[임베디드 애플리케이션 분석 - 1 번]

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 ?

   답 : 스택

[임베디드 애플리케이션 분석 - 2 번]

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)

[임베디드 애플리케이션 분석 - 3 번]

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.

[임베디드 애플리케이션 분석 - 4 번]

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

중첩된 반복문을 빠져나올 때 break 를 사용하는 것은 한 반복문만을 빠져나오기 때문에 코드가 복잡해지기도 하고 파이프라인이 더 많이 깨지게 된다. 하지만 goto 를 사용하게 되면 코드를 간략하게 사용하여 빠져나올 수 있고 break 를 이용해 중첩된 반복문을 빠져나오는 방법보다 파이프라인이 적게 깨져 시간적인 요소 등 에서 성능이 효율적이어서 goto 를 사용한다.

[임베디드 애플리케이션 분석 - 5 번]

5. 포인터 크기에 대해 알고 있는대로 기술하시오.

포인터의 크기는 컴퓨터의 HW(메모리)의 크기에 따라 달라진다. 포인터는 주소를 나타낼 수 있어야 하므로 잘리면 안되기 때문에 64bit 시스템에서 포인터의 크기는 8byte, 32bit 시스템에서는 4byte 가 된다.

[임베디드 애플리케이션 분석 - 6 번]

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)

답)

```
#include<stdio.h>

int find(char *str)
{

	int c=0;
	while(str[c])
	{
		if((str[c]==77)&&(str[c+1]==67)&&(str[c+2]==85))
		{
			return c+1;
		}

	c++;
	}
}

int main(void)
{
	char *str="TI Cortex-R5F Safety MCU is very good to Real-Time System.";
	printf("%d 번째",find(str));
```

```
	return 0;
}
```

결과)
22 번째

[임베디드 애플리케이션 분석 - 7 번]
7. 이중 배열을 함수의 인자로 입력 받아 3 by 3 행렬의 곱셈을 구현하시오.

답)

```
#include<stdio.h>

void mul(int arr[3][3],int brr[3][3],int crr[3][3])
{

	int i,j;

	for(i=0;i<3;i++)
	{
		for(j=0;j<3;j++)
		{
			crr[i][j]=arr[i][0]*brr[0][j] + arr[i][1]*brr[1][j] + arr[i][2]*brr[2][j];
		}
	}
}

void print(int crr[3][3])
{
	int i,j;
	for(i=0;i<3;i++)
	{
		for(j=0;j<3;j++)
		{
			printf("%d\t",crr[i][j]);
		}
		printf("\n");
	}
}
int main(void)
{
	int arr[3][3]={{1,2,3},{0,1,4},{0,2,5}};
	int brr[3][3]={{1,2,3},{0,1,4},{0,2,5}};
	int crr[3][3]={0};
	int i;
	mul(arr,brr,crr);
	print(crr);
	return 0;
}
```

결과)

```
1	10	26
```

0 9 24
0 12 33

[임베디드 애플리케이션 분석 - 8 번]
8. void (* signal(int signum, void (* handler)(int)))(int) 의 프로토타입을 기술하시오.

리턴 : void (*)(int)
함수이름 : signal
인자 : int signum, void (* handler)(int)

[임베디드 애플리케이션 분석 - 9 번]
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)

(void) (*p1)(void) (*p3)((void) (*p2) (void)) func (int,int)
(void)(*p1(*p3(*p2 func (int,int)) (void)))((void))(void)
(void) (*p1 (*p3)((void) (*p2) (void)))(void)

프로토타입
리턴 : (void) (*p1)(void) (*p3)((void) (*p2) (void))
함수이름 : func
인자 : (int,int)

[임베디드 애플리케이션 분석 - 10 번]
10. 파이프라인이 깨지는 경우 어떠한 이점이 있는지 기술하시오.

없다

[임베디드 애플리케이션 분석 - 11 번]
11. 4x^2 + 5x + 1 을 1 ~ 3 까지 정적분하는 프로그램을 구현하도록 한다.

[임베디드 애플리케이션 분석 - 12 번]
21. 함수 포인터를 활용하여 float 형 3 by 3 행렬의 덧셈과
int 형 3 by 3 행렬의 덧셈을 구현하도록 하시오.

[임베디드 애플리케이션 분석 - 13 번]
22. 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... 형태로 숫자가 진행된다.
1 ~ 27 번째까지의 수들로 홀수들의 합을 하고 짝수들의 합을 구한다.
홀수들의 합 - 짝수들의 합의 결과를 출력하시오.
(프로그래밍 하시오)

답)
#include<stdio.h>

```
int fib(int *bef,int *aft)
{
int i=0;
int tmp;

for(i=1;i<=27;i++)
{
	tmp=*aft;
	*aft+=*bef;
	*bef=tmp;
//	printf("%d, %d\n",i,*bef);
	if(i>=27)
		return *bef;
}

}

int main(void)
{
int bef=0;
int aft=1;

printf("\n%d",fib(&bef,&aft));

return 0;
}
```

결과)

196418

[임베디드 애플리케이션 분석 - 14 번]

23. 1, 4, 5, 9, 14, 23, 37, 60, 97, ... 형태로 숫자가 진행된다.
23 번째 숫자는 무엇일까 ?
(프로그래밍 하시오)

답)

```
#include<stdio.h>

int fib(int *bef,int *aft)
{
	int i=0;
	int tmp;

	for(i=1;i<=23;i++)
	{
		tmp=*aft;
		*aft+=*bef;
		if(i>=23)
			return *bef;
		*bef=tmp;
	}
```

```
}

int main(void)
{
	int bef=1;
	int aft=4;

	printf("23 번째 : %d",fib(&bef,&aft));

	return 0;
}
```

결과)
23 번째 : 81790

[임베디드 애플리케이션 분석 - 15 번]
24. Intel Architecture 와 ARM Architecture 의 차이점은 ?

[임베디드 애플리케이션 분석 - 16 번]
25. 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.

[임베디드 애플리케이션 분석 - 17 번]
26. 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)

답)

```
#include<stdio.h>
int func(int n)
{
	return n*n;
}

void sapply(int arr[2][3],int (*p)(int n))
{
	int i,j;
	for(i=0;i<3;i++)
```

```
        {
            arr[1][i]=func(arr[0][i]);
            printf("%d\t",arr[0][i]);
        }
        printf("\n");
        for(i=0;i<3;i++)
            printf("%d\t",arr[1][i]);
}
int main(void)
{
        int arr[2][3]={{2,4,6},{2,4,6}};
        sapply(arr,func);
        return 0;
}
```

결과)

```
2       4       6
4       16      36
```

[임베디드 애플리케이션 분석 - 18 번]

27. 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 small_count(char *str)
{
        int c=0,count=0,size=0;

        while(str[c])
        {
            if(str[c]<122&&str[c]>97)
                count++;
            c++;
        }
        return count;

}

int main(void)
{
        char *str="WTF, Where is my Pointer ? Where is it ?";

        printf("%d",small_count(str));
        return 0;
}
```

결과)

```
22
```

[임베디드 애플리케이션 분석 - 19 번]

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

    다른 것이다. int *p[3]는 주소를 저장할 수 있는 3 개짜리 배열이고, int (*p)[3]는 int 형 3 개짜리 배열을 가리키는 포인터이기 때문에 다른 것이다.

[임베디드 애플리케이션 분석 - 20 번]

29. 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)

    답)

```
#include<stdio.h>

int square(int size)
{
	int num;
	for(num=1;num<=size;num++)
	{
		num*=2;
	}
	return num;
}

void array(int val,int x)
{
	int y=0;
	if(x<val)
	{
		printf("%d",x);
	}
	else if(x>=val)
	{
		y= x & ~(val-1);
		printf("y = %d",y);
	}

}

int main(void)
{
	int size=27;
	int val,x;
	val=square(size);
	printf("정렬할 값 : ");
	scanf("%d",&x);

	array(val,x);
	return 0;
```

```
}
```

결과)
정렬할 값 : 134217730
y = 134217728

[임베디드 애플리케이션 분석 - 21 번]
30. 단 한 번의 연산으로 대소문자 전환을 할 수 있는 연산에 대해 기술하시오.
(프로그래밍 하시오), 덧셈 혹은 뺄셈 같은 기능이 아님

답)

```
#include<stdio.h>
void change(char arr[10])
{
	int i,re;
	printf("전환 후 : ");
	for(i=0;i<10;i++)
	{
		re=arr[i]^32;
		printf("%c",re);
	}
}
int main(void)
{
	char arr[10];
	scanf("%s",arr);
	printf("전환 전 : %s\n",arr);
	change(arr);
	return 0;
}
```

결과)
Aa
전환 전 : Aa
전환 후 : aA

[임베디드 애플리케이션 분석 - 22 번]
31. 변수의 정의를 기술하시오.

변수 : 데이터를 저장하는 메모리 공간.

[임베디드 애플리케이션 분석 - 23 번]
32. 포인터의 정의를 기술하시오.

포인터 : 주소를 저장할 수 있는 메모리 공간.

[임베디드 애플리케이션 분석 - 24 번]
33. 함수 포인터의 정의를 기술하시오.

함수 포인터 : 함수를 가리키는 포인터

[임베디드 애플리케이션 분석 - 25 번]
35. 파이프라인은 언제 깨지는가 ?

goto 명령문을 사용하거나 break 문을 사용 했을 때 기계어로 보게 되면 call 명령이 실행되게 되는데 이 명령이 실행 될 때 파이프라인이 깨지게 된다.

[임베디드 애플리케이션 분석 - 26 번]
36. 메모리 계층 구조에 대해 기술하시오.

레지스터, 캐시, 메모리, 디스크 로 구성되어 있으며 레지스터-캐시-메모리- 디스크 순으로 속도가 빠그다.
하지만 용량면에서 본다면 디스크-메모리-캐시-레지스터 순으로 용량이 크다.

[임베디드 애플리케이션 분석 - 27 번]
37. C 언어의 기본 메모리 구조에 대해 기술하시오.

메모리구조는 text, data, heap, stack 으로 구성되어 있다. text 는 기계어 코드가 위치하는 영역으로 쓰기가 금지되어있고 크기가 고정되어있다. heap 은 동적할당되는 메모리가 저장되는 공간이다. data 는 전역변수 및 정적변수(static)으로 선언된 변수들이 저장되는 메모리 공간이다. stack 은 지역변수가 위치하는 영역으로 함수 내에서만 존재할 수 있고 함수호출이 완료되면 사라지게 된다.

[임베디드 애플리케이션 분석 - 28 번]
38. 디버깅 하면서 보는 변수의 주소는 무엇일까?
우리가 사용하는 윈도우나 리눅스, 맥(유닉스)에서 보는
변수 주소는 진짜 주소일까 아닐까 ?
알고 있는대로 기술하시오.

우리가 디버깅 하면서 보는 변수의 주소는 가짜주소이다. 추측해서 생각해 보면 기계어 분석 할 때 어딘가에 있던 rbp 가 기계어의 처음에서 push 될 때 그때부터 새로운 주소가 할당되게 되는데 이 주소는 가짜주소여서 프로그램이 다 실행되고 완료하게되면 가짜주소가 사라지게 될 것이라 생각된다. 프로그램을 돌릴 때 마다 가짜주소를 그때그때 주소 값이 바뀔 것이다.

[임베디드 애플리케이션 분석 - 29 번]
39. 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)

[임베디드 애플리케이션 분석 - 30 번]
40. 리눅스에서 디버깅을 수행하기 위한 프로그램 컴파일 방법을 기술하시오.

gcc -g 디버깅파일이름 c 파일 이름.c
위의 명령어를 사용하면 컴파일을 하고 중간에 작성된 파일 이름으로 디버깅 파일을 생성한다.

[임베디드 애플리케이션 분석 - 31 번]
44. vi 에서 코드가 정렬이 잘 안되어 있다.
이 상황에서 어떻게 해야 예쁘고 깔끔하게 정렬되는가 ?

v 를 눌러 정렬할 코드를 선택한 뒤 =를 누르면 정렬된다.

[임베디드 애플리케이션 분석 - 32 번]
45. 프로그램을 최적화하는 컴파일 옵션을 적고
반대로 어떠한 최적화도 수행하지 않는 컴파일 옵션을 적도록 한다.

최적화하여 컴파일하는 옵션
gcc -g -o -o0 디버깅파일이름 파일이름.c

최적화하지않고 컴파일하는 옵션
gcc -g -o 디버깅파일이름 파일이름.c

[임베디드 애플리케이션 분석 - 33 번]
51. gdb 를 사용하는 이유를 기술하라.

디버깅은 컴파일은 성공적으로 문법적오류는 없지만 논리적 오류가 존재하는 경우 수행한다.
[임베디드 애플리케이션 분석 - 34 번]
52. 기계어 레벨에서 스택을 조정하는 명령어는 어떤것들이 있는가 ?

sub, mov, lea,callq

[임베디드 애플리케이션 분석 - 35 번]
53. a 좌표(3, 6), b 좌표(4, 4) 가 주어졌다.
원점으로부터 이 두 좌표가 만들어내는 삼각형의 크기를 구하는 프로그램을 작성하라.

[임베디드 애플리케이션 분석 - 36 번]
54. 가위 바위 보 게임을 만들어보자.
프로그램을 만들고 컴퓨터랑 배틀 한다.

[임베디드 애플리케이션 분석 - 37 번]
55. 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.

[임베디드 애플리케이션 분석 - 38 번]
등차 수열의 합을 구하는 프로그램을 for 문을 도는 방식으로 구현하고

등차 수열의 합 공식을 활용하여 구현해보자.

함수 포인터로 각각의 실행 결과를 출력하고 이 둘의 결과가 같은지 여부를 파악하는 프로그램을 작성하라.

[임베디드 애플리케이션 분석 - 39 번]
57. sin(x) 값을 프로그램으로 구현해보도록 한다.
어떤 radian 값을 넣던지 그에 적절한 결과를 산출할 수 있도록 프로그래밍 한다.

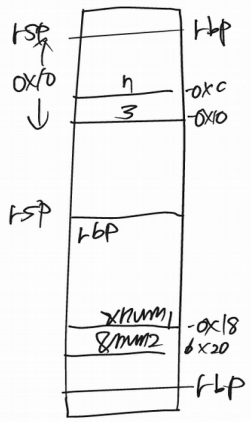
[임베디드 애플리케이션 분석 - 40 번]
한 달간 수업을 진행하면서 본인이 느낀점을 20 줄 이상 최대한 상세하게 기술하시오.

또한 앞으로에 대한 포부 또한 기술하길 바란다.

그리고 앞으로 어떤 일을 하고 싶은지 상세히 기술하도록 한다.

영상신호처리 한번 배워보고싶다 자세하게 서술하고싶은데요 직ㅁ 시간이없습ㄴ다

[임베디드 애플리케이션 분석 - 41 번]



eax 0 → &num1 → 0.
rdx → &num2
rax &num1 → &num2
rsi &num2
rdi &num1
edx &num1
rcx

[자료구조 - 1 번 (복합문제 1.1)]
12. 값이 1 ~ 4096 까지 무작위로 할당되어 배열에 저장되도록 프로그래밍 하시오.
(배열의 크기는 100 개정도로 잡는다)

답)

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

int input(int arr[100])
{
	int i;
	for(i=0;i<100;i++)
	{
		arr[i]=rand()%4096+1;

	}

	return *arr;
}

int main(void)
{
	int arr[100];
	int i;
	srand(time(NULL));

	input(arr);

	for(i=0;i<100;i++)
	{
		printf("%d\t",arr[i]);
	}
	return 0;
}
```

결과)

120	3088	2666	483	2763	2687	994	2995	3858	1304	3734
2749	3888	2864	545	420	807	1756	2113	3988	2470	2209
2564	2231	2987	1329	3291	916	3622	511	2813	3742	3598
1382	128	2265	4068	1121	1163	3829	2425	800	2481	2216
3664	3025	2635	374	684	651	266	3153	2860	2829	1288
1750	62	482	2665	3683	992	1381	3328	494	2762	3456
2758	2733	480	3920	2465	2904	623	849	1024	190	3873
3658	564	460	213	829	3613	3072	3657	804	725	3718
1285	3390	3305	2277	674	2536	2770	3436	1895	1431	2072
2375										

[자료구조 - 2 번 (복합문제 1.2)]
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.

답)

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

int input(int arr[100])
{
	int i;
	for(i=0;i<100;i++)
	{
		arr[i]=rand()%4096+1;
	}

	return *arr;
}

int main(void)
{
	int arr[100];
	int i;
	srand(time(NULL));

	input(arr);

	for(i=0;i<100;i++)
	{
		printf("들어간 값 : %d, 낭비된 공간 : %d\n",arr[i],4096-arr[i]);
	}
	return 0;
}
```

결과)

들어간 값 : 286, 낭비된 공간 : 3810	들어간 값 : 1227, 낭비된 공간 : 2869	들어간 값 : 1137, 낭비된 공간 : 2959
들어간 값 : 822, 낭비된 공간 : 3274	들어간 값 : 3790, 낭비된 공간 : 306	들어간 값 : 3196, 낭비된 공간 : 900
들어간 값 : 3073, 낭비된 공간 : 1023	들어간 값 : 1002, 낭비된 공간 : 3094	들어간 값 : 3696, 낭비된 공간 : 400
들어간 값 : 2932, 낭비된 공간 : 1164	들어간 값 : 204, 낭비된 공간 : 3892	들어간 값 : 4061, 낭비된 공간 : 35
들어간 값 : 2456, 낭비된 공간 : 1640	들어간 값 : 2625, 낭비된 공간 : 1471	들어간 값 : 3602, 낭비된 공간 : 494
들어간 값 : 871, 낭비된 공간 : 3225	들어간 값 : 3457, 낭비된 공간 : 639	들어간 값 : 3195, 낭비된 공간 : 901
들어간 값 : 300, 낭비된 공간 : 3796	들어간 값 : 1074, 낭비된 공간 : 3022	들어간 값 : 3336, 낭비된 공간 : 760
들어간 값 : 1046, 낭비된 공간 : 3050	들어간 값 : 2925, 낭비된 공간 : 1171	들어간 값 : 1356, 낭비된 공간 : 2740
들어간 값 : 2522, 낭비된 공간 : 1574	들어간 값 : 406, 낭비된 공간 : 3690	들어간 값 : 4019, 낭비된 공간 : 77
들어간 값 : 448, 낭비된 공간 : 3648	들어간 값 : 3596, 낭비된 공간 : 500	들어간 값 : 3096, 낭비된 공간 : 1000
들어간 값 : 1446, 낭비된 공간 : 2650	들어간 값 : 3372, 낭비된 공간 : 724	들어간 값 : 1748, 낭비된 공간 : 2348
들어간 값 : 1326, 낭비된 공간 : 2770	들어간 값 : 1851, 낭비된 공간 : 2245	들어간 값 : 1001, 낭비된 공간 : 3095
들어간 값 : 486, 낭비된 공간 : 3610	들어간 값 : 825, 낭비된 공간 : 3271	들어간 값 : 2835, 낭비된 공간 : 1261
들어간 값 : 2638, 낭비된 공간 : 1458	들어간 값 : 3857, 낭비된 공간 : 239	들어간 값 : 3645, 낭비된 공간 : 451

들어간 값 : 254, 낭비된 공간 : 3842	들어간 값 : 393, 낭비된 공간 : 3703	들어간 값 : 2796, 낭비된 공간 : 1300
들어간 값 : 4075, 낭비된 공간 : 21	들어간 값 : 1078, 낭비된 공간 : 3018	들어간 값 : 2869, 낭비된 공간 : 1227
들어간 값 : 1506, 낭비된 공간 : 2590	들어간 값 : 3835, 낭비된 공간 : 261	들어간 값 : 1635, 낭비된 공간 : 2461
들어간 값 : 718, 낭비된 공간 : 3378	들어간 값 : 1898, 낭비된 공간 : 2198	들어간 값 : 3955, 낭비된 공간 : 141
들어간 값 : 296, 낭비된 공간 : 3800	들어간 값 : 1796, 낭비된 공간 : 2300	들어간 값 : 3906, 낭비된 공간 : 190
들어간 값 : 189, 낭비된 공간 : 3907	들어간 값 : 35, 낭비된 공간 : 4061	들어간 값 : 2905, 낭비된 공간 : 1191
들어간 값 : 3461, 낭비된 공간 : 635	들어간 값 : 2086, 낭비된 공간 : 2010	들어간 값 : 2978, 낭비된 공간 : 1118
들어간 값 : 1004, 낭비된 공간 : 3092	들어간 값 : 1160, 낭비된 공간 : 2936	들어간 값 : 2423, 낭비된 공간 : 1673
들어간 값 : 3281, 낭비된 공간 : 815	들어간 값 : 1038, 낭비된 공간 : 3058	들어간 값 : 2315, 낭비된 공간 : 1781
들어간 값 : 1961, 낭비된 공간 : 2135	들어간 값 : 1271, 낭비된 공간 : 2825	들어간 값 : 1535, 낭비된 공간 : 2561
들어간 값 : 1577, 낭비된 공간 : 2519	들어간 값 : 3120, 낭비된 공간 : 976	들어간 값 : 3615, 낭비된 공간 : 481
들어간 값 : 3281, 낭비된 공간 : 815	들어간 값 : 2614, 낭비된 공간 : 1482	들어간 값 : 1133, 낭비된 공간 : 2963
들어간 값 : 1961, 낭비된 공간 : 2135	들어간 값 : 3507, 낭비된 공간 : 589	들어간 값 : 273, 낭비된 공간 : 3823
들어간 값 : 1577, 낭비된 공간 : 2519	들어간 값 : 2654, 낭비된 공간 : 1442	들어간 값 : 1938, 낭비된 공간 : 2158
들어간 값 : 2237, 낭비된 공간 : 1859	들어간 값 : 1193, 낭비된 공간 : 2903	들어간 값 : 3740, 낭비된 공간 : 356
들어간 값 : 3631, 낭비된 공간 : 465	들어간 값 : 2915, 낭비된 공간 : 1181	들어간 값 : 12, 낭비된 공간 : 4084
들어간 값 : 2676, 낭비된 공간 : 1420	들어간 값 : 2834, 낭비된 공간 : 1262	들어간 값 : 464, 낭비된 공간 : 3632
들어간 값 : 3505, 낭비된 공간 : 591	들어간 값 : 2420, 낭비된 공간 : 1676	들어간 값 : 780, 낭비된 공간 : 3316
들어간 값 : 181, 낭비된 공간 : 3915	들어간 값 : 2608, 낭비된 공간 : 1488	들어간 값 : 3207, 낭비된 공간 : 889
들어간 값 : 2623, 낭비된 공간 : 1473	들어간 값 : 3836, 낭비된 공간 : 260	들어간 값 : 64, 낭비된 공간 : 4032
		들어간 값 : 744, 낭비된 공간 : 3352

[자료구조 - 3 번 (복합문제 1.3)]

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)

[자료구조 - 4 번]

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

[자료구조 - 5 번]

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

[자료구조 - 6 번]

17. AVL 트리를 재귀 호출을 사용하여 구현하도록 한다.

18. AVL 트리를 재귀 호출 없이 구현하도록 한다.

[자료구조 - 7 번]
19. Red Black 트리와 AVL 트리를 비교해보도록 한다.

[자료구조 - 8 번]
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.

답)

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

typedef struct __queue{
	int data;
	struct __queue *link;
}queue;

queue *get_node(void)
{
	queue *tmp;
	tmp=(queue *)malloc(sizeof(queue));
	tmp->link=NULL;
	return tmp;
}

int mk_rand(int arr[20],int i)
{
	int num,count;
	num=rand()%101;
	for(count=0;count<i;count++)
	{
		if(arr[count]==num)
			num=mk_rand(arr,i);
	}
	return num;
}


void print_queue(queue *top)
{
	queue *tmp;
	tmp=top;
	if(tmp->link != NULL)
	{
```

```
            printf("print : %d\n",tmp ->data);
            print_queue(tmp->link);
        }
        else if(tmp-> link ==NULL)
        {
            printf("print : %d\n",tmp->data);
            return ;
        }
}

void enqueue(queue **top,int data)
{

        queue *tmp=NULL;

        if(*top==NULL)
        {
            *top=get_node();
            (*top)->data=data;
            printf("enqueue : %d\n",(*top)->data);
            return ;
        }
        else if((*top)->link!=NULL)
        {
            enqueue(&(*top)->link,data);
        }

        else if((*top)->link==NULL)
        {
            tmp=get_node();
            (*top)->link=tmp;
            tmp->data=data;
            printf("enqueue : %d\n",tmp->data);
        }

}

void dequeue(queue **top,int data)
{
        queue *tmp;
        tmp=*top;
        if((*top)->data==data)
        {
            *top=tmp->link;
            free(tmp);
            return ;
        }
        else
        {
            dequeue(&(*top)->link,data);
        }
}
```

```
int main(void)
{
	queue *top=NULL;
	queue *head=NULL;
	int i,output;
	int arr[20];
	srand(time(NULL));
	for(i=0;i<20;i++)
	{
		arr[i]=mk_rand(arr,i);
		printf("%d\n",arr[i]);
	}

	for(i=0;i<20;i++)
	{
		enqueue(&top,arr[i]);
	}
		print_queue(top);
		printf("dequeue\n");
		printf("빼낼 숫자 : ");
		scanf("%d",&output);
		dequeue(&top,output);
		print_queue(top);
	return 0;
}
```

결과)

```
enqueue : 13
enqueue : 43
enqueue : 25
enqueue : 91
enqueue : 46
enqueue : 89
enqueue : 77
enqueue : 97
enqueue : 35
enqueue : 34
enqueue : 82
enqueue : 6
enqueue : 51
enqueue : 52
enqueue : 60
enqueue : 15
enqueue : 96
enqueue : 16
enqueue : 74
enqueue : 56
print : 13
print : 43
print : 25
print : 91
print : 46
```

print : 89
print : 77
print : 97
print : 35
print : 34
print : 82
print : 6
print : 51
print : 52
print : 60
print : 15
print : 96
print : 16
print : 74
print : 56
dequeue
빼낼 숫자 : 16
print : 13
print : 43
print : 25
print : 91
print : 46
print : 89
print : 77
print : 97
print : 35
print : 34
print : 82
print : 6
print : 51
print : 52
print : 60
print : 15
print : 96
print : 74
print : 56

[자료구조 - 9 번]

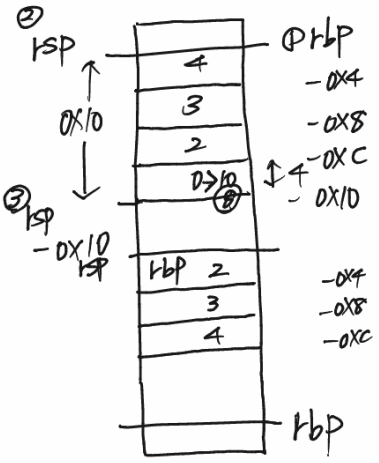
34. 재귀호출을 사용하여 queue 를 구현하고 10, 20 을 집어넣는다.
enqueue 과정에 대한 기계어 분석을 수행하여 동작을 파악하도록 한다.
그림과 함께 자세하게 설명하시오.

```
Dump of assembler code for function main:
   1)    0x00000000004007ec <+0>:     push   %rbp
   2)    0x00000000004007ed <+1>:     mov    %rsp,%rbp
   3)    0x00000000004007f0 <+4>:     sub    $0x10,%rsp
   4)    0x00000000004007f4 <+8>:     mov    %fs:0x28,%rax
   5)  => 0x00000000004007fd <+17>:    mov    %rax,-0x8(%rbp)
   6)    0x0000000000400801 <+21>:    xor    %eax,%eax
                                      //같은 값을 xor 시켜 0 을 eax 에 넣는다.
```

```
7)  0x0000000000400803 <+23>:   movq   $0x0,-0x10(%rbp)
8)  0x000000000040080b <+31>:   lea    -0x10(%rbp),%rax
                                //-0x10(%rbp)에 저장 된 &top 주소를 rax 에 넘겨준다.
9)  0x000000000040080f <+35>:   mov    $0xa,%esi
                                //10 을 esi 에 저장함
10) 0x0000000000400814 <+40>:   mov    %rax,%rdi
11) 0x0000000000400817 <+43>:   callq  0x4006bf <enqueue>
                                0x4006bf 위치에 있는  enqueue 함수로 감
                                복귀주소 : 0x000000000040081c

12) 0x000000000040081c <+48>:   lea    -0x10(%rbp),%rax
13) 0x0000000000400820 <+52>:   mov    $0x14,%esi
14) 0x0000000000400825 <+57>:   mov    %rax,%rdi
15) 0x0000000000400828 <+60>:   callq  0x4006bf <enqueue>
16) 0x000000000040082d <+65>:   mov    $0x0,%eax
17) 0x0000000000400832 <+70>:   mov    -0x8(%rbp),%rdx
18) 0x0000000000400836 <+74>:   xor    %fs:0x28,%rdx
19) 0x000000000040083f <+83>:   je     0x400846 <main+90>
                                ZF 플래그 값이 1 이 되었을 때 0x400846 로 이동
                                소스코드에서 *top 이 NULL 이면 return 한다.
20) 0x0000000000400841 <+85>:   callq  0x4004e0 <__stack_chk_fail@plt>
21) 0x0000000000400846 <+90>:   leaveq
22) 0x0000000000400847 <+91>:   retq

    Dump of assembler code for function enqueue:
23) 0x00000000004006bf <+0>:    push   %rbp
24) 0x00000000004006c0 <+1>:    mov    %rsp,%rbp
25) 0x00000000004006c3 <+4>:    sub    $0x20,%rsp
26) 0x00000000004006c7 <+8>:    mov    %rdi,-0x18(%rbp)
                                &top 를 가져옴
27) 0x00000000004006cb <+12>:   mov    %esi,-0x1c(%rbp)
                                int data 의 값인 10
28) => 0x00000000004006ce <+15>:  movq   $0x0,-0x8(%rbp)
                                tmp 의 NULL
29) 0x00000000004006d6 <+23>:   mov    -0x18(%rbp),%rax
30) 0x00000000004006da <+27>:   mov    (%rax),%rax
31) 0x00000000004006dd <+30>:   test   %rax,%rax
                                rax 와 rax 의 값을 비교해본다.
32) 0x00000000004006e0 <+33>:   jne    0x400719 <enqueue+90>
                                ZF 플래그가 1 이 되면  0x400719 로 이동
33) 0x00000000004006e2 <+35>:   callq  0x400626 <get_node>
                                0x400626 위치의 get_node 함수로 이동
34) 0x00000000004006e7 <+40>:   mov    %rax,%rdx
35) 0x00000000004006ea <+43>:   mov    -0x18(%rbp),%rax
36) 0x00000000004006ee <+47>:   mov    %rdx,(%rax)
37) 0x00000000004006f1 <+50>:   mov    -0x18(%rbp),%rax
38) 0x00000000004006f5 <+54>:   mov    (%rax),%rax
39) 0x00000000004006f8 <+57>:   mov    -0x1c(%rbp),%edx
40) 0x00000000004006fb <+60>:   mov    %edx,(%rax)
41) 0x00000000004006fd <+62>:   mov    -0x18(%rbp),%rax
```

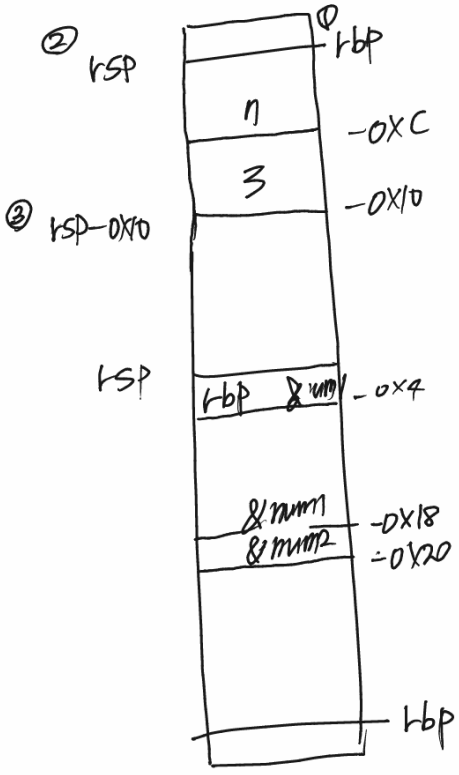
```
42)  0x0000000000400701 <+66>:    mov    (%rax),%rax
43)  0x0000000000400704 <+69>:    mov    (%rax),%eax
44)  0x0000000000400706 <+71>:    mov    %eax,%esi
45)  0x0000000000400708 <+73>:    mov    $0x4008e0,%edi
46)  0x000000000040070d <+78>:    mov    $0x0,%eax
47)  0x0000000000400712 <+83>:    callq  0x4004f0 <printf@plt>
                                  //printf 명령어
48)  0x0000000000400717 <+88>:    jmp    0x40078b <enqueue+204>
                                  //0x40078b 로 이동
49)  0x0000000000400719 <+90>:    mov    -0x18(%rbp),%rax

50)  0x000000000040071d <+94>:    mov    (%rax),%rax
51)  0x0000000000400720 <+97>:    mov    0x8(%rax),%rax
52)  0x0000000000400724 <+101>:   test   %rax,%rax
53)  0x0000000000400727 <+104>:   je     0x400743 <enqueue+132>
                                  값을 비교해보고  0x400743 로 이동한다.
                                  *top->data 에 10 을 넣고 *top 에 주소값도 저장하고 나감
54)  0x0000000000400729 <+106>:   mov    -0x18(%rbp),%rax
55)  0x000000000040072d <+110>:   mov    (%rax),%rax
56)  0x0000000000400730 <+113>:   lea    0x8(%rax),%rdx
57)  0x0000000000400734 <+117>:   mov    -0x1c(%rbp),%eax
58)  0x0000000000400737 <+120>:   mov    %eax,%esi
59)  0x0000000000400739 <+122>:   mov    %rdx,%rdi
60)  0x000000000040073c <+125>:   callq  0x4006bf <enqueue>
61)  0x0000000000400741 <+130>:   jmp    0x40078b <enqueue+204>
62)  0x0000000000400743 <+132>:   mov    -0x18(%rbp),%rax
63)  0x0000000000400747 <+136>:   mov    (%rax),%rax
64)  0x000000000040074a <+139>:   mov    0x8(%rax),%rax
65)  0x000000000040074e <+143>:   test   %rax,%rax
66)  0x0000000000400751 <+146>:   jne    0x40078b <enqueue+204>
67)  0x0000000000400753 <+148>:   callq  0x400626 <get_node>
68)  0x0000000000400758 <+153>:   mov    %rax,-0x8(%rbp)
69)  0x000000000040075c <+157>:   mov    -0x18(%rbp),%rax
70)  0x0000000000400760 <+161>:   mov    (%rax),%rax
71)  0x0000000000400763 <+164>:   mov    -0x8(%rbp),%rdx
72)  0x0000000000400767 <+168>:   mov    %rdx,0x8(%rax)
73)  0x000000000040076b <+172>:   mov    -0x8(%rbp),%rax
74)  0x000000000040076f <+176>:   mov    -0x1c(%rbp),%edx
75)  0x0000000000400772 <+179>:   mov    %edx,(%rax)
76)  0x0000000000400774 <+181>:   mov    -0x8(%rbp),%rax
77)  0x0000000000400778 <+185>:   mov    (%rax),%eax
78)  0x000000000040077a <+187>:   mov    %eax,%esi
79)  0x000000000040077c <+189>:   mov    $0x4008e0,%edi
80)  0x0000000000400781 <+194>:   mov    $0x0,%eax

     Dump of assembler code for function get_node:
81)  0x0000000000400626 <+0>:     push   %rbp
82)  0x0000000000400627 <+1>:     mov    %rsp,%rbp
83)  0x000000000040062a <+4>:     sub    $0x10,%rsp
84) => 0x000000000040062e <+8>:   mov    $0x10,%edi
85)  0x0000000000400633 <+13>:    callq  0x400510 <malloc@plt>
```

```
                                        malloc 은 너무 길어서 생략
86)  0x0000000000400638 <+18>:    mov    %rax,-0x8(%rbp)
87)  0x000000000040063c <+22>:    mov    -0x8(%rbp),%rax
88)  0x0000000000400640 <+26>:    movq   $0x0,0x8(%rax)
89)  0x0000000000400648 <+34>:    mov    -0x8(%rbp),%rax
90)  0x000000000040064c <+38>:    leaveq
91)  0x000000000040064d <+39>:    retq
```



레지스터

edx 4
ecx 3
eax 2 → ⑤ 3 → ⑥ 12 → ⑪ 10 → 0
esi 3 → 10
edi 2 → 0x400624 (출력문자열).
rbp ① → ⑦ → ⑪



eax ⑧ 0 → &num1
rdx &num2
rax &num1 → &num2 → num
→ num2
rsi &num2
rdi &num1
edx &num2

[자료구조 – 10 번(복합문제 2.2)]

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)

답)

```
#include<stdio.h>
#include<malloc.h>
#include<stdlib.h>
#include<time.h>
typedef struct __stack{
	int data;
	struct __stack *link;
}Stack;
Stack *get_node(void)
{
	Stack *tmp;
	tmp=(Stack*)malloc(sizeof(Stack));
	tmp->link=NULL;
	return tmp;
}
int check_rand(int arr[20],int i)
{
	int num,count;
	num=rand()%100+1;
	for(count=0;count<i;count++)
	{
		if(arr[count]==num)
			num=check_rand(arr,i);}
	return num;
}
void push(Stack **top,int data)
{
	Stack *tmp;
	tmp=*top;
	*top=get_node();
	(*top)->data=data;
	(*top)->link=tmp;

}
int pop(Stack **top)
{
	int n;
	Stack *tmp;
	tmp=*top;
	if(*top==NULL)
```

```
        {
                printf("Stack is Empty!");
                return 0;
        }
        *top=tmp->link;
        n=tmp->data;
        free(tmp);
        return n;
}
int main(void)
{
        Stack *top=NULL;
        int i;
        int arr[20];
        srand(time(NULL));
        printf("난수 생성\n");
        for(i=0;i<20;i++)
        {
                arr[i]=check_rand(arr,i);
                printf("%d\n",arr[i]);
        }

        for(i=0;i<20;i++)
        {
                push(&top,arr[i]);
        }
        printf("pop\n");
        for(i=0;i<20;i++)
        {
                printf("data : %d\n",pop(&top));
        }
        return 0;
}
```

결과)

```
난수 생성
62
51
3
31
32
7
73
63
28
77
11
71
39
80
86
6
```

```
pop
data : 60
data : 21
data : 17
data : 49
data : 6
data : 86
data : 80
data : 39
data : 71
data : 11
data : 77
data : 28
data : 63
data : 73
data : 7
data : 32
```

49 17 21 60	data : 31 data : 3 data : 51 data : 62

[자료구조 - 11 번(복합문제 2.2)]
42. 41 번에서 홀수만 빼내서 AVL 트리를 구성하도록 한다.

[자료구조 – 12 번(복합문제 2.3)]
43. 41 번에서 짝수만 빼내서 RB 트리를 구성하도록 한다.

[자료구조 - 13 번]
46. 최적화 프로세스를 기술하도록 한다.

```
gcc -g -o -o0 디버깅파일이름 파일이름.c
```

[자료구조 - 14 번]
47. 기존에는 자료구조에서 숫자값만을 받았다.
이제 Queue 에서 데이터로서 숫자가 아닌 문자열을 받아보자.

[자료구조 - 15 번]
AVL 트리에 데이터로 숫자가 아닌 문자열을 입력하도록 프로그램하시오.

[자료구조 – 16 번]
Binary Tree 에 문자열을 입력한다.

[자료구조 - 17 번]
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