 6 = 54  
9 \* 7 = 63  
9 \* 8 = 72  
9 \* 9 = 81

12. 리눅스에서 디버깅 하는 방법 정리, gdb상에서 아직 소개하지 않은 명령들 bt, c 이 2개에 대해 조사하고 활용해보자

-디버깅 하는 법

- 1) 소스코드를 짜고 저장한다.
- 2) 소스코드가 저장된 디렉토리에 들어가 ls를 눌러 소스코드가 있는걸 확인한다.
- 3) 최적화를 못하게 하는 디버깅 실행한다. 'gcc -g -O0 -o debug func1.c'
- 4) ls를 입력하여 debug가 존재하는지 확인한다.
- 5) 어셈블리어를 확인하고 싶을 때에는'gdb debug'를 입력하여 디버거를 켜다.
- 6) 'b main'을 입력하여 main 함수에 breakpoint(정지)를 걸어준다.
- 7) 'r'을 눌러 실행하고 disas를 입력하여 어셈블리어 코드를 확인한다.

-bt, c 조사

7번 문제에서의 gdb에서 bt와 c 명령어를 입력했을 때의 결과이다.

(gdb) bt

#0 main () at test.c:15

(gdb) c

Continuing.

Breakpoint 1, main () at test.c:17

17 int i, sum = 0, result;

(gdb)

bt는 전체 스택 프레임을 출력한다.

c는 컨티뉴 명령인데 브레이크 포인트가 걸려있는 지점부터 다시 실행하는 명령어이다. 브레이크 포인트의 위치와 변수선언 한 것들을 보여준다.