

1 번

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 ?

1 번답 :

callq

callq 는 push 와 jump 동시에 있는 명령어다

함수 호출시 push 와 jump 는 필수이다.

jump 는 다음 실행할 명령어의 주소를 저장함

---

2 번

goto 의 정체성은 ?

전세계 각지의 천재들이 모여서 개발하는 리눅스 운영체제 코드에는 엄청 많은 양의 goto 가 사용되고 있다.

goto 를 도대체 왜 사용해야만 할까 ?

2 번답 :

goto 사용이유 GOTO 문의 남용은 소스코드의 이해를 어렵게 만들지만,

적절한 사용은 오히려 소스코드의 가독성과 명료성을 높이는 경우가 있다.

가령 다중 반복문에서의 탈출, 에러에 대한 예외 처리 등 일부 작업에 한해서는 GOTO 문을 사용하는 경우가 더 명료한 경우도 있다.

<프로그램을 작성하다 보면 중간의 코드는 무시하고 원하는 부분으로 건너뛰어야 하는 상황이 생기는데

이럴때 사용하는것이 goto.>

<스파게티 코드 : goto 를 과도하게 사용하여 프로그램의 흐름이 마치 스파게티 면발처럼 꼬여있다는 데서 붙여진 이름.>

3 번

포인터 크기에 대해 알고 있는대로 기술하시오.

3 번답 :

1. 포인터 변수의 크기는 4 바이트로 고정된 것이 아니다.
2. 운영체제의 bit 수가 포인터 변수의 크기에 영향을 미치는 것은 아니다.
3. 포인터 변수의 크기(double \*,char \*, int\*)의 크기는 컴파일러의 상황에 따라 다르다-32bit 로 컴파일 시, 4 바이트로 모두 같고-64bit 로 컴파일 시, 8 바이트로 모두 같다.
4. 64bit 의 운영체제에서 32bit 로 컴파일 시 포인터변수의 크기가 4 바이트로 나타난다.
5. 64bit 의 운영체제에서 64bit 로 컴파일 시 포인터변수의 크기가 8 바이트로 나타난다.

<운영체제에 따라 포인터변수의 크기가 정해지는 것이 아님.>

-----

7 번

이중 배열을 함수의 인자로 입력 받아 3 by 3 행렬의 곱셈을 구현하시오.

```
1 #include<stdio.h>
2
3 void func(int *arr1,int *arr2)
4 {
5     int i,j;
6     int sum[3][3]={0.};
7     printf("두 배열의 합은 : \n");
8     for(i=0;i<3;i++)
9     {
10         for(j=0;j<3;j++)
11         {
12             sum[i][j]=arr1[i][0]*arr2[0][i]+arr1[i][1]*arr2[1][j]+arr1[i][2]*arr2[2][j];
13
14             printf("%d",sum);
15         }
16     }
17
18
19
20
21
22 int main()
23 {
24     int arr1[3][3]={{1,1,1},{1,1,1}};
25     int arr2[3][3]={{1,2,3},{4,5,6}};
26     func(arr1,arr2);
27     return 0;
28 }
```

9 번

함수 포인터를 반환하고 함수 포인터를 인자로 취하는 함수의 주소를 반환하고

인자로 int 2 개를 취하는 함수를 작성하도록 한다.(잘몰라서 비슷하게 했습니다...)

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

struct person{
    char name[20];
    int age;
    char address[40];
};

void changeperson(struct person *p1)
{
    strcpy(p1->name,"김 김 민 민 호 호 ");
    p1->age=36;
    strcpy(p1->address,"서울 시 서 초 구 서 초 동 ");
}

int main()
{
    struct person p1;
    strcpy(p1.name,"김 민 호 ");
    p1.age=28;
    strcpy(p1.address,"서울 시 중 랑 구 상 봉 동 ");

    printf("-----변경 전 신상 -----\\n");
    printf("이름 :%s\\n",p1.name);
    printf("나 이 :%d\\n",p1.age);
    printf("주 소 :%s\\n",p1.address);
    printf("\\n\\n\\n\\n\\n");

    changeperson(&p1);
    printf("-----변경 후 신상 -----\\n");
    printf("이름 :%s\\n",p1.name);
    printf("나 이 :%d\\n",p1.age);
    printf("주 소 :%s\\n",p1.address);
    printf("\\n\\n\\n\\n\\n");
    return 0;
}
```

```
minking@minking-Z20NH-AS51B1U:~/test$ ./a.out
-----변경 전 신상 -----
이름 :김 민 호
나 이 :28
주 소 :서울 시 중 랑 구 상 봉 동

-----변경 후 신상 -----
이름 :김 김 민 민 호 호
나 이 :36
주 소 :서울 시 서 초 구 서 초 동

minking@minking-Z20NH-AS51B1U:~/test$
```

12 번

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

```
0
1
2 #include<stdio.h>
3
4 void addfunc(int *arr1,float *arr2)
5 {
6     int i,j;
7     int sum=0;
8     float sum1=0;
9     for(i=0;i<3;i++)
10     {
11         for(j=0;j<3;j++)
12         {
13             sum=sum+arr1[i][j];
14             sum1=sum1+arr2[i][j];
15         }
16     }
17     printf("인트 배열의 합은 %d",sum);
18     printf("실수 배열의 합은 %f",sum1);
19 }
20
21
22
23
24
25
26
27
28
29 int main()
30 {
31
32     int arr1[3][3]={1,1,1},{1,1,1},{1,1,1};
33     float arr2[3][3]={3.1,1.1,7.7},{1.1,1.3,8.5},{5.1,1.2,1.3}};
34     addfunc(arr1,arr2);
35
36
37     return 0;
38 }
39
```

13 번

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.

```
232
233 #include<stdio.h>
234 int func(int num)
235 {
236     if(num ==1 || num ==2)
237     {
238
239         return 1;
240     }
241     else
242         return func(num-1) + func(num-2);
243 }
244
245 int main()
246 {
247     int i,k,b;
248     int res1=0,res2=0,arr[27] ;
249
250     for(i=0 ; i <27 ; i++)
251     {
252         arr[i] = func(i+1);
253     }
254     for(k=0 ; k<27; k++)
255     {
256
257         if(!(k%2))
258         {
259             res1 = res1 + arr[k];
260         }
261     }
262     for(b=0 ; b<27; b++)
263     {
264         if(b%2)
265         {
266
267             res2 = res2 + arr[b];
268         }
269     }
270     printf("27항 까지 홀수들의 합 : %d\n",res1);
271     printf("27항 까지 짝수들의 합 : %d\n",res2);
272     printf("홀수들의 합 - 짝수들의 합 : %d\n",res1-res2);
273     return 0;
274 }
```

```
collect2: error: ld returned 1 exit status
minking@minking-Z20NH-AS51B1U:~/C
minking@minking-Z20NH-AS51B1U:~/C
minking@minking-Z20NH-AS51B1U:~/C
minking@minking-Z20NH-AS51B1U:~/C
minking@minking-Z20NH-AS51B1U:~/C
27항 까지 홀수들의 합 : 317811
27항 까지 짝수들의 합 : 196417
홀수들의 합 - 짝수들의 합 : 121394
minking@minking-Z20NH-AS51B1U:~/C
```

14 번

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

```
380 /
381
382 #include<stdio.h>
383
384 int fib(int final_val)
385 {
386     if(final_val==1 || final_val==2)
387         return 1;
388     else
389         return fib(final_val-1)+fib(final_val-2);
390 }
391
392 }
393
394
395
396 int main()
397 {
398     int result;
399     int final_val;
400     printf("피보나치 수열의 항의 개수를 입력하시오 : ");
401     scanf("%d",&final_val);
402     result=fib(final_val);
403     printf("%d번째 항의 수는=%d\n",final_val,result);
404     return 0;
405 }
406
407
408 █
409
410
411
412
```

```
minking@minking-Z20NH-AS51B1U:~/test$ ./a.out
피보나치 수열의 항의 개수를 입력하시오 : 23
23번째 항의 수는=28657
minking@minking-Z20NH-AS51B1U:~/test$ █
```

15 번

Intel Architecture 와 ARM Architecture 의 차이점은 ?

15 번 답

## arm

주로 모바일이나 휴대하기 쉬운 전자모델에 쓰이는것은 arm 프로세서이다.

저전력 작동을 위해 설계되어 있기 때문에 배터리를 효율적으로 쓸수있다.

가능한 단순하게 설계되어 있어 에너지 낭비를 최소화 한다.

최대한 간소화한 아키텍처가 특징이며 군더더기가 없다.

## 인텔

데스크탑이나 고성능이 필요한 전자모델에 쓰인다.

x86 은 소프트웨어 개발이나 하드웨어, 인프라스트럭처 측면에서도 호환성을 갖음.(arm 플랫폼은 안됨)

명령어셋이 복잡해서 arm 보다 효율성이 떨어진다.

x86 의 경우 성능향상을 위해 추가기능을 많이 붙여냈기때문에 arm 에 비해 최적화가 되어있지 않다.(임베디드에서)

19 번

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

19 번 답

`int *p[3]` p 는 int 형을 가리키는 포인터를 3 개저장하는 배열이다.

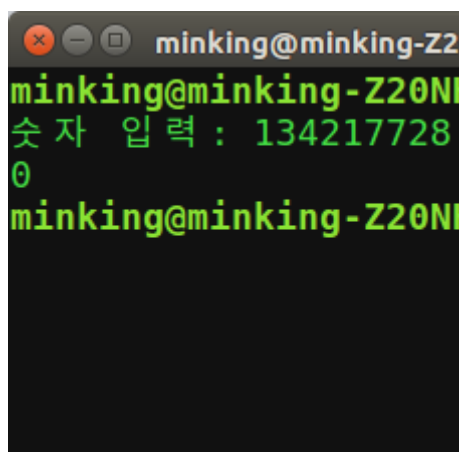
`int (*p)[3]` p 는 int 형을 3 개저장하고 있는 배열을 가리키는 포인터다.

20 번

임의의 값 x 가 있는데, 이를 134217728 단위로 정렬하고 싶다면 어떻게 해야할까 ?

어떤 숫자를 넣던 134217728 의 배수로 정렬이 된다는 뜻임

```
37
38
39 #include<stdio.h>
40
41 int func(int num)
42 {
43
44     return ~(134217728)&num;
45 }
46
47
48
49
50 int main()
51 {
52     int num;
53     int abc;
54     printf("숫자 입력 : ");
55     scanf("%d",&num);
56     abc=func(num);
57     printf("%d\n",abc);
58     return 0;
59 }
60
```



```
minking@minking-Z20N
minking@minking-Z20N
숫자 입력 : 134217728
0
minking@minking-Z20N
```



## 21 번단

한번의 연산으로 대소문자 변환을 할 수 있는 연산에 대해 기술하시오.

```
1 //30번 -----
2
3
4 #include<stdio.h>
5
6 void func1(char abc,char xor)
7 {
8     printf("알 파 벳 입 력 하 세 요  : ");
9     scanf ("%c",&abc);
10    printf("변 경 전 : %c\n",abc);
11    abc^=xor;
12    printf("변 경 후 : %c\n",abc);
13
14 }
15
16
17 int main()
18 {
19     char abc;
20     char xor=0x20;
21     func1(abc,xor);
22     // printf("알 파 벳 입 력  : ");
23     // scanf ("%c",&abc);
24
25     //printf("변 경 전 : %c\n",abc);
26     //abc^=xor;
27
28     //printf("변 경 후 : %c\n",abc);
29
30     return 0;
31 }
32
```

```
minking@minking-Z20NH-AS51B1U:~/시 험 $ ./a.out
알 파 벳 입 력 하 세 요  : v
변 경 전 : v
변 경 후 : V
minking@minking-Z20NH-AS51B1U:~/시 험 $ █
```

22 번

변수의 정의

변수란 쉽게 말하면 값을 담아두는 기억 공간.

변수는 하나의 데이터 값을 가질수 있으며,한번 정해진 값은 고정되어 있는게 아니라 계속 변할수 있는 수입니다.

이는 즉, 대입되는 수가 항상 변할수가 있다는 말입니다.

이 변수는 담기는 음식에 따라 그릇이 달라지듯 담기는 데이터에 따라 변수의 자료형(Data Type)이 결정됩니다.

문자는 문자를 담을수 있는 자료형이 있으며, 정수는 정수를 담을수 있는 자료형이, 실수는 실수를 담을수 있는 자료형이 따로 존재합니다.

char : 1 바이트 /short : 2 바이트/int 4 바이트/long 4 바이트/long long 8 바이트/float 4 바이트/double 8 바이트 long double 8 바이트이상

23 번

포인터의 정의

포인터는 어떠한 값을 저장하는게 아니라 어떠한 값의 주소를 저장하는 것이다.

주소라함은 해당값의 저장된 컴퓨터의 메모리상의 주소를 의미한다.

24 번

함수포인터의 정의

함수를 호출하려면 함수이름으로 직접 호출했다.

함수를 배열또는 구조체에 넣거나, 함수자체를 함수의 매개변수로 넘겨주고 , 반환값으로 가져오기 위하여함수포인터를 사용한다.

26 번

메모리 계층구조

레지스터

캐시

메모리

하드디스크

27 번

c 언어 메모리구조

스택 - 프로그램이 자동으로 사용하는 임시메모리 영역 지역함수,매개변수,리턴값이 할당되는 영역

힙- 동적으로 메모리를 할당한곳

데이터-전역변수,지역변수 초기값이 없는 변수는 data 영역에 초기값이 없는경우 bss 영역에 할당.

코드(텍스트)- 프로그램이 위치한 부분으로 코드가 메모리에 로드되면 이 부분에서 실행.

29 번

이름과 급여를 저장하도록 만든다.

이름은 마음대로 고정시키도록 하고 급여는 rand() 를 활용

급여의 평균을 출력하고 가장 높은 한 사람의 이름과 급여를 출력하시오.

```
92
93 #include<stdio.h>
94 #include<time.h>
95 #include<stdlib.h>
96 struct person
97 {
98     char name[50];
99     int pay;
100 };
101
102 void func1(struct person *p1)
103 {
104     int i;
105     int sum;
106     for(i=0;i<7;i++)
107     {
108
109         printf("사람 이름 입력 (7명): ");
110         scanf("%s",p1[i].name);
111         p1[i].pay=rand()%300+100;
112         printf("%d\n",p1[i].pay);
113     }
114 }
115
116 void func2(struct person *p1)
117 {
118     int i;
119     int sum=0;
120     int ave;
121     for(i=0;i<7;i++)
122     {
123         sum=sum+p1[i].pay;
124     }
125     ave=sum/7;
126     printf("평균 월급은 : %d\n",ave);
127     printf("평균 월급 이상인 사람은 : ");
128     for(i=0;i<7;i++)
129     {
130         if(p1[i].pay>ave)
131         {
132             printf("%s\n",p1[i].name);
133         }
134     }
135 }
136
137 int main()
138 {
139     struct person p1[7];
140     srand(time(NULL));
141     func1(p1);
142     func2(p1);
143     return 0;
144 }
```

```
minking@minking-Z20NH-AS51B1U:~/test$ ./
사람 이름 입력 (7명): 김병지1
261
사람 이름 입력 (7명): 김병균2
261
사람 이름 입력 (7명): 김병균3
110
사람 이름 입력 (7명): 김민호4
382
사람 이름 입력 (7명): 김병지아들5
369
사람 이름 입력 (7명): 김병지6
389
사람 이름 입력 (7명): 김병지병지7
239
평균 월급은 : 287
평균 월급 이상인 사람은 : 김민호4
김병지아들5
김병지6
minking@minking-Z20NH-AS51B1U:~/test$
```

30 번

리눅스에서 디버깅을 수행하기 위한 컴파일 방법

30 번답

gcc -g -o 바꿀이름 만든파일명.c 를하면

바꿀이름으로 디버깅할수있는 파일이 만들어지고

gdb 바꿀이름 하면 디버깅을 할수있다.

(최적화 x , 최적화 는 다른 문제에 있어 적지 않았습니다.)

31 번

vi 에서 코드가 정렬이 잘 안되어 있다.

이 상황에서 어떻게 해야 예쁘고 깔끔하게 정렬되는가 ?

1. insert 상태에서 벗어나서
2. v 를 누르고 정렬을 원하는곳까지 화살표 커서를 내린다
3. '=' 키를 누르면 보기 좋게 정렬이 된다.

처음부터 루트를 건들어서 들여쓰기 설정을 해주는 방법도 있다.

32 번

프로그램을 최적화하는 컴파일 옵션을 적고

반대로 어떠한 최적화도 수행하지 않는 컴파일 옵션을 적도록 한다.

최적화 : gcc -O5 :사이즈를 최적화 함(공간이 협소한 곳에서 사용 - 임베디드)

최적화 x : gcc -O0:최적화를 수행하지 않음.

33 번

gdb 를 사용하는 이유를 기술하시오.

1. 사람의 오타
2. 특수 케이스를 생각지 못한 논리적 오류
3. os, 다른 라이브러리의 버그로 인한 오류하드웨어의 설계로 인한 오류등등이 있지만
4. 가장 큰 이유는 논리적인 오류 때문이다.(문법적으로는 문제가 없으나 실행해보면 문제가 생긴다.)

34 번

기계어 레벨에서 스택을 조정하는 명령어는 어떤것들이 있는가 ?

1.push ,2. pop 은 실행시마다 sp 값이 증가한다(밑으로)

sp 를 직접 조정하는것도 스택이 밑으로 증가한다.

3. callq : Push+jump (push 가 들어있으니 sp 밑으로)

4. retq: pop +ripsi (pop 이 들어있으니 sp 밑으로)

### 36 번 가위바위보

```
0
1 */
2 //54번
3
4
5 #include <stdio.h>
6 #include<stdlib.h>
7 #include<time.h>
8 void func1(int com,int you)
9 {
10     com=rand()%3+1;
11     printf("1.주먹 2.가위 3.보\n");
12     scanf("%d",&you);
13     switch(you){
14         case 1 : if(com==1) printf("비겼습니다.");
15                  else if(com==2) printf("당신이 이겼습니다.");
16                  else printf("컴퓨터가 이겼습니다.");
17                 break;
18         case 2 : if(com==2) printf("비겼습니다.");
19                  else if(com==3) printf("당신이 이겼습니다.");
20                  else printf("컴퓨터가 이겼습니다.");
21                 break;
22         case 3 : if(com==3) printf("비겼습니다.");
23                  else if(com==1) printf("당신이 이겼습니다.");
24                  else printf("컴퓨터가 이겼습니다.");
25                 break;
26         default : printf("잘못 입력하셨습니다.");
27     }
28 }
29
30 int main()
31 {
32     int com, you;
33     srand(time(NULL));
34     func1(com,you);
35     return 0;
36 }
37 //22번
```

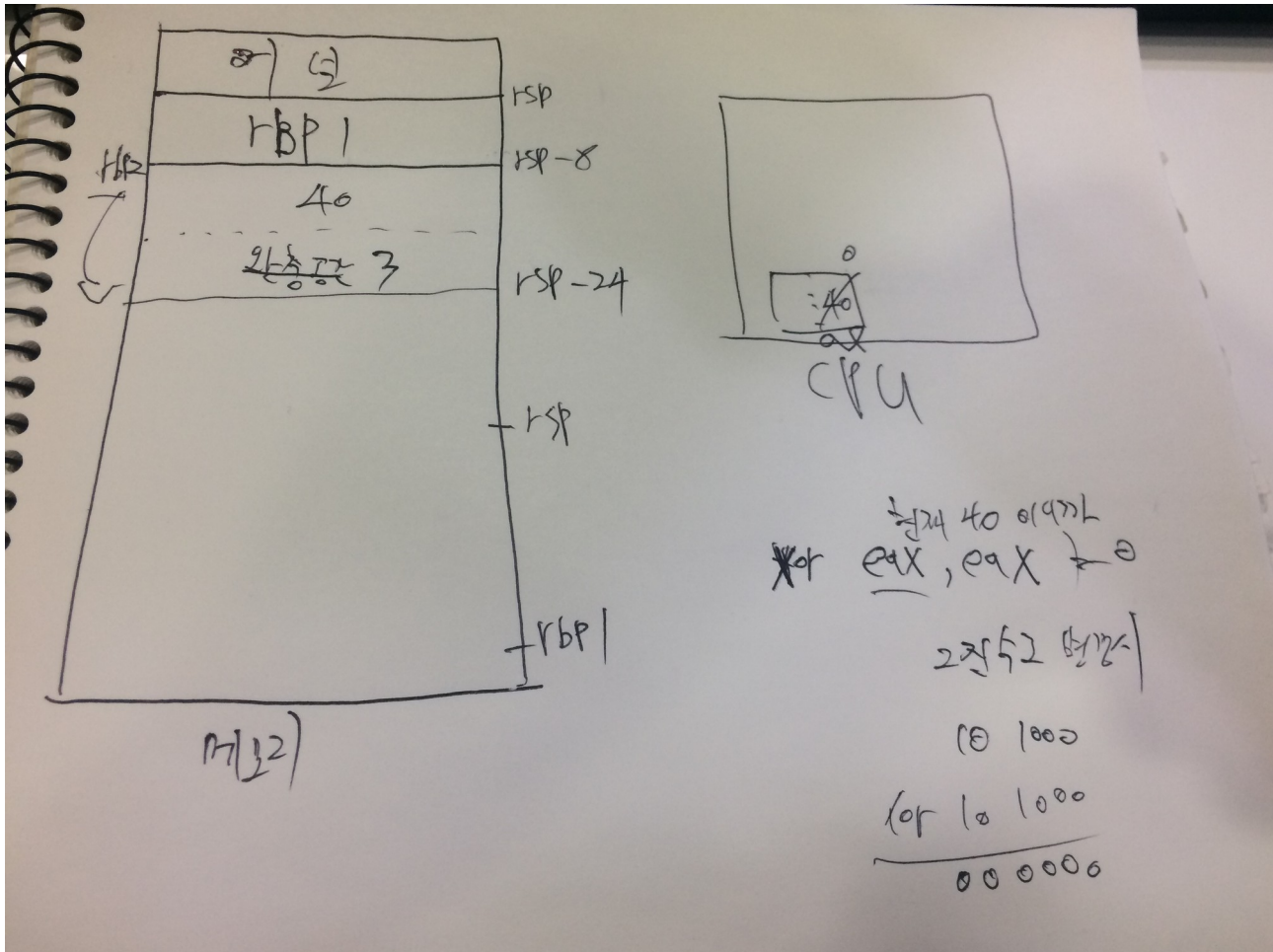
```
minking@minking-Z20NH-AS51B1U:~/시험 $ ./a.out
1.주먹 2.가위 3.보
2
컴퓨터가 이겼습니다.minking@minking-Z20NH-AS51B1U:~/시험 $ ./a.out
1.주먹 2.가위 3.보
1
컴퓨터가 이겼습니다.minking@minking-Z20NH-AS51B1U:~/시험 $ ./a.out
1.주먹 2.가위 3.보
3
당신이 이겼습니다.minking@minking-Z20NH-AS51B1U:~/시험 $ ./a.out
1.주먹 2.가위 3.보
2
당신이 이겼습니다.minking@minking-Z20NH-AS51B1U:~/시험 $ ./a.out
1.주먹 2.가위 3.보
3
비겼습니다.minking@minking-Z20NH-AS51B1U:~/시험 $ █
```

## 41 번 기계어

```
http://minking.org/contents/gdb/development/
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from deb...done.
(gdb) r
Starting program: /home/minking/시험/deb
[Inferior 1 (process 10320) exited normally]
(gdb) b main
Breakpoint 1 at 0x40057b: file test1.c, line 294.
(gdb) r
Starting program: /home/minking/시험/deb

Breakpoint 1, main () at test1.c:294
294      {
(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  0
   0x00000000004005c5 <+82>:     retq
End of assembler dump.
(gdb) █
```





28 번

우리가 사용하는 윈도우나 리눅스, 맥(유닉스)에서 보는 변수 주소는 진짜 주소일까 아닐까?

알고 있는대로 기술하시오.

```
int num=7;
```

```
printf("%p\n",num);
```

```
return 0;
```

하면 주소가 나오는데 컴퓨터 실행할때 마다 바뀐다.

가상메모리기 때문에 계속 주소가 바뀐다.

컴파일 해보면 계속 주소가 다르다.

25 번

파이브라인은 언제 깨지는가?

Mov mov mov

mov mov mov

call call call

mov mov push push push  
mov mov mov  
mov

저곳 move 가 손실

잘몰라서 은태영군 깃허브 참고해서 아무거라도 썼습니다 ...

자료구조 1

값이 1 ~ 4096 까지 무작위로 할당되어 배열에 저장되도록 프로그래밍 하시오.

배열의 크기는 100 개 정도로 잡는다

```
64 // 1 ~ 4096 까지 무작위로 할당되어 배열에 저장되도록 프로그래밍 하시오.
65
66 #include<stdio.h>
67
68 void func(int *a)
69 {
70     int i;
71
72     for(i=0;i<100;i++)
73     {
74
75         a[i]=rand()%4096+1;
76         printf("%d\n",a[i]);
77     }
78
79 }
80
81
82 int main()
83 {
84     int a[100];
85     func(a);
86     return 0;
87 }
88
```

```
minking@minking-Z20NH-AS51B1U:~/시 험 $ ./a.out
1384
967
2154
2164
3154
3328
1099
2285
3882
3278
2235
1964
499
3836
2532
327
125
707
2133
2041
796
2537
3560
910
3959
1371
2351
612
564
1952
1226
1947
2919
3379
14
1976
2610
1113
164
2395
294
2398
262
792
2137
2794
1119
2261
3500
3251
206
199
1692
3765
1109
1554
1039
3459
2165
1602
1314
3390
3549
136
2673
3562
```

자료구조 9 번 que 그림과 함께 설명

```

1 #include <stdio.h>
2 #include <malloc.h>
3 #define EMPTY 0
4 struct __queue
5 {
6     int data;
7     struct __queue *link;
8 };
9 typedef struct __queue queue;
10
11 queue *get_node()
12 {
13     queue *tmp;
14     tmp =(queue *)malloc(sizeof(queue));
15     tmp -> link = EMPTY;
16     return tmp;
17 }
18
19 void enqueue(queue **head, int data)
20 {
21     if(*head == NULL)
22     {
23         *head = get_node();
24         (*head) -> data = data;
25
26         return;
27     }
28     enqueue(&(*head) -> link, data);
29 }
30
31 int print_queue(queue *head)
32 {
33     queue *tmp;
34     tmp = head;
35     while(tmp)
36     {
37         printf("%d\n", tmp -> data);
38         tmp = tmp -> link;
39     }
40 }
41
42
43 int main(void)
44 {
45     queue *head = EMPTY;
46     enqueue(&head, 10);
47     enqueue(&head, 20);
48     //enqueue(&head, 30);
49     print_queue(head);
50
51     return 0;
52 }
53

```

main  
head (1000)

end  
head (1000) data (10)

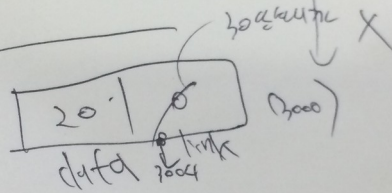
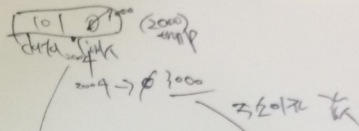
get node  
2000  
tmp

end  
head (1000) data (20)

end (2000)  
head (2000) data (20)

get node

3000  
tmp



13 번 회사에서 프로그래밍 최적화 하는 방법(사이트에서 복사해왔습니다 ... 한마디라도 적는게 나을것 같았음)

<1> Option Explicit 변수선언요구를 설정해 두고 코딩을 하라.

(도구-옵션에서 설정 가능함)

<2> 변수의 Variant 사용을 제한하라.

(Long -> Integer -> Byte 순으로 사용 32BIT CPU)

<3> VB 에서 자체적으로 제공하는 함수들을 최대한 활용하라.

;예를 들어, 24-Feb-1973 이란 형태의 날짜형식을 1973-02-24 일로 바꾼다고 할 때, 일반적으로 Select Case 문과 문자열함수 등을 이용하여 코딩을 할 수도 있지만, VB 에서 제공하는 Format() 함수를 이용하면 단 한 줄로 처리가 가능하다. MsgBox Format("24-Feb-1973","yyyy-MM-dd")

<4> 사용하는 컨트롤들을 하나씩 만들어 사용하지 말고 배열로 생성시켜 활용하라.(메모리와 디스크하드절약)

<5> 변수명은 최대한 짧게, 헝가리안 표기법을 준수하라.

<6> 한글변수명은 피하고, 한글은 데이터로만 사용.

<7> Code 상에서 아래와 같이 Property 연산은 하지 말 것.

```
For i= 1 To 10  
Text1.Text = Text1.Text + 1  
Next i
```

<8> IF 문 보다는 Select Case 문이 더 효율적이며, 최종적으로는 Choose 문을 사용하라.

<9> 상수를 많이 사용할 것. (상수는 컴파일 될 때 한번만 처리됨)

<10> 너무 길어서 생략함 ...

<11> Nothing 을 이용하여 불필요한 메모리를 해제하라.

(Set Form = Nothing)

<12> 경제적 컨트롤을 주로 사용하라.

(Label > Textbox , Image > Picture)

<13> Erase 문을 사용하여 동적배열을 비워라.

(Erase DArray)

<14> 폼,컨트롤 AutoRedraw 를 True 로 하면 폼/컨트롤 이미지를 메모리에 저장해 둔다.

<15> 배포판 작업 전 소스화면을 모두 닫고, 재부팅 한 후 컴파일 하라.

<16> 불필요한 코드는 제거하라.

주석 ' 은 실행에 아무런 영향을 주지 않으니 많이 사용해도 무방하다.

\* 출처 <http://YAHOO.sy.to>

하고싶은말

알면서도 모르는 문제가 많네요 ... 피보나치 수열인데 앞에 1 이 없어서 ... 응용도 못하고 그냥 피보나치 제출 하는등... 무튼

한달 정도 교육받았는데 참 잘왔다는 생각이 듭니다. 물론 따라가는게 남들보다 느리고 힘들지만 하루하루 집에 갈 때마다 보람차다는 생각이 듭니다.

생각보다 선생님 나이가 어리셔서 놀랐습니다. 한분야의 대가이신데도 ... 부럽습니다 저는 지금까지 뭘했는지 참 동기부여의 말도 가끔 올려주시면 참 잘 읽고 있습니다.

점점 더워질텐데 학생들 가리쳐주시느라 너무 고생이 많으신거 같습니다 ㅠㅠ

감사합니다 .

점심먹을때나 쉬는시간에 많은 대화를 나누고 싶습니다.