임베디드 애플리케이션 분석!!!!!!!!!!!!

1. 이것이 없으면 사실상 C 언어를 사용할 수 없다.
C 언어에서 함수를 호출하기 위해서도 이것은 반드시 필요하다.
이와 같은 이유로 운영체제의 부팅 코드에서도
이것을 설정하는 작업이 진행되는데 이것은 무엇일까 ?

stack

2. 배열에 아래와 같은 정보들이 들어있다.
2400, 2400, 2400, 2400, 2400, 2400, 2400,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5,
2400, 2400, 2400, 2400, 2400, 2400, 1, 2, 3, 4,
5, 1, 2, 3, 4, 5, 2400, 2400, 2400, 2400, 2400, 5000,
1, 2, 3, 4, 5, 5000, 5000, 500, 500, 500, 500, 500,
1, 2, 3, 4, 5, 500, 500, 500, 500, 500, 500, 500,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15, 16, 17, 18, 234, 345, 26023, 346, 345, 234,
457, 3, 1224, 34, 646, 732, 5, 4467, 45, 623, 4, 356, 45, 6, 123,
3245, 6567, 234, 567, 6789, 123, 2334, 345, 4576, 678, 789, 1000,
2400, 2400, 2400, 2400, 2400, 2400, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5,
2400, 2400, 2400, 2400, 978, 456, 234756, 5000, 5000, 5000, 2400, 500, 5000, 2400, 500,
500, 500, 500, 500, 500, 1, 2, 3, 4, 5, 500, 500, 500, 500, 500,
500, 500, 500, 500, 500, 1, 2, 3, 4, 5, 500, 500, 500, 5000, 2400, 5000,
5000, 5000, 5000, 5000, 5000, 5000, 5000, 1, 2, 3, 4, 5, 5000, 5000,
5000, 5000, 2400 5000, 500, 2400, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 5000, 5000, 5000, 5000, 5000,
1, 2, 3, 4, 5, 5000, 5000, 5000, 5000, 5000, 234, 4564, 3243, 876,
645, 534, 423, 312, 756, 235, 75678, 2400, 5000, 500, 2400, 5000, 500, 2400, 5000,
500, 2400, 5000, 500, 2400, 5000, 500, 2400, 5000, 500, 2400, 5000, 500, 2400, 5000, 500,
2400,
5000, 500, 2400, 5000, 500, 7, 8, 9, 6, 7, 8, 9, 6, 7, 8, 9, 6, 7, 8, 9, 6, 7, 8, 9, 6, 7, 8,
9, 6, 500, 2400, 5000, 500, 2400, 5000, 500, 2400, 5000, 500, 2400, 5000, 500, 2400, 5000,
500, 2400, 5000,

여기서 가장 빈도수가 높은 3 개의 숫자를 찾아 출력하시오!
함수에서 이 작업을 한 번에 찾을 수 있도록 하시오.
(찾는 작업을 여러번 분할하지 말란 뜻임)

3. 12 비트 ADC 를 가지고 있는 장치가 있다.
보드는 12 V 로 동작하고 있고 ADC 는 -5 ~ 5 V 로 동작한다.
ADC 에서 읽은 값이 2077 일 때
이 신호를 디지털 관점에서 재해석하도록 프로그래밍 한다.

4. 전세계 각지의 천재들이 모여서 개발하는 리눅스 운영체제 코드에는
엄청나게 많은 양의 goto 가 사용되고 있다.
goto 를 도대체 왜 사용해야만 할까 ?

goto 문법의 이점

예를 들어 **for** 문을 중복해서 사용하고 **if** 와 **break** 로 **for** 문에서 빠져 나올 때**,** 상당히 많은 불필요한 명령어들이 사용 된다**.**
이러한 단점을 보완하기 위해 **goto** 문법을 사용하면 불필요한 코드를 사용하지 않고 한번에 **for** 문을 빠져나올 수 있다**.**
if 와 **break** 를 사용하면 기본적으로 **mov,cmp,jmp** 를 해야한다**.**
하지만 **goto** 는 **jmp** 하나로 끝난다**.**
또한 **, for** 문과 **if,break** 를 여러 개 조합을 할수록 **mov,cmp,jmp** 가 늘어난다**.** 여기서 문제는 **jmp** 다 **call** 이나 **jmp** 를 **cpu instrction** 레벨에서 분기 명령어라고 하고 이들은 **cpu** 파이프라인에 치명적인 손실을 가져다 준다**.**
기본적으로 분기 명령어는 파이프라인을 부순다**.**
이 뜻은 위의 가장 단순한 **cpu** 가 실행까지 **3clock** 을 소요하는데 기존의 파이프라인이 깨지니 쓸데없이 또 다시 **3clock** 을 버려야 함을 의미한다**.**
만약 이러한 파이프라인의 단계가 수십 단계라면
여기서 낭비되는 **cpu clock** 이 수없이 많음**.**
즉**,** 성능면으로만 보아도 **goto** 가 월등히 좋다는 것을 알 수 있다**.**
goto 는 다음 코드를 넘어 원하는 곳으로 넘어 갈 수 있다**.**

5. 포인터 크기에 대해 알고 있는대로 기술하시오.

8 비트 시스템의 경우 **1byte**
16 비트는 **2byte**
32 비트는 **4byte**
64 비트는 **8byte**
why?
컴퓨터의 산술 연산이 **ALU** 에 의존적이기 때문이다**.**
ALU 의 연산은 범용 레지스터에 종속적이고
컴퓨터가 **64** 비트라는 의미는 이들이 **64** 비트로 구성되었음을 의미한다**.**
변수의 정의는 메모리에 정보를 저장하는 공간이였다**.**

포인터의 정의는 메모리에 주소를 저장하는 공간이다
그렇다면 **64** 비트로 표현할 수 있는 최대값 또한 저장할 수 있어야한다**.**
포인터의 크기가 작다면 이 주소를 표현할 방법이 없기 때문에
최대치인 **64** 비트**(8byte)**가 포인터의 크기가 된 것이다**.**

실제 확인

```
#include <stdio.h>

int main(void) {
 printf("sizeof(int *) = %lu\n", sizeof(int *));
 printf("sizeof(double *) = %lu\n", sizeof(double *));
 printf("sizeof(float *) = %lu\n", sizeof(float *));
 return 0;
}
```

모두 **8** 이 나온다**.**

6. TI Cortex-R5F Safety MCU is very good to Real-Time System.

위의 문장에서 Safety MCU 가 몇 번째 부터
시작하는지 찾아내도록 프로그래밍 해보자.
(이정도는 가볍게 해야 파싱 같은것도 쉽게 할 수 있다)

```
#include<stdio.h>

int func(char arr[])
{
        int j;
        for(j = 0 ; j  <58; j++){

                if(arr[j] == 83)

        return j+1;
        }



}

int main(void)

{
        int i;

        char arr[58] = "TI Cortex-R5F Safety MCU is very good to Real-time System";
```

```
	i = func(arr);

	printf("Safety MCU 는 %d 번째부터 시작\n", i);

	return 0 ;

}
```

Safety MCU 는 **15** 번째부터 시작

7 중 배열을 함수의 인자로 입력 받아 3 by 3 행렬의 곱셈을 구현하시오.

```
#include<stdio.h>

void func(int(*p)[3])
{
	int a,b;

	for(a= 0 ; a<3; a++){
		for(b=0 ; b<3 ; b++){
			printf("%d ",p[a][b]*2);
		}
		printf("\n");
	}



}




int main(void)
{
	int i,j;

	int arr[3][3] = {{1,2,3},{1,2,3},{1,2,3}};



	for(i=0;i<3;i++){
		for(j=0;j<3;j++){
			printf("%d ",arr[i][j]);
		}
		printf("\n");
	}
```

```
        printf("행렬 곱 이후\n");

        func(arr);


        return 0;
}
```

1 2 3
1 2 3
1 2 3
행렬 곱 이후
2 4 6
2 4 6
2 4 6

8. void (* signal(int signum, void (* handler)(int)))(int) 의 프로토타입을 기술하시오.

void (*)(int)signal(int signum, void (* handler)(int))

리턴형 **void(*)(int)**
이름 **signal**
인자 **(int signum, void (* handler)(int))**

9. 함수 포인터를 반환하고 함수 포인터를 인자로 취하는 함수의 주소를 반환하고
인자로 int 2 개를 취하는 함수를 작성하도록 한다.
(프로토타입이 각개 다를 수 있으므로 프로토타입을 주석으로 기술하도록 한다)

10. 파이프라인이 깨지는 경우 어떠한 이점이 있는지 기술하시오.

11. 4x^2 + 5x + 1 을 1 ~ 3 까지 정적분하는 프로그램을 구현하도록 한다.

**********88

12. 값이 1 ~ 4096 까지 무작위로 할당되어 배열에 저장되도록 프로그래밍 하시오.
(배열의 크기는 100 개정도로 잡는다)

13. 12 번 문제에서 각 배열은 물건을 담을 수 있는 공간에 해당한다.
앞서서 100 개의 공간에 물건들을 담았는데 공간의 낭비가 있을 수 있다.
이 공간의 낭비가 얼마나 발생했는지 파악하는 프로그램을 작성하시오.

14. 13 번 문제에서 확장하여 공간을 보다 효율적으로 관리하고 싶어서
4096, 8192, 16384 등의 4096 배수로 크기를 확장할 수 있는 시스템을 도입했다.
이제부터 공간의 크기는 4096 의 배수이고
최소 크기는 4096, 최대 크기는 131072 에 해당한다.
발생할 수 있는 난수는 1 ~ 131072 로 설정하고
이를 효율적으로 관리하는 프로그램을 작성하시오.
(사실 리눅스 커널의 Buddy 메모리 관리 알고리즘임)

15. 이진 트리를 재귀 호출을 사용하여 구현하도록 한다.
(일반적인 SW 회사들 면접 당골 문제 - 이게 되면 큐 따위야 문제 없음)

16. 이진 트리를 재귀 호출 없이 구현하도록 한다.
결과를 확인하는 print 함수(전위, 중위, 후위 순회) 또한 재귀 호출을 수행하면 안됨

17. AVL 트리를 재귀 호출을 사용하여 구현하도록 한다.

18. AVL 트리를 재귀 호출 없이 구현하도록 한다.

19. Red Black 트리와 AVL 트리를 비교해보도록 한다.

20. 난수를 활용하여 Queue 를 구현한다.
(중복되는 숫자를 허용하지 않도록 프로그래밍 하시오)
제일 좋은 방법은 배열을 16 개 놓고 rand() % 16 을 해서
숫자가 겹치지 않는지 확인하면 된다.

21. 함수 포인터를 활용하여 float 형 3 by 3 행렬의 덧셈과
int 형 3 by 3 행렬의 덧셈을 구현하도록 하시오.

1 ~ 27 번째까지의 수들로 홀수들의 합을 하고 짝수들의 합을 구한다.
홀수들의 합 - 짝수들의 합의 결과를 출력하시오.
(프로그래밍 하시오)

```
#include<stdio.h>

int func(int num)
{
	if(num ==1 || num ==2){

		return 1;
	}

	else
		return func(num-1) + func(num-2);
}

int add_odd(int arr[])
{

	int a,result1=0;

	for(a=0 ; a <27; a++){

		if(!(a%2)){
			result1 = result1 + arr[a];
		}
	}

	return result1;
}




int add_even(int arr[])
{

	int b,result2=0;


	for(b=0 ; b <27; b++){

		if(b%2){
			result2 = result2 + arr[b];
		}
	}

	return result2;
}


int main(void)
{
```

```
	int i;
	int res1,res2,arr[27] ;


	for(i=0 ; i <27 ; i++){

		arr[i] = func(i+1);
	}

	res1 = add_odd(arr);
	res2 = add_even(arr);



	printf("피보나치 27 항까지 홀수들의 합 : %d\n",res1);
	printf("피보나치 27 항까지 짝수들의 합 : %d\n",res2);
	printf("홀수들의 합 - 짝수들의 합 : %d\n",res1-res2);



	return 0;

}
```

피보나치 **27** 항까지 홀수들의 합 **: 317811**
피보나치 **27** 항까지 짝수들의 합 **: 196417**
홀수들의 합 - 짝수들의 합 **: 121394**

141, 4, 5, 9, 14, 23, 37, 60, 97, ... 형태로 숫자가 진행된다.
23 번째 숫자는 무엇일까 ?
(프로그래밍 하시오)

```
#include<stdio.h>

int func(int num)
{
	if(num == 1)
		return 1;
	if(num == 2)
		return 4;
	else
		return func(num-1) + func(num-2);
}

int main(void)
{

	int res,i;
```

```
        printf("n 항을 입력하시오 : ");
        scanf("%d",&i);
        res = func(i);

        printf("n 번째 숫자는 : %d\n" , res);

        return 0;
}
```

n 항을 입력하시오 **: 23**
n 번째 숫자는 **: 81790**

15.el Architecture 와 ARM Architecture 의 차이점은 ?

x86 의 경우엔 함수의 입력을 스택으로 전달함
ARM 의 경우엔 함수 입력을 **4** 개까진 레지스터로 처리
하고 **4** 개가 넘어갈 경우엔 스택에 집어넣음
(그러므로 성능을 높이고자 한다면
ARM 에서는 함수 입력을 **4** 개 이하로 만드는 것이 좋음**)**

* 추가 정보

우선 모든 프로세서는
레지스터 에서 레지스터로 연산이 가능하다.
x86 은 메모리 에서 메모리로 연산이 가능하다.
하지만 ARM 은 로드/스토어 아키텍처라고 해서
메모리 에서 메모리로 연산이 불가능하다.
(반도체 다이 사이즈가 작기 때문임)
다이 사이즈가 작아서 Functional Unit 갯수가 적음
(이런 연산을 지원해줄 장치가 적어서 안됨)

그래서 ARM 은 먼저 메모리에서 레지스터로 값을 옮기고
다시 이 레지스터 값을 메모리로 옮기는 작업을함

16 우리반 학생들은 모두 25 명이다.
반 학생들과 함께 진행할 수 있는 사다리 게임을 만들도록 한다.
참여 인원수를 지정할 수 있어야하며
사다리 게임에서 걸리는 사람의 숫자도 조정할 수 있도록 만든다.

17. 아래와 같은 행렬을 생각해보자!

2 4 6
2 4 6

sapply(arr, func) 으로 위의 행렬을 아래와 같이 바꿔보자!

2 4 6
4 16 36

sapply 함수를 위와 같이 구현하라는 의미다.
(R 이나 python 같은 언어에서 지원되는 기법중 하나에 해당한다)

```
#include<stdio.h>

void func(int(*arr)[3])
{
	arr[1][0] +=2;
	arr[1][1] +=12;
	arr[1][2] +=30;
}

void sapply(int (*arr)[3],void(*func)(int(*arr)[3]))
{
	func(arr);
}

int main(void)
{

	int i,j;
	int arr[2][3] = {{2,4,6},{2,4,6}};

	sapply(arr,func);

	for(i=0;i<2;i++){

		for(j=0;j<3;j++){

			printf("%d ",arr[i][j]);
		}
		printf("\n");
```

```
        }
        return 0;
}
```

2 4 6
4 16 36

18. char *str = "WTF, Where is my Pointer ? Where is it ?" 라는 문자열이 있다
여기에 소문자가 총 몇 개 사용되었는지 세는 프로그램을 만들어보자

```
#include<stdio.h>
int func(char arr[])
{
        int i,num=0 ;

        for(i=0;i<41;i++){

                if(arr[i]<123 && arr[i]>96){

                        num++;
                }
        }
        return num;
}

int main(void)
{
        int res;

        char *str = "WTF, Where is my Pointer ? Where is it?";

        res = func(str);

        printf("소문자 개수 : %d\n",res);

        return 0 ;
}
```

소문자 개수 **: 22**

19. int *p[3] 와 int (*p)[3] 는 같은 것일까 ? 다른 것일까 ?
이유를 함께 기술하도록 한다.
다르다

int *p[3] : 포인터배열 인트형 포인터 **3** 개 저장할 수 있는 배열 **p**
int (*p)[3] :배열포인터 인트형 **3** 개를 담을 수 있는 배열 **p**

20. 임의의 값 x 가 있는데, 이를 134217728 단위로 정렬하고 싶다면 어떻게 해야할까 ?
어떤 숫자를 넣던 134217728 의 배수로 정렬이 된다는 뜻임
(힌트 : 134217728 = 2^27)

```
#include<stdio.h>

int main(void)
{
        int x,y;



        printf("수를 입력하시오 :");
        scanf("%d",&x);

        y=x>>27;

        printf("%d\n", y<<=27);

        return 0;
}
```

수를 입력하시오 **:150000000**
134217728

21. 단 한 번의 연산으로 대소문자 전환을 할 수 있는 연산에 대해 기술하시오.
(프로그래밍 하시오), 덧셈 혹은 뺄셈 같은 기능이 아님

```
#include<stdio.h>

char func(char num)
{

        return num^32;
}
```

```
int main(void)
{
	char i;


	printf("알파벳 대문자나 소문자를 적으시오 : ");
	scanf("%c",&i);
	printf("%c\n",func(i));

	return 0;
}
```

알파벳 대문자나 소문자를 적으시오 : T
t

22. 변수의 정의를 기술하시오.

변수의 정의는 메모리에 정보를 저장하는 공간이다

23. 포인터의 정의를 기술하시오.

포인터의 정의는 메모리에 주소를 저장하는 공간이다

24. 함수 포인터의 정의를 기술하시오.

함수를 메모리에 저장하는 공간

****

34. 재귀호출을 사용하여 queue 를 구현하고 10, 20 을 집어넣는다.
enqueue 과정에 대한 기계어 분석을 수행하여 동작을 파악하도록 한다.
그림과 함께 자세하게 설명하시오.

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

#define EMPTY 0


struct node
{
	int data;
	struct node *link;
```

```
};
typedef struct node Queue;


Queue *get_nude()
{
	Queue *tmp;
	tmp = (Queue *)malloc(sizeof(Queue));
	tmp->link =EMPTY;
	return tmp;

}



void enqueue(Queue **head , int data)
{


	if(*head ==EMPTY)
	{
		*head = get_nude();
		(*head)->data = data;
		return ;
	}
	enqueue(&(*head)->link,data);


}



void print_queue(Queue *head)
{
	Queue *tmp = head;

	while(tmp)
	{
		printf("%d\n", tmp->data);
		tmp = tmp->link;
	}

}


int main(void)
{
	Queue *head  =EMPTY ;
	enqueue(&head,10);
	enqueue(&head,20);
	print_queue(head);
```

return 0;

}
10
20

25. 파이프라인은 언제 깨지는가 ?

call 이나 **jmp** 를 **cpu instrction** 레벨에서 분기 명령어라고 하고 이들은 **cpu** 파이프라인에 치명적인 손실을 가져다 준다**.**
기본적으로 분기 명령어는 파이프라인을 부순다**.**

26. 메모리 계층 구조에 대해 기술하시오.

메모리 계층 구조란 메모리를 필요에 따라 여러가지 종류로 나누어 둠을 의미한다**.**

속도 순으로
1 레지스터
2 캐시
3 메모리
4 디스크
용량은 역순

27. C 언어의 기본 메모리 구조에 대해 기술하시오.

메모리 구조의 기초
메모리공간 이름/ 들어오는 데이터 / 관리방식

Stack 지역 변수가 위치하는 영역 동적

Heap 동적 할당된 메모리들이 위치하는 영역 동적
Data 전역 변수 및 static 으로정적
선언된 것들이 위치하는 영역
초기화 되지 않은 모든것은 0 으로 저장됨

코드 Machine Code 가 위치하는 영역 정적

변수는 공통적으로 자료가 저장되는 공간

동적:프로그램 실행 중 바뀌는 공간(메모리 할당 및 해제)

정적**:**프로그램 실행 중 바뀌지 않는 공간

28. 우리가 사용하는 윈도우나 리눅스, 맥(유닉스)에서 보는
변수 주소는 진짜 주소일까 아닐까 ?
알고 있는대로 기술하시오.

가상주소다**.**

29. 이름과 급여를 저장하도록 만든다.
이름은 마음대로 고정시키도록 하고 급여는 rand() 를 활용
급여의 평균을 출력하고 가장 높은 한 사람의 이름과 급여를 출력하시오.
(값이 같을 수 있음에 유의해야 한다)

30. 리눅스에서 디버깅을 수행하기 위한 프로그램 컴파일 방법을 기술하시오.

소스 파일 만든후
$ gcc test.c 컴파일

****

41. 난수를 활용해서 Stack 을 구성한다.
같은 숫자가 들어가지 않게 하고 20 개를 집어넣는다.
이때 들어가는 숫자는 1 ~ 100 사이의 숫자로 넣는다.
(마찬가지로 중복되지 않게 한다)

42. 41 번에서 홀수만 빼내서 AVL 트리를 구성하도록 한다.

43. 41 번에서 짝수만 빼내서 RB 트리를 구성하도록 한다.

-------임베디드 애플리케이션 분석

31. vi 에서 코드가 정렬이 잘 안되어 있다.
이 상황에서 어떻게 해야 예쁘고 깔끔하게 정렬되는가 ?

esc 를 눌러 비입력 상태로 만들고 시작부분을 커서로 클릭한 후 **v** 를 눌러 복사할 방향으로
복사할 만큼 지정한 후 = 키를 누르면 된다**.**

32. 프로그램을 최적화하는 컴파일 옵션을 적고
반대로 어떠한 최적화도 수행하지 않는 컴파일 옵션을 적도록 한다.

요즘 gcc(컴파일러)가 너무 똑똑해져서

최적화 옵션을 주지 않아도 알아서 최적화해버리는데 -→>gcc g -o debug func.c <-(최적화 된 것)

강제로 최적화를 못하게 할 수 있다.

-O0 옵션을 추가하면 된다: 영문자 O 와 숫자 0 이다.

즉 위의 컴파일 옵션을 아래와 같이 바꾼다.

gcc -g -O0 -o debug func.c

모든 최적화를 방지하고 디버깅 옵션을 집어넣었음을 의미한다**.**

****

46. 최적화 프로세스를 기술하도록 한다.

47. 기존에는 자료구조에서 숫자값만을 받았다.
이제 Queue 에서 데이터로서 숫자가 아닌 문자열을 받아보자.

48. Binary Tree 에서 위 작업을 수행해보라.

49. AVL Tree 에서 위 작업을 수행해보라.

50. 성적 관리 프로그램을 만들어보자.

1. **통계 기능(총 합산, 평균, 표준 편차 계산)**
2. **성적순 정렬 기능**
3. **성적 입력 기능**
4. **학생 정보 삭제 기능**

----------임베디드 애플리케이션 분석

33. gdb 를 사용하는 이유를 기술하라.

컴퓨터 프로그램 오류**(**보통 논리적 오류**)** 를 찾아내어 바로잡기 위해

34. 기계어 레벨에서스택을 조정하는 명령어는 어떤것들이 있는가 ?

gdb debug

이와 같이 입력하면 디버거가 켜지면서 (gdb) 창이 보임

b main

위와 같이 입력하면 main 함수에서 멈춰달라는 뜻이다.

r 을 눌러서 프로그램을 구동

disas

이 명령어를 입력하면 현재 프로그램에 대한
디스어셈블리된 어셈블리어 코드가 보인다.

b *복사한주소

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

35. a 좌표(3, 6), b 좌표(4, 4) 가 주어졌다.
원점으로부터 이 두 좌표가 만들어내는 삼각형의 크기를 구하는 프로그램을 작성하라.

36. 가위 바위 보 게임을 만들어보자.
프로그램을 만들고 컴퓨터랑 배틀 한다.

37. 화면 크기가 가로 800, 세로 600 이다.
여기서 표현되는 값은 x 값이 [-12 ~ + 12] 에 해당하며 y 값은 [-8 ~ +8] 에 해당한다.
x 와 y 가 출력하게 될 값들을 800, 600 화면에 가득차게 표현할 수 있는
스케일링 값을 산출하는 프로그램을 작성하도록 한다.

38. 등차 수열의 합을 구하는 프로그램을 for 문을 도는 방식으로 구현하고
등차 수열의 합 공식을 활용하여 구현해본다.
함수 포인터로 각각의 실행 결과를 출력하고
이 둘의 결과가 같은지 여부를 파악하는 프로그램을 작성하라.

39. sin(x) 값을 프로그램으로 구현해보도록 한다.
어떤 radian 값을 넣던지 그에 적절한 결과를 산출할 수 있도록 프로그래밍 한다.

40

한달간 적은 시간이기도 하고 많은 시간이
기도 했다.

	한 달간 수업을 진행하면서 본인이 느낀점을 20 줄 이상 최대한 상세하게 기술하시오.
내용	또한 앞으로에 대한 포부 또한 기술하길 바란다. 그리고 앞으로 어떤 일을 하고 싶은지 상세히 기술하도록 한다.

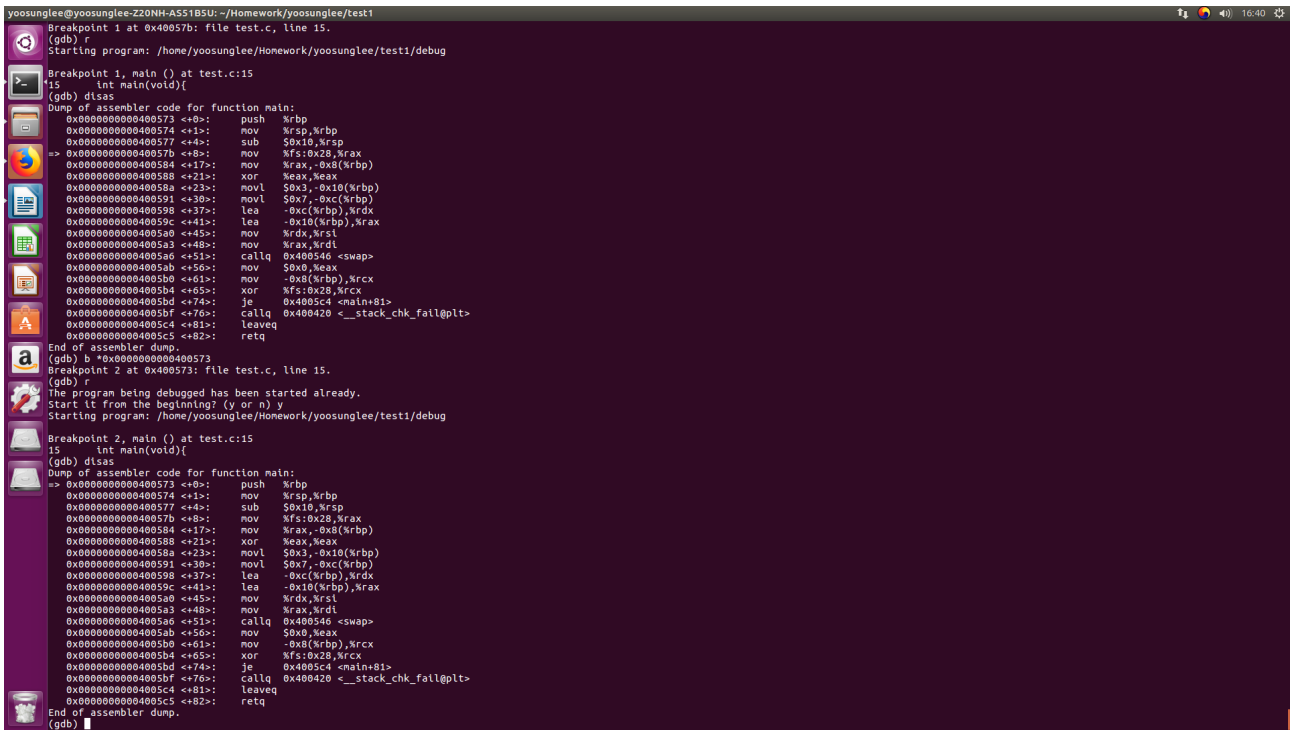
한 달간 수업받으면서 많이 빡셌고 잘하고 싶은 생각이 들었음**.**
앞으로도 열심히 배울 생각**.**
어떤 일을 하고 싶은지 구체적으로는 더 하면서 다른 생각도 해볼 생각이다

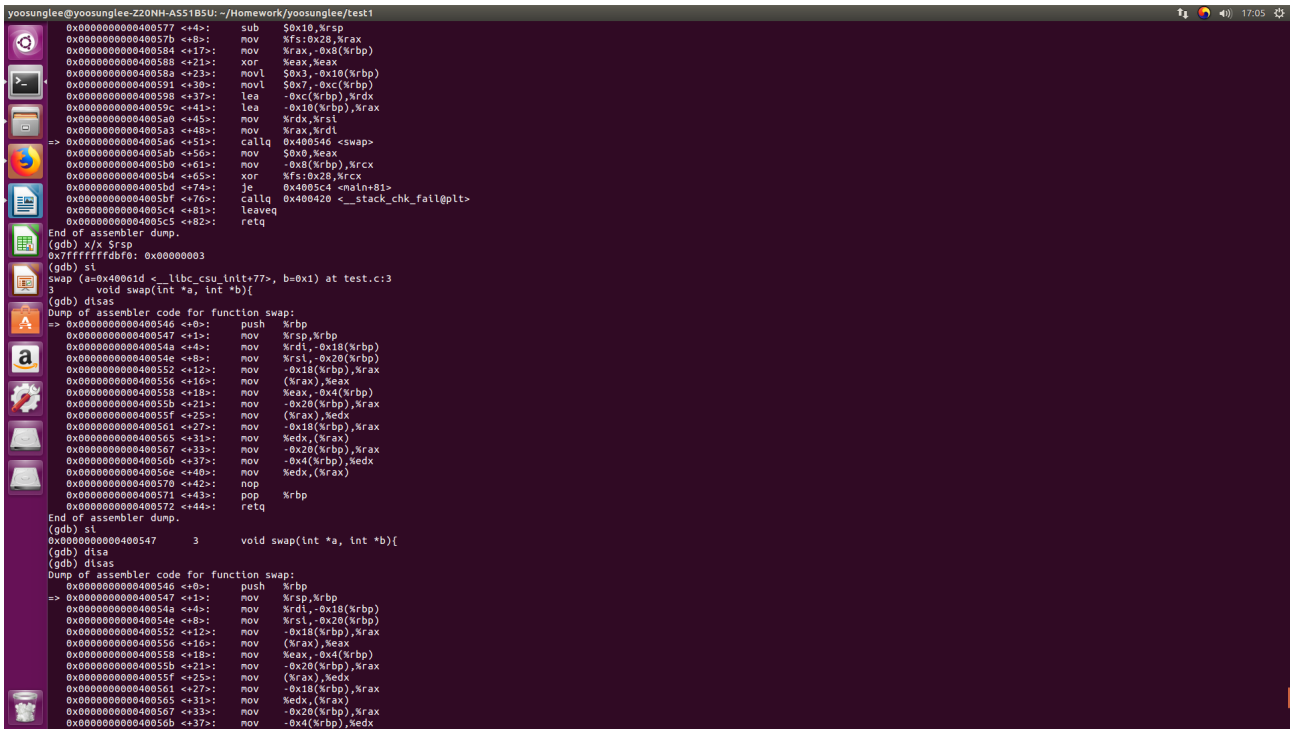
아래 **Code** 를 작성하고 이 **Code** 의 기계어에 대한 그림을 그리고 분석하시오**.**

```
#include<stdio.h>

void swap(int *a, int *b){
```

```
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
}
int main(void){
    int num1 = 3, num2 = 7;
    swap(&num1, &num2);
    return 0;
}
```


yoosunglee@yoosunglee-Z20NH-AS51B5U: ~/Homework/yoosunglee/test1
16:40
Breakpoint 1 at 0x40057b: file test.c, line 15.
(gdb) r
Starting program: /home/yoosunglee/Homework/yoosunglee/test1/debug
Breakpoint 1, main () at test.c:15
15 int main(void){
(gdb) disas
Dump of assembler code for function main:
0x0000000000400573 <+0>: push %rbp
0x0000000000400574 <+1>: mov %rsp,%rbp
0x0000000000400577 <+4>: sub $0x10,%rsp
=> 0x000000000040057b <+8>: mov %fs:0x28,%rax
0x0000000000400584 <+17>: mov %rax,-0x8(%rbp)
0x0000000000400588 <+21>: xor %eax,%eax
0x000000000040058a <+23>: movl $0x3,-0x10(%rbp)
0x0000000000400591 <+30>: movl $0x7,-0xc(%rbp)
0x0000000000400598 <+37>: lea -0xc(%rbp),%rdx
0x000000000040059c <+41>: lea -0x10(%rbp),%rax
0x00000000004005a0 <+45>: mov %rdx,%rsi
0x00000000004005a3 <+48>: mov %rax,%rdi
0x00000000004005a6 <+51>: callq 0x400546 <swap>
0x00000000004005ab <+56>: mov $0x0,%eax
0x00000000004005b0 <+61>: mov -0x8(%rbp),%rcx
0x00000000004005b4 <+65>: xor %fs:0x28,%rcx
0x00000000004005bd <+74>: je 0x4005c4 <main+81>
0x00000000004005bf <+76>: callq 0x400420 <__stack_chk_fail@plt>
0x00000000004005c4 <+81>: leaveq
0x00000000004005c5 <+82>: retq
End of assembler dump.
(gdb) b *0x0000000000400573
Breakpoint 2 at 0x400573: file test.c, line 15.
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/yoosunglee/Homework/yoosunglee/test1/debug
Breakpoint 2, main () at test.c:15
15 int main(void){
(gdb) disas
Dump of assembler code for function main:
=> 0x0000000000400573 <+0>: push %rbp
0x0000000000400574 <+1>: mov %rsp,%rbp
0x0000000000400577 <+4>: sub $0x10,%rsp
0x000000000040057b <+8>: mov %fs:0x28,%rax
0x0000000000400584 <+17>: mov %rax,-0x8(%rbp)
0x0000000000400588 <+21>: xor %eax,%eax
0x000000000040058a <+23>: movl $0x3,-0x10(%rbp)
0x0000000000400591 <+30>: movl $0x7,-0xc(%rbp)
0x0000000000400598 <+37>: lea -0xc(%rbp),%rdx
0x000000000040059c <+41>: lea -0x10(%rbp),%rax
0x00000000004005a0 <+45>: mov %rdx,%rsi
0x00000000004005a3 <+48>: mov %rax,%rdi
0x00000000004005a6 <+51>: callq 0x400546 <swap>
0x00000000004005ab <+56>: mov $0x0,%eax
0x00000000004005b0 <+61>: mov -0x8(%rbp),%rcx
0x00000000004005b4 <+65>: xor %fs:0x28,%rcx
0x00000000004005bd <+74>: je 0x4005c4 <main+81>
0x00000000004005bf <+76>: callq 0x400420 <__stack_chk_fail@plt>
0x00000000004005c4 <+81>: leaveq
0x00000000004005c5 <+82>: retq
End of assembler dump.
(gdb)


yoosunglee@yoosunglee-Z20NH-AS51B5U: ~/Homework/yoosunglee/test1
17:05
0x0000000000400577 <+4>: sub $0x10,%rsp
0x000000000040057b <+8>: mov %fs:0x28,%rax
0x0000000000400584 <+17>: mov %rax,-0x8(%rbp)
0x0000000000400588 <+21>: xor %eax,%eax
0x000000000040058a <+23>: movl $0x3,-0x10(%rbp)
0x0000000000400591 <+30>: movl $0x7,-0xc(%rbp)
0x0000000000400598 <+37>: lea -0xc(%rbp),%rdx
0x000000000040059c <+41>: lea -0x10(%rbp),%rax
0x00000000004005a0 <+45>: mov %rdx,%rsi
0x00000000004005a3 <+48>: mov %rax,%rdi
=> 0x00000000004005a6 <+51>: callq 0x400546 <swap>
0x00000000004005ab <+56>: mov $0x0,%eax
0x00000000004005b0 <+61>: mov -0x8(%rbp),%rcx
0x00000000004005b4 <+65>: xor %fs:0x28,%rcx
0x00000000004005bd <+74>: je 0x4005c4 <main+81>
0x00000000004005bf <+76>: callq 0x400420 <__stack_chk_fail@plt>
0x00000000004005c4 <+81>: leaveq
0x00000000004005c5 <+82>: retq
End of assembler dump.
(gdb) x/x $rsp
0x7fffffffdbf0: 0x00000003
(gdb) si
swap (a=0x40061d <__libc_csu_init+77>, b=0x1) at test.c:3
3 void swap(int *a, int *b){
(gdb) disas
Dump of assembler code for function swap:
=> 0x0000000000400546 <+0>: push %rbp
0x0000000000400547 <+1>: mov %rsp,%rbp
0x000000000040054a <+4>: mov %rdi,-0x18(%rbp)
0x000000000040054e <+8>: mov %rsi,-0x20(%rbp)
0x0000000000400552 <+12>: mov -0x18(%rbp),%rax
0x0000000000400556 <+16>: mov (%rax),%eax
0x0000000000400558 <+18>: mov %eax,-0x4(%rbp)
0x000000000040055b <+21>: mov -0x20(%rbp),%rax
0x000000000040055f <+25>: mov (%rax),%edx
0x0000000000400561 <+27>: mov -0x18(%rbp),%rax
0x0000000000400565 <+31>: mov %edx,(%rax)
0x0000000000400567 <+33>: mov -0x20(%rbp),%rax
0x000000000040056b <+37>: mov -0x4(%rbp),%edx
0x000000000040056e <+40>: mov %edx,(%rax)
0x0000000000400570 <+42>: nop
0x0000000000400571 <+43>: pop %rbp
0x0000000000400572 <+44>: retq
End of assembler dump.
(gdb) si
0x0000000000400547 3 void swap(int *a, int *b){
(gdb) disa
(gdb) disas
Dump of assembler code for function swap:
0x0000000000400546 <+0>: push %rbp
=> 0x0000000000400547 <+1>: mov %rsp,%rbp
0x000000000040054a <+4>: mov %rdi,-0x18(%rbp)
0x000000000040054e <+8>: mov %rsi,-0x20(%rbp)
0x0000000000400552 <+12>: mov -0x18(%rbp),%rax
0x0000000000400556 <+16>: mov (%rax),%eax
0x0000000000400558 <+18>: mov %eax,-0x4(%rbp)
0x000000000040055b <+21>: mov -0x20(%rbp),%rax
0x000000000040055f <+25>: mov (%rax),%edx
0x0000000000400561 <+27>: mov -0x18(%rbp),%rax
0x0000000000400565 <+31>: mov %edx,(%rax)
0x0000000000400567 <+33>: mov -0x20(%rbp),%rax
0x000000000040056b <+37>: mov -0x4(%rbp),%edx