2018. 2. 26 월 <4 회차>

1. <3 회차 숙제확인>
2. 디버그
3. SCOPE
4. Static Statement
5. Continue Statement
6. Do while Statement
7. #define A B
8. for 문
9. 무한루프
10. goto Statement
11. SW 와 HW 의 차이
12. 파이프라인
13. 재귀함수

1. <3 회차 숙제확인>
생략

2. 디버그

디버그 - 어셈블리 디버그(disas 로 진입)
- C 레벨에서의 디버그(gdb 로 그냥 진입)

	함수내부로 진입	함수가 있어도 들어가지 않음
어셈블리 디버그	si	ni
C 레벨에서의 디버	s	n

디버깅을 하는 이유(2 가지)
1. 문법적인 오류 확인 : 컴파일이 성공적이지 못한 경우
2. 논리적인 오류 확인 : 컴파일은 성공적이나 논리적으로 원하는 값이 나오지 않는 경우
즉, 논리적인 오류인 경우는 동작은 하나 왜 이상한 동작을 하는지 알기위해 하는 것

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

int main(void)
{
   int number = 1;
   while(1)
   {
      printf("%d\n", number);
      number += number;
      if(number > 100)
         break;
   }
```

```
    return 0;
}
```

문제점. 위의 경우, number += number 는 2^n 이 되어버린다.
해결책. int number = 0; - 변경
        number ++; - 추가

참고.
1. (gdb)에서 c : c 는 continue(다음 breakpoint 만날때까지)
2. Enter : 앞에 실행했던 명령문 반복업
3. l : list 의 약자로 C 코드 확인가

3. Scope
C 언어에서 if, while, main, 함수 사용시
{}부분을 말함
구조적으로 stack frame 의 생성과 해제 - stack 이 관리, 안에서 지역변수를 관리가
{ : 생성능
} : 해제

Ex>

```
#include <stdio.h>

int main(void)
{
    int global_area = 1;

    {
      int local_area1 = 2;
      printf("global_area = %d\n", global_area);
      printf("local_area1 = %d\n", local_area1);
    }

    {
      int local_area2 = 3;
      printf("global_area = %d\n", global_area);
      printf("local_area2 = %d\n", local_area2);
    }

    printf("global_area = %d\n", global_area);


    //scope 개념
    //printf("local_area1 = %d\n, local_area1);
    //printf("local_area2 = %d\n, local_area2);   위 두 문장을 줄 경우, 동작 x(main 함수에서 local_area1,2
에 대한 변수를 주지 않았으므로 동작하지 않는다.)
    //위 두문장을 동작이 되게 하려면 main 함수 또는 외부에서 설정을 해주어야한다.

    return 0;
}
```

위의 예제처럼 {}안의 변수를 밖에서 코딩한 경우 동작 x
전역변수(global variable) : int main(void)위에 작성하는 변수
stack 에 상관없이 어느 Scope 에서도 적용
data 에 저장

4. Static Statement – 정적변수
강제로 전역변수로 만들어버림
전역변수와의 차이점 : 정적변수는 이 변수를 선언한 함수내에서만 접근가능함(Scope 외에서 적용 x)

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

void count_static_value(void)
{
   static int count = 1;
   printf("count = %d\n", count);
   count++;
}

int main(void)
{
   int i;
   for(i = 0; i < 7; i++)
     count_static_value();

   return 0;

}

// static 을 사용하면, scope 내에 있어도 전역변수의 역할을 한다.
//                  강제적으로 전역변수가 되어 count++이 7 까지 진행된다.
// static 이 없을경우, count 는 1 로 고정
```

5. Continue Satatement
사용하는 이유 - 해당 case 를 제끼고 반복은 계속하고 싶어서

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

//contine 예제. number == 5 인 경우는 제외하고 출력
//0 : 거짓, 1 : 참   -    while(1)인 경우는 계속 반복
//                      while(num<10) 에서 0 이 되면 동작이 정지


int main(void)
{
  int number = 0;

   while(1)
  {
```

```
        number++;

        if(number == 5)
                continue;

        printf("%d\n", number);

        //contine 가 이 쪽에 위치하면 printf 에서 이미 5 가 나와서 의미가 없음

        if(number == 10)
                break;
    }
    return 0;
}
```

continue 로 인하여 number 가 5 일때는 뛰어넘고 동작.

참고.
1. NaN : 어떤수를 0 으로 나눔
2. Inf : infinity

6. do while Sratement
사용하는 이유 - 최소 1 번은 작업, 반복은 할 수도 안 할 수도 있음
주로 kernel 매크로에 사용

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

/* do while 예제.
*/

int main(void)
{
    int number = 0;

    do
    {
        number++;
        if(number ==5)
            continue;
        printf("%d\n", number);
    } while(number < 10);

    return 0;
}
```

7. #define A B
B 가 길거나하는 등의 이유로 B 를 A 로 치환
또는 단어나 문장을 수많이 루프시키는 경우, 코드수정이 필요하면 치환으로 수

Ex> 수업필기참고(4 일차)

```
// #define 예제.
정
#include <stdio.h>

#define TEST 1000

int main(void)
{
	printf("result = %d\n", TEST);
	return 0;
}
```

8. for 문

사용하는 이유 - while 을 대체하려고 만듬
while 초기화, 조건식, 증감식 3 개 짜리를 for(초기화; 조건식; 증감식) 한 문장으로 감축

Ex>

```
#include <stdio.h>

// result = 'A'에서 A 로 작성하면 error
// for 예제.   for(i = 0, result = 'A') 대신에 변수선언시 i = 0, result = 'A'가능)
// widows 에서 c 언어 사용시 변수미리 지정해주어야 된다. 위 문장에서 후자선택

int main(void)
{
  int i, result;

  for(i = 0, result = 'A'; i < 10; i++, result++)
  {
    printf("%c\n", result);
  }
  return 0;
}
```

9. 무한루프
형태 : for(;;)
;;의 사이가 비어있는 경우, 무한루프(양 사이드에는 어떤 것이 있어도 상관 x)
무한루프 종료방법 : ctrl +c

Ex>

```
#include <stdio.h>

//for(;;) 무한루프 예제.
//무한루프 끄는 법 : ctrl + c

int main(void)
{
   int i, result = 'A';

   for(;;)
```

```
    {
      printf("%c\n", result);
    }
    return 0;
}
```

문제점. int i 는 오타

10. goto Statement
주로 kernel 에서 사용
Buffer 사용하는 곳에서 주로 사용함

Ex> 수업필기첨부

```
#include <stdio.h>

//goto 대신에 if 를 사용한 예제. goto 나 if 와 break 를 사용하면 Error 에서 멈추지만,  if 만 사용하면 Error 가 나와도 계속 끝까지 진행해버린다.
//결국 하고자 하는 말은 goto 사용할 것
//jmp 명령어는 CPU 에 매우 안좋음
//goto 의 이점과 CPU 파이프라인
/*
앞서서 만든 goto 예제는
if 와 break 를 조합한 버전과
goto 로 처리하는 버전을 가지고 있다.

if 문은 기본적은로 mov, cmp, jmp 로 구성된다.
goto 는 jmp 하나로 끝이다.

for 문이 여러개 생기면 if, break 조합의 경우
for 문의 갯수만큼 mov, cmp, jmp 를 해야한다.
문제는 바로 jmp 명령어이다.

call 이나 jmp 를 CPU instruction(명령어) 레벨에서
분기 명령어라고 하고 이들은
CPU 파이프라인에 매우 치명적인 손실을 가져다준다.

기본적으로 아주 단순한
CPU 의 파이프라인을 설명하자면
아래와 같은 3 단계로 구성된다.

1. Fetch - 실행해야할 명령어를 물어옴
2. Decode - 어떤 명령어인지 해석함
3. Executr - 실제 명령어를 실행시킴

파이프라인이 짧은 것부터 긴 것이
5 단계 ~ 수십 단계로 구성된다.
(ARM, Intel 등등 다양한 프로세서들 모두 마찬가지)
```

그런데 왜 jmp 나 call 등의 분기 명령어가 문제가 될까?

기본적으로 분기 명령어는 파이프라인을 때려부순다.
이 뜻은 가장 단순한 CPU 가 실행가지 3clock 을 소요하는데
파이프라인이 깨지니 쓸데없이 또 다시 3 clock 을 버려야함을 의미한다.
(만약 파이프라인의 단계가 수십단계라면
분기가 여러번 발생하면
파이프라인 단계 x 분기 횟수만큼
CPU clock 을 낭비하게 된다.
즉 성능면에서도 goto 가 월등히 압도적이다.
(jmp 1 번에 끝나니까)

*/

```
int main(void)
{
  int i, j, k;

  for(i = 0; i < 5; i++)
  {
    for(j = 0; j < 5; j++)
    {
      for(k = 0; k < 5; k++)
      {
        if((i == 2) && (j ==2) && (k == 2))
        {
          printf("Error!!!\n");
          goto err_handler;
        }
        else
        {
          printf("Data\n");
        }
      }
    }
  }
  return 0;

err_handler:
   printf("Goto Zzang!\n");

   return -1;

}
```

Ex>비교예제 – if 1 번 사용, Error 가 나와도 계속 진행

```
#include <stdio.h>

//goto 대신에 if 를 사용한 예제. goto 나 if 와 break 를 사용하면 Error 에서 멈추지만,  if 만 사용하면 Error 가 나와도 계속 끝까지 진행해버린다.
```

```
int main(void)
{
  int i, j, k;

  for(i = 0; i < 5; i++)
  {
    for(j = 0; j < 5; j++)
    {
      for(k = 0; k < 5; k++)
      {
        if((i == 2) && (j ==2) && (k == 2))
        {
          printf("Error!!!\n");
        }
        else
        {
          printf("Data\n");
        }
      }
    }
  }
  return 0;
}
```

Ex> 비교예제 2 – if 와 break 를 for 문 갯수만큼 작성

```
#include <stdio.h>

//goto 대신에 if 와 break 를 사용한 예제. goto 는 전체에 한 번만 사용하나, if 와 break 를  사용할 경우 for 문 개수만큼 설정해주어야 한다.

int main(void)
{
	int i, j, k, flag;

	for(i = 0; i < 5; i++)
	{
		for(j = 0; j < 5; j++)
		{
			for(k = 0; k < 5; k++)
			{
				if((i == 2) && (j ==2) && (k == 2))
				{
					printf("Error!!!\n");
					flag = 1;
				}
				else
				{
					printf("Data\n");
				}

				if(flag)
				{
```

```
                                        break;
                                }
                        }
                        if(flag)
                        {
                                break;
                        }
                }
                if(flag)
                {
                        break;
                }



        }
}
```

11. SW 와 HW 의 차이

수업필기첨부

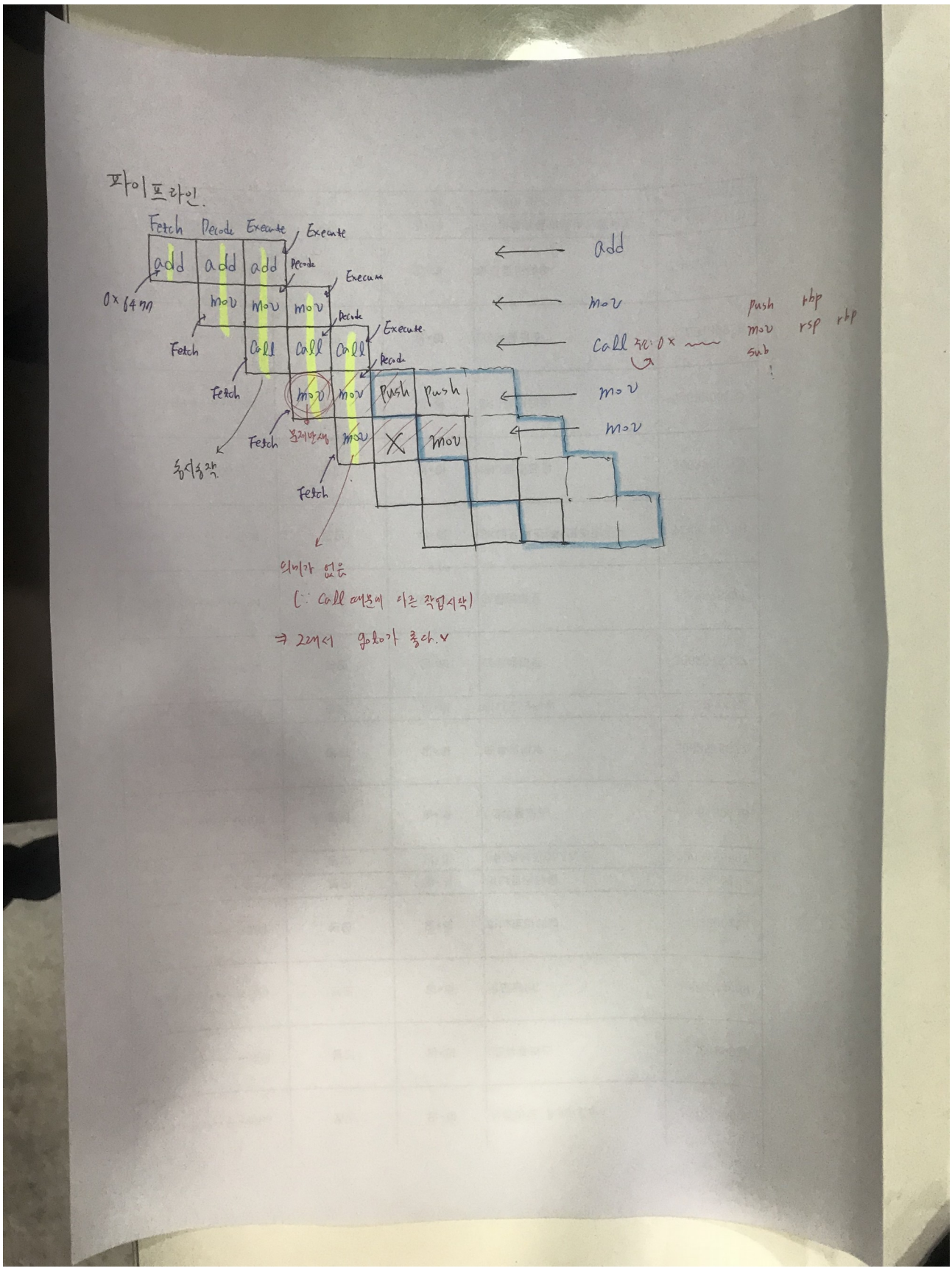
```
/*
 sw 와 hw 동작을 생각할 때 주의할 점

-sw
 멀티 코어 상황이 아니면
 어떤 상황에서도
 한 번에 한 가지 동작만 실행할 수 있음

 좀 더 정확하게 이야기 하자면
 CPU 한 개는
 오로지 한 순간에 한 가지 동작만 할 수 있음

-HW
 반면 HW 회로는 병렬회로가 존재하듯이
모든 회로가 동시에 동작할 수 있다.
파이프라인은 CPU 에 구성된 회로이기 때문에
모든 모듈들이 동시에 동작할 수 있다.
(FPGA 프로그래밍은 병렬 동작임 - FPGA 는 그리하여 실제로 회로만들 때 사용)
*/
```

12. 파이프라인


파이프라인
Fetch
Decode
Execute
add
mov
call
push
0x64까
의미가 없음
(: call 때문에 다른 작업시작)
⇒ 2개씩 go하기 좋다.

13. 재귀함수

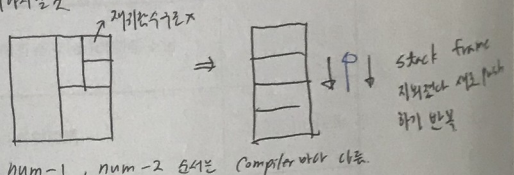
재귀함수 : 자기가 자기자신을 호출

형태 ⇒ 함수안에 함수(똑같은)가 있는 형태.

사용이유 : 편리. But, 무조건 좋지 않은 (∵ 파이프라인, 스택프레임 사용)

참고 : 모든 자료구조는 재귀함수 호출입이 재현가능.
재귀함수는 푸는 것이 고급기술.

↳ 오해하지말고

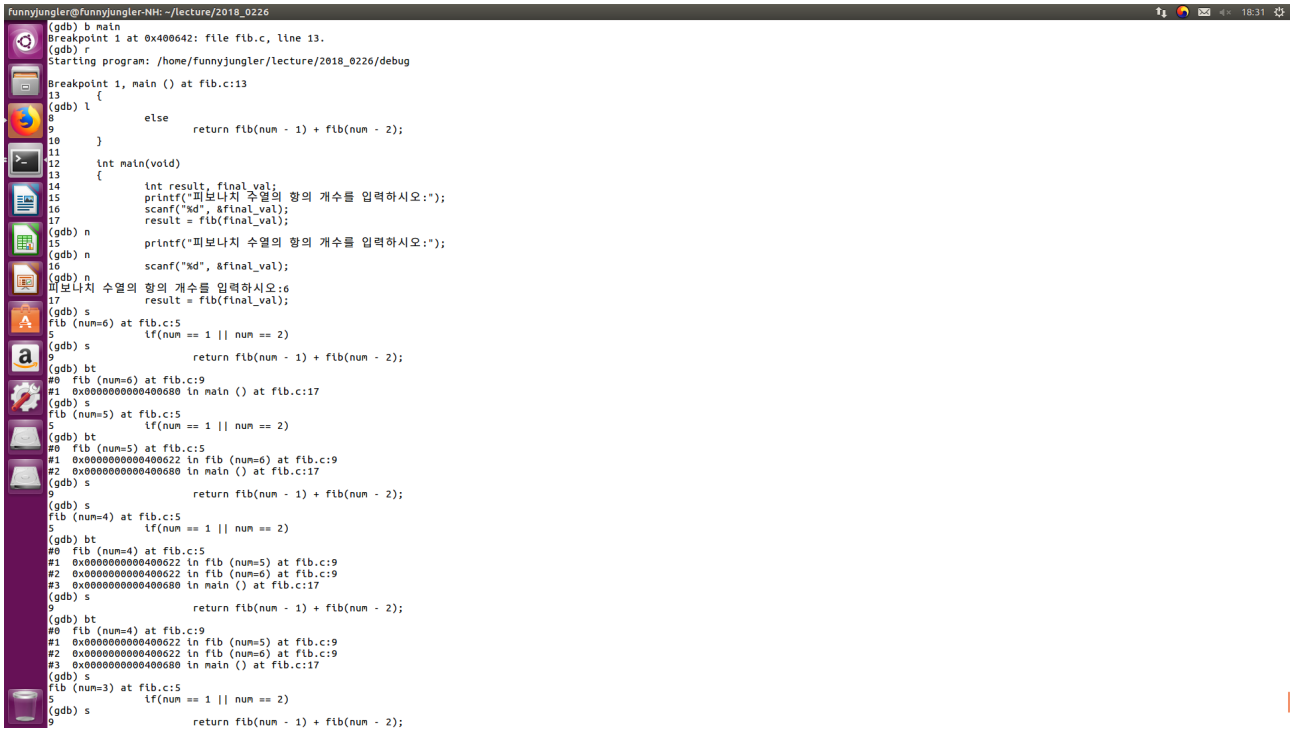


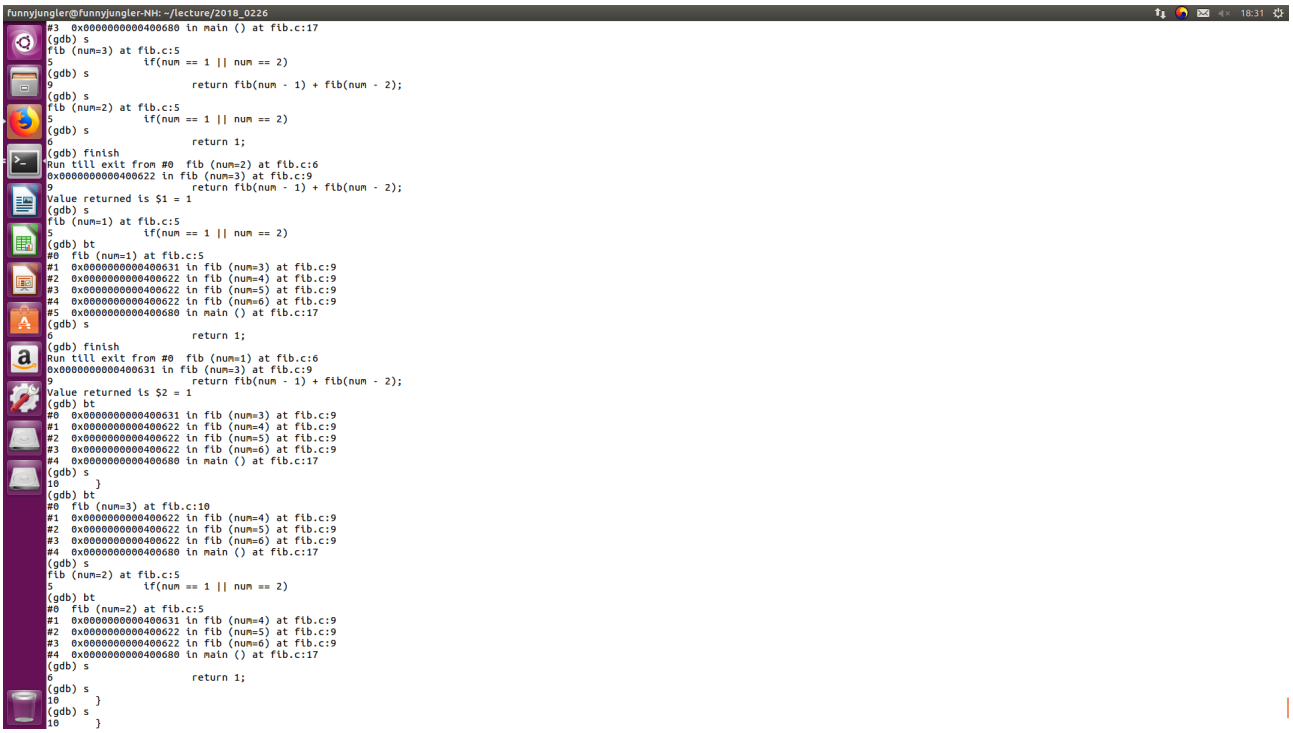
참고. num-1 , num-2 등식은 compiler에 따름.

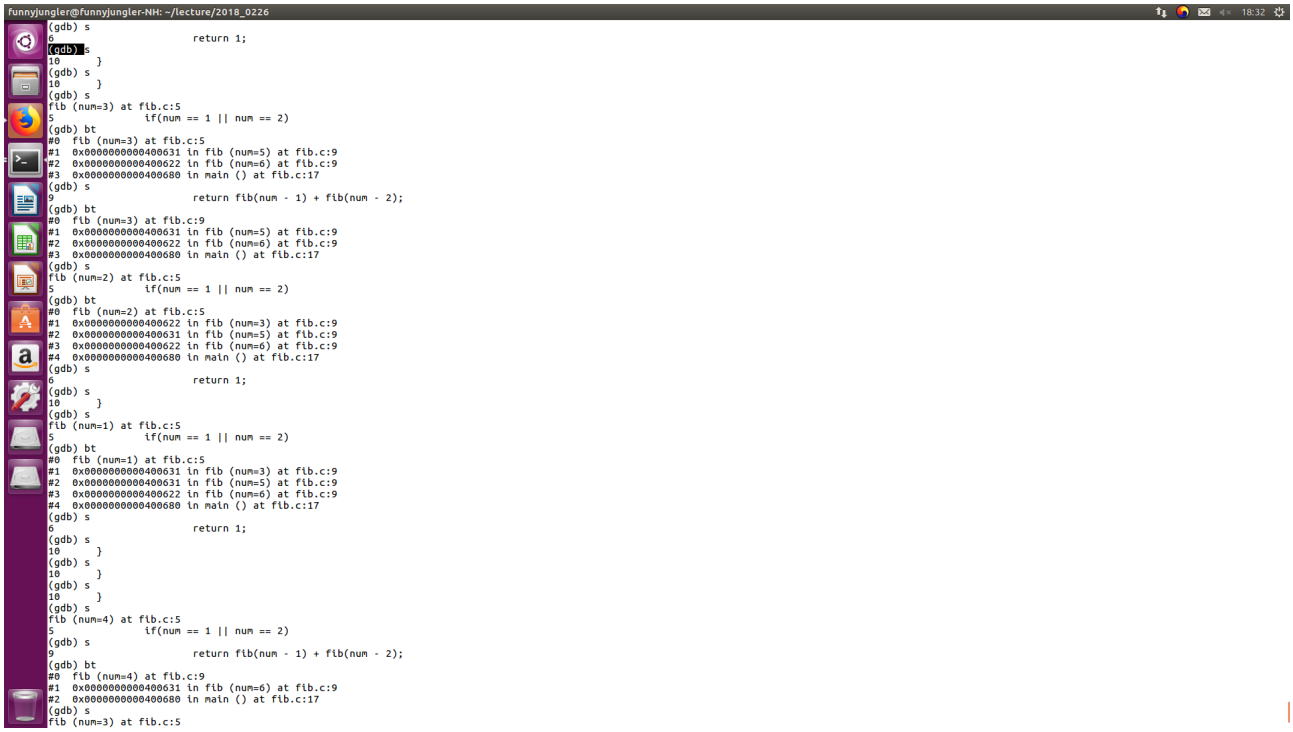
Ex) '피보나치 수열' (코 면접)

* ax : 함수의 리턴값.
shell clear : ~~de~~ gdb에서 clear
b # : 확인.
finish : return 마무리.

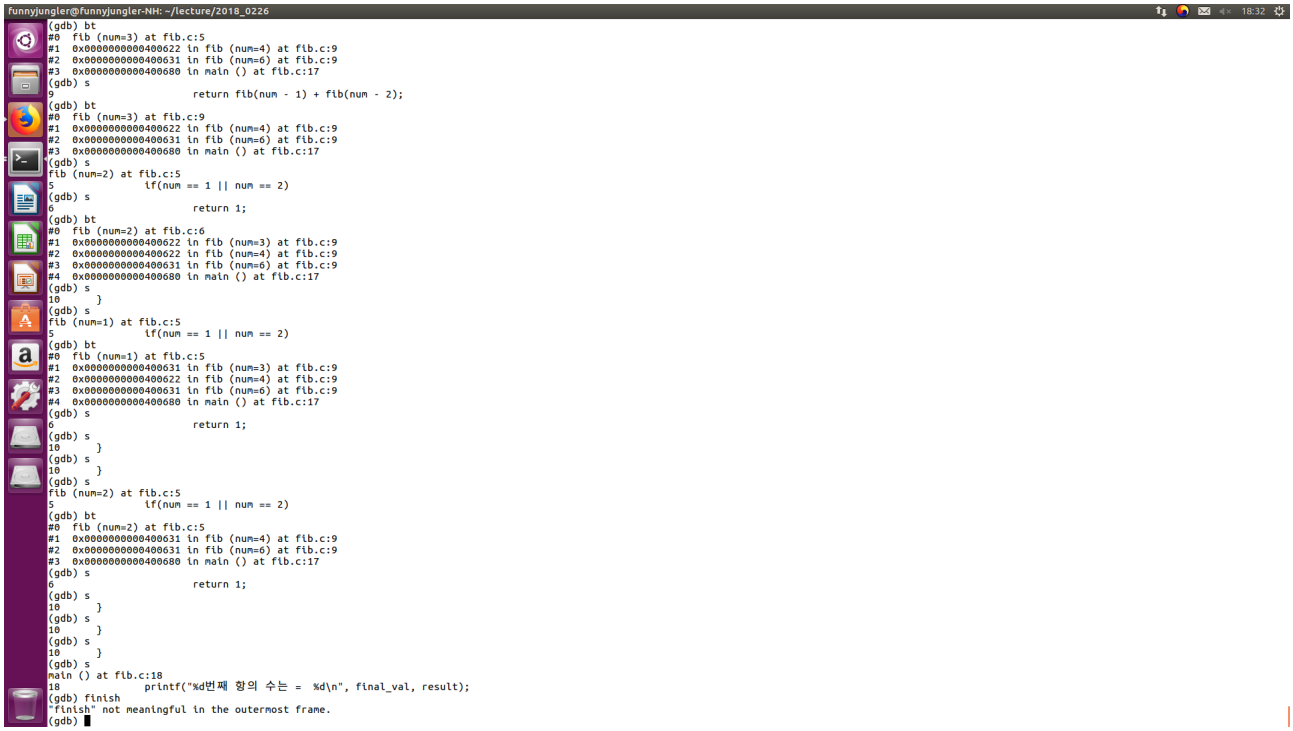
재귀함수 디버깅


funnyjungler@funnyjungler-NH: ~/lecture/2018_0226
18:31
(gdb) b main
Breakpoint 1 at 0x400642: file fib.c, line 13.
(gdb) r
Starting program: /home/funnyjungler/lecture/2018_0226/debug
Breakpoint 1, main () at fib.c:13
13 {
(gdb) l
8 else
9 return fib(num - 1) + fib(num - 2);
10 }
11
12 int main(void)
13 {
14 int result, final_val;
15 printf("피보나치 수열의 항의 개수를 입력하시오:");
16 scanf("%d", &final_val);
17 result = fib(final_val);
(gdb) n
15 printf("피보나치 수열의 항의 개수를 입력하시오:");
(gdb) n
16 scanf("%d", &final_val);
(gdb) n
피보나치 수열의 항의 개수를 입력하시오:6
17 result = fib(final_val);
(gdb) s
fib (num=6) at fib.c:5
5 if(num == 1 || num == 2)
(gdb) s
9 return fib(num - 1) + fib(num - 2);
(gdb) bt
#0 fib (num=6) at fib.c:9
#1 0x0000000000400680 in main () at fib.c:17
(gdb) s
fib (num=5) at fib.c:5
5 if(num == 1 || num == 2)
(gdb) bt
#0 fib (num=5) at fib.c:5
#1 0x0000000000400622 in fib (num=6) at fib.c:9
#2 0x0000000000400680 in main () at fib.c:17
(gdb) s
9 return fib(num - 1) + fib(num - 2);
(gdb) s
fib (num=4) at fib.c:5
5 if(num == 1 || num == 2)
(gdb) bt
#0 fib (num=4) at fib.c:5
#1 0x0000000000400622 in fib (num=5) at fib.c:9
#2 0x0000000000400622 in fib (num=6) at fib.c:9
#3 0x0000000000400680 in main () at fib.c:17
(gdb) s
9 return fib(num - 1) + fib(num - 2);
(gdb) bt
#0 fib (num=4) at fib.c:9
#1 0x0000000000400622 in fib (num=5) at fib.c:9
#2 0x0000000000400622 in fib (num=6) at fib.c:9
#3 0x0000000000400680 in main () at fib.c:17
(gdb) s
fib (num=3) at fib.c:5
5 if(num == 1 || num == 2)
(gdb) s
9 return fib(num - 1) + fib(num - 2);


funnyjungler@funnyjungler-NH: ~/lecture/2018_0226
18:31
#3 0x0000000000400680 in main () at fib.c:17
(gdb) s
fib (num=3) at fib.c:5
5 if(num == 1 || num == 2)
(gdb) s
9 return fib(num - 1) + fib(num - 2);
(gdb) s
fib (num=2) at fib.c:5
5 if(num == 1 || num == 2)
(gdb) s
6 return 1;
(gdb) finish
Run till exit from #0 fib (num=2) at fib.c:6
0x0000000000400622 in fib (num=3) at fib.c:9
9 return fib(num - 1) + fib(num - 2);
Value returned is $1 = 1
(gdb) s
fib (num=1) at fib.c:5
5 if(num == 1 || num == 2)
(gdb) bt
#0 fib (num=1) at fib.c:5
#1 0x0000000000400631 in fib (num=3) at fib.c:9
#2 0x0000000000400622 in fib (num=4) at fib.c:9
#3 0x0000000000400622 in fib (num=5) at fib.c:9
#4 0x0000000000400622 in fib (num=6) at fib.c:9
#5 0x0000000000400680 in main () at fib.c:17
(gdb) s
6 return 1;
(gdb) finish
Run till exit from #0 fib (num=1) at fib.c:6
0x0000000000400631 in fib (num=3) at fib.c:9
9 return fib(num - 1) + fib(num - 2);
Value returned is $2 = 1
(gdb) bt
#0 0x0000000000400631 in fib (num=3) at fib.c:9
#1 0x0000000000400622 in fib (num=4) at fib.c:9
#2 0x0000000000400622 in fib (num=5) at fib.c:9
#3 0x0000000000400622 in fib (num=6) at fib.c:9
#4 0x0000000000400680 in main () at fib.c:17
(gdb) s
10 }
(gdb) bt
#0 fib (num=3) at fib.c:10
#1 0x0000000000400622 in fib (num=4) at fib.c:9
#2 0x0000000000400622 in fib (num=5) at fib.c:9
#3 0x0000000000400622 in fib (num=6) at fib.c:9
#4 0x0000000000400680 in main () at fib.c:17
(gdb) s
fib (num=2) at fib.c:5
5 if(num == 1 || num == 2)
(gdb) bt
#0 fib (num=2) at fib.c:5
#1 0x0000000000400631 in fib (num=4) at fib.c:9
#2 0x0000000000400622 in fib (num=5) at fib.c:9
#3 0x0000000000400622 in fib (num=6) at fib.c:9
#4 0x0000000000400680 in main () at fib.c:17
(gdb) s
6 return 1;
(gdb) s
10 }
(gdb) s
10 }


funnyjungler@funnyjungler-NH: ~/lecture/2018_0226
18:32
(gdb) s
6   return 1;
(gdb) s
10   }
(gdb) s
10   }
(gdb) s
fib (num=3) at fib.c:5
5   if(num == 1 || num == 2)
(gdb) bt
#0 fib (num=3) at fib.c:5
#1 0x0000000000400631 in fib (num=5) at fib.c:9
#2 0x0000000000400622 in fib (num=6) at fib.c:9
#3 0x0000000000400680 in main () at fib.c:17
(gdb) s
9   return fib(num - 1) + fib(num - 2);
(gdb) bt
#0 fib (num=3) at fib.c:9
#1 0x0000000000400631 in fib (num=5) at fib.c:9
#2 0x0000000000400622 in fib (num=6) at fib.c:9
#3 0x0000000000400680 in main () at fib.c:17
(gdb) s
fib (num=2) at fib.c:5
5   if(num == 1 || num == 2)
(gdb) bt
#0 fib (num=2) at fib.c:5
#1 0x0000000000400622 in fib (num=3) at fib.c:9
#2 0x0000000000400631 in fib (num=5) at fib.c:9
#3 0x0000000000400622 in fib (num=6) at fib.c:9
#4 0x0000000000400680 in main () at fib.c:17
(gdb) s
6   return 1;
(gdb) s
10   }
(gdb) s
fib (num=1) at fib.c:5
5   if(num == 1 || num == 2)
(gdb) bt
#0 fib (num=1) at fib.c:5
#1 0x0000000000400631 in fib (num=3) at fib.c:9
#2 0x0000000000400631 in fib (num=5) at fib.c:9
#3 0x0000000000400622 in fib (num=6) at fib.c:9
#4 0x0000000000400680 in main () at fib.c:17
(gdb) s
6   return 1;
(gdb) s
10   }
(gdb) s
10   }
(gdb) s
10   }
(gdb) s
fib (num=4) at fib.c:5
5   if(num == 1 || num == 2)
(gdb) s
9   return fib(num - 1) + fib(num - 2);
(gdb) bt
#0 fib (num=4) at fib.c:9
#1 0x0000000000400631 in fib (num=6) at fib.c:9
#2 0x0000000000400680 in main () at fib.c:17
(gdb) s
fib (num=3) at fib.c:5


funnyjungler@funnyjungler-NH: ~/lecture/2018_0226
18:32
(gdb) bt
#0 fib (num=4) at fib.c:9
#1 0x0000000000400631 in fib (num=6) at fib.c:9
#2 0x0000000000400680 in main () at fib.c:17
(gdb) s
fib (num=3) at fib.c:5
5   if(num == 1 || num == 2)
(gdb) bt
#0 fib (num=3) at fib.c:5
#1 0x0000000000400622 in fib (num=4) at fib.c:9
#2 0x0000000000400631 in fib (num=6) at fib.c:9
#3 0x0000000000400680 in main () at fib.c:17
(gdb) s
9   return fib(num - 1) + fib(num - 2);
(gdb) bt
#0 fib (num=3) at fib.c:9
#1 0x0000000000400622 in fib (num=4) at fib.c:9
#2 0x0000000000400631 in fib (num=6) at fib.c:9
#3 0x0000000000400680 in main () at fib.c:17
(gdb) s
fib (num=2) at fib.c:5
5   if(num == 1 || num == 2)
(gdb) s
6   return 1;
(gdb) bt
#0 fib (num=2) at fib.c:6
#1 0x0000000000400622 in fib (num=3) at fib.c:9
#2 0x0000000000400622 in fib (num=4) at fib.c:9
#3 0x0000000000400631 in fib (num=6) at fib.c:9
#4 0x0000000000400680 in main () at fib.c:17
(gdb) s
10   }
(gdb) s
fib (num=1) at fib.c:5
5   if(num == 1 || num == 2)
(gdb) bt
#0 fib (num=1) at fib.c:5
#1 0x0000000000400631 in fib (num=3) at fib.c:9
#2 0x0000000000400622 in fib (num=4) at fib.c:9
#3 0x0000000000400631 in fib (num=6) at fib.c:9
#4 0x0000000000400680 in main () at fib.c:17
(gdb) s
6   return 1;
(gdb) s
10   }
(gdb) s
10   }
(gdb) s
fib (num=2) at fib.c:5
5   if(num == 1 || num == 2)
(gdb) bt
#0 fib (num=2) at fib.c:5
#1 0x0000000000400631 in fib (num=4) at fib.c:9
#2 0x0000000000400631 in fib (num=6) at fib.c:9
#3 0x0000000000400680 in main () at fib.c:17
(gdb) s
6   return 1;
(gdb) s
10   }
(gdb) s
10   }
(gdb) s


funnyjungler@funnyjungler-NH: ~/lecture/2018_0226
18:32
(gdb) bt
#0 fib (num=3) at fib.c:5
#1 0x0000000000400622 in fib (num=4) at fib.c:9
#2 0x0000000000400631 in fib (num=6) at fib.c:9
#3 0x0000000000400680 in main () at fib.c:17
(gdb) s
9   return fib(num - 1) + fib(num - 2);
(gdb) bt
#0 fib (num=3) at fib.c:9
#1 0x0000000000400622 in fib (num=4) at fib.c:9
#2 0x0000000000400631 in fib (num=6) at fib.c:9
#3 0x0000000000400680 in main () at fib.c:17
(gdb) s
fib (num=2) at fib.c:5
5   if(num == 1 || num == 2)
(gdb) s
6   return 1;
(gdb) bt
#0 fib (num=2) at fib.c:6
#1 0x0000000000400622 in fib (num=3) at fib.c:9
#2 0x0000000000400622 in fib (num=4) at fib.c:9
#3 0x0000000000400631 in fib (num=6) at fib.c:9
#4 0x0000000000400680 in main () at fib.c:17
(gdb) s
10   }
(gdb) s
fib (num=1) at fib.c:5
5   if(num == 1 || num == 2)
(gdb) bt
#0 fib (num=1) at fib.c:5
#1 0x0000000000400631 in fib (num=3) at fib.c:9
#2 0x0000000000400622 in fib (num=4) at fib.c:9
#3 0x0000000000400631 in fib (num=6) at fib.c:9
#4 0x0000000000400680 in main () at fib.c:17
(gdb) s
6   return 1;
(gdb) s
10   }
(gdb) s
10   }
(gdb) s
fib (num=2) at fib.c:5
5   if(num == 1 || num == 2)
(gdb) bt
#0 fib (num=2) at fib.c:5
#1 0x0000000000400631 in fib (num=4) at fib.c:9
#2 0x0000000000400631 in fib (num=6) at fib.c:9
#3 0x0000000000400680 in main () at fib.c:17
(gdb) s
6   return 1;
(gdb) s
10   }
(gdb) s
10   }
(gdb) s
10   }
(gdb) s
main () at fib.c:18
18   printf("%d번째 항의 수는 = %d\n", final_val, result);
(gdb) finish
"finish" not meaningful in the outermost frame.
(gdb)