2월27일 수업자료

재귀호출을 while로?

```
int fib (int num)
{
 int first = 1 ;
 int second = 1 ;
 int tmp = 0 ;

 if (num == 1 || num ==2 )
    return 1;

 while ( num-- > 2)
{
 tmp = first + second ;
 first = second ;
 second = tmp;
}
return tmp;
}
```

——

```
#include <stdio.h>

int fact(int num)
{
        if(num == 0 )
                return 1;
        else
                return num*fact(num - 1);
}

int main(void)
{
        int result , fact_val ;
        printf("계산할 팩토리얼을 입력하시오 :");
        scanf("%d",&fact_val);
        result = fact(fact_val);
        printf("%d번째 항의 수는 = %d₩n", fact_val, result);
        return 0;
}
~
~
```

을 while로

```
#include <stdio.h>

int fact(int num)
{
	int first = 1;
	while(num >0)
		first = first*num--;
	return first;
}

int main(void)
{
	int result , fact_val ;
	printf("계산할 팩토리얼을 입력하시오 :");
	scanf("%d",&fact_val);
	result = fact(fact_val);
	printf("%d번째 항의 수는 = %d₩n", fact_val, result);
	return 0;
}
```

————————————————————————————————

배열이 필요한 이유는?

100개의 변수가 필요한 경우를 생각해보자
학교의 학생관리 system에서 명단에 문자열을 담음
다수의 변수에 값을 할당시 for문을 이용 할 수 있음

배열 형태
ex) int sensor_data[100] = {0}; *시작은 0부터라 [100]을 넣었으면 0부터 99까지 라는말.

(int num1_arr[] = {1,2,3,4,5}면 []안의 수는 자동으로 5로 채워짐 컴터가.
근데 예를들어 [i] = i; 로 선언하면 초기화가 필요없음

- 배열은 순차적으로. 그러므로 3차원배열 이런거 없다. 2중배열 3중배열이 올바른 표현.

배열길이를 구하는것은
배열 데이터타입의 크기값을 전체값으로 나눔
예) int len1 = sizeof(num1_arr)/sizeof(int);

```
#include <stdio.h>

int main(void)
{
 int i ;
 int num1_arr[] = {1,2,3,4,5};
```

```
 int num2_arr[3] = { 1,2,3};

int len1 = sizeof(num1_arr)/sizeof(int);
int len2 = sizeof(num2_arr)/sizeof(int);

printf("num1_arr length = %d₩n", len1);
printf("num2_arr length = %d₩n", len2);

for ( i = 0; i<len1; i++)
{
printf("num1_arr[%d] = %d₩n",i,num1_arr[i]);
}
for ( i = 0 ; i < len2; i++)
{
printf("num2_arr[%d] = %d₩n",i,num2_arr[i]);
}
return 0;
}
==>
num1_arr length = 5
num2_arr length = 3
num1_arr[0] = 1
num1_arr[1] = 2
num1_arr[2] = 3
num1_arr[3] = 4
num1_arr[4] = 5
num2_arr[0] = 1
num2_arr[1] = 2
num2_arr[2] = 3

————————————————————————————————

#include <stdio.h>

int main(void)
{
int i ;
int num1_arr[7] = {1,2,3};    ->>>>나머지 4자리는 0으로 초기화됨

for(i = 0; i < 7 ; i++)
{
printf("num1_arr[%d] = %d₩n",i,num1_arr[i]);
}

return 0;
}
```

=>>

num1_arr[0] = 1
num1_arr[1] = 2
num1_arr[2] = 3
num1_arr[3] = 0
num1_arr[4] = 0
num1_arr[5] = 0
num1_arr[6] = 0

————————————————————————————

Character Type Array

char형 배열이 필요한 이유?

string인 문자열 "im marth kim" 은 변경 불가능
char형 배열은 내부 데이터 변경이 가능하다
**마지막 data에 Null Character가 필요함 (문자는 null character가 , 필수 숫자와달리)

Null character는 무엇?

NULL문자는 문자열의 마지막의미

%d 는 마지막. (요즘은 컴파일러가 안써도 자동으로 해줌)

[ ] = " , , ," 할때 , , 들어갈 공간보다 +1 해서 []안의 숫자를 넣줘야 안터지는 안정성이 있음

————————————————————————————
4x4 단위행렬은

```
#include <stdio.h>

int main(void)
{
	int arr[4][4];
	int i,j;

	for(i = 0; i<4 ; i++)
	{
		for(j=0;j<4; j++)
		{
			if(i == j)
				arr[i][j] = 1;
```

```
                else
                        arr[i][j] = 0;
        }
    }
    for(i=0; i<4; i++)
    {
        for(j = 0 ; j <4; j++)
        {
            printf("%d",arr[i][j]);
        }
        printf("₩n");
}
        return 0;
```

다중 배열사용이유

2중배열은 행렬의표현에 용이
[x][y]로 x명의 y개 과목관리가능
3중배열은 위경우에서 반이z개 잇을경우

* int arr[][] 이 경우는 []에 []가 있다는뜻
[4][4]라면 4박스안에 4박스가 더잇음 총 16개

=> arr[2][2][3]이거는 3개짜리가 2개들어있고 그런게 다시 2개있다는 뜻

```
        {
         {{1,2,3},{1,2,3}},
         {{1,2,3},{1,2,3}}
        } ;
```

2중배열

```
#include <stdio.h>

int main(void)
{
    int arr[2][2] = {{10,20},{30,40}};
    int i, j;

    for(i = 0; i<2; i++)
    {
        for(j = 0; j<2; j++)
        {
            printf("arr[%d][%d] = %d₩n",i,j,arr[i][j]);
        }
    }
    return 0;
```

```
}
```

그림을 그리면 간단하게 알수있다 답을.
arr[0][0] = 10
arr[0][1] = 20
arr[1][0] = 30
arr[1][1] = 40
——

*모든것은 주소다.
배열은 주소.
배열의 이름 그자체가 주소다. %p는 주소값알때 쓰임

```
#include <stdio.h>

int main(void)
{
	int arr[4] = {10,20,30,40};
	int i;

	printf("address = %p₩n",arr);

	for(i =0; i < 4; i++)
	{
		printf("address = %p, arr[%d] = %d₩n", &arr[i],i,arr[i]);
	}

	return 0;
}
```

하면 첫번째 arr에 0번째해당 즉 첫번째해당하는게 배열의 대표 이름 주소

포인터 - 메모리주소를 저장할떄 *를 쓴다

```
#include <stdio.h>

int main(void)
{
int arr[3] = {1,2,3};
int *p = arr;

for(i =0; i<3; i++)
 printf ("p[%d] = %d₩n" i , p[i]);

	return 0;
}
```

예제)#include <stdio.h>

```
int main(void)
{
 int arr[3][4];

printf("arr address = %p₩n",arr);
printf("arr[0] address = %p₩n",arr[0]);
printf("arr[1] address = %p₩n",arr[1]);
printf("arr[2] address = %p₩n",arr[2]);


        return 0;
}
```

4바이트씩 4개 라서 1칸옮길때마다 16진수 10 즉 16씩 늘어남 (답 확인해보면)

그걸 확인하는 것 (사이즈체크)

```
#include <stdio.h>

int main(void)
{
int arr[3][4];

printf("arr size = %lu₩n",sizeof(arr));
printf("arr[0] size = %lu₩n", sizeof(arr[0]));
printf("arr[1] size = %lu₩n", sizeof(arr[1]));
printf("arr[2] size = %lu₩n", sizeof(arr[2]));

        return 0;
}
```

실행하면

```
arr size = 48
arr[0] size = 16
arr[1] size = 16
arr[2] size = 16
```

————————————————————————————————

함수의 인자로 배열을 전달하려면?
#include <stdio.h>

```
void arr_print(int arr[])
{
	int i;
	for(i = 0; i <3 ; i++)
		printf("%4d", arr[i]);
}

int main(void)
{
	int arr[3] = { 3 ,33 , 333};
	arr_print(arr);
	return 0;
}
```

어떻게 메인함수에 3을넣고 i<3이 되지?

주소를 알고있으면 스택의 베리어를 건너뛸수있음
--> arr[3]은 main() 함수내의 지역 변수
arr_print로 위 배열의 주소가 전달됨

즉,arr_print 함수에서는
main() 함수의 지역 변수인
arr[3]에 접근하여 값을 수정할 수 있음

* %4d는 4자리만큼 만들어준거 보여줄때

————————————————————————————
함수 어디서든 어떤변수에 대한 제어를 수행하고싶다
=> 포인터 쓰는이유!

```
#include <stdio.h>

void add_arr(int *arr)
{
	int i;
	for( i=0; i<3; i++)
	{
		arr[i] +=7;
	}
}

void print_arr(int *arr)
{
	int i;
	for(i = 0; i < 3; i++)
	{
		printf("arr[%d] = %d₩n", i , arr[i]);
```

```
	}
}

int main(void)
{
	int arr[3] ={1,2,3};
return 0;
}
```

지역변수지만 메인함수의 주소값이 연결되어있으므로
다 가져다 쓸수있음.

포인터 주소값
*(&num) += 30 & -< 주소값
num += 30 이랑 같다

```
#include <stdio.h>

int main(void)
{
	int num = 3 ;

	*(&num) += 30;
	printf("num = %d₩n",num);
	return 0 ;
}
```

문자열상수는 전역변수. data에 저장됨.주소값을 갖고있고 1byte에 순차적으로 값이 있어서
*str2 = "po~~~~"; 하면 받을 수 있음

```
#include <stdio.h>

int main(void)
{
	char str1[33] = "Pointer is important!";
	char *str2 = "Pointer is important!";

	printf("str1 = %s₩n",str1);
	printf("str2 = %s₩n",str2);
	return 0;
}
```

————————————————————————————
(int *) => 이거자체가 데이터타입
 int형에 주소값을 저장할수 있는 타입이다
int*p 해석: 인트형 주소값을 받을수 있는 변수 p다

int *p랑

printf(~~~,*p) 에 *p랑은 다르다.

————————————————————————————
*Segmentation Fault 가 나는 이유?

우리가 기계어를 보면서 살펴봤던 주소값들이
사실은 전부 가짜 주소라고 말했었ㄷ.
이 주소값은 엄밀하게 가상 메모리 주소에 해당하고
운영체제의 Paging 메커니즘을 통해서 실제 물리 메모리의 주소로 변환된다.
(윈도우도 가상 메모리 개념을 베껴서 사용한다)
그렇다며ㄴ 당연히 맥(유닉스)도 사용함을 알 수 있다.

가상 메모리는 리눅스의 경우
32 비트 버전과 64비트 버전이 나뉜다.

32 비트 시스템은 2^32 = 4GB 의 가상 메모리 공간으 가짐
여기서 1:3으로 1을 커널이 3을 유저가 가져간다.
1은 시스템(HW,CPU,SW 각종 주요 자원들)에
관련된 중요한 함수 루틴과 정보들을 관리하게 된다.
3은 사용자들이 사용하는 정보들로
문제가 생겨도 그다지 치명적이지 않는 정보들로 구성됨

64비트 시스템은 1:1로 2>63승에 해당하는 가상메모리를 각각 가진다.
문제는 변수를 초기화하지 않았을 경우 가지는 쓰레기값이 OXCCCCCC..CCCC로
구성됨이다.

32비트의 경우에도 1:3 경계인 0XC000000000000을 넘어가게됨
64비트의 경우엔 시작이 C이므로 이미 1:1경계를 한참 넘어감

그러므로 접근하면 안되는 메모리 영역에 접근하였기에
엄밀하게는 Page Fault(물리 메모리 할당되지 않음)가 발생하게 되고 원래는
Interrupt가
발생해서 Kernel 이 Page Handler(페이지 제어기)가 동작해서
가상 메모리에 대한 Paging 처리를 해주고 실제 물리 메모리를 할당해 주는데
문제는 이것이 User 쪽에서 들어온 요청이므로
Kernel 쪽에서 강제로 기각해버리면서
Segmentation Fault가 발생하는 것이다.

실제 Kernel 쪽에서 들어온 요청일 경우에는
위의 매커니즘에 따라서 물리 메모리를 할당해주게 된다.

ㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡㅡ

포인터에 대한 포인터

num = 3
int num;
int *p = #
int**pp = &p;
를 가져오는 방법3가지
printf("num")
("*p")
("**pp")

예)

```
#include <stdio.h>

int main(void)
{
	int num = 3;
	int *p = &num;
		int **pp = &p;

	printf("num = %d₩n", num);
	printf("*p = %d₩n", *p);
	printf("**pp = %d₩n", **pp);

	return 0;
}
```

예)

```
#include <stdio.h>

int main(void)
{
	int num1 = 3 , num2 = 7 ;
	int *temp = NULL;
		int *num1_p = &num1;
	int *num2_p = &num2;
	int **num_p_p = &num1_p;

	printf("*num1_p = %d₩n", *num1_p);
	printf("*num2_p = %d₩n", *num2_p);

	temp = *num_p_p;
```

```
        *num_p_p = num2_p;
        num2_p = temp;

        printf("*num1_p = %d₩n", *num1_p);
        printf("*num2_p = %d₩n", *num2_p);

        return 0;
}
~
~
~
```
돌리면

```
*num1_p = 3
*num2_p = 7
*num1_p = 7
*num2_p = 3
```

————————————————————————————————

포인터배열
포인터를 저장하는 배열

둘의차이점
int(*p)[2] int형 2개짜리 , 포인터 8바이트 (= int(*)[2] p) ==> 8바이트에 대한 포인터
 int *p[2]

```
#include <stdio.h>

int main(void)
{
        int i,j,n1,n2,n3;
        int a[2][2] = {{10,20},{30,40}};
        int *arr_ptr[3] = {&n1,&n2,&n3};
        int (*p)[2] = a;

        for(i =0; i<3; i++)
                *arr_ptr[i] = i;

        for(i = 0 ; i < 3; i++)
                printf("n%d = %d₩n",i,*arr_ptr[i]);

        for(i = 0 ; i < 2 ; i++)
                printf("p[%d] = %d₩n",i,*p[i]);

                return 0;
```

```
}
```