TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 – GJ (박현우)
uc820@naver.com

임베디드 애플리케이션 분석

1. 이것이 없으면 사실상 C 언어를 사용할 수 없다 .

   어셈블리 언어로 call 하는 것이 필요하다 .

임베디드 애플리케이션 분석

4. goto 의 정체성은 ?

예를 들어 , for 문을 중복해서 사용하고 if 와 break 로 for 문에서 빠져 나올 때 , 상당히 많은 불필요한 명령어들이 사용된다 .

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 이다 .

Call 이나 jpm 를 cpu instruction 레벨에서 분기 명령어라고 하고 이들은 cpu 파이프라인에 치명적인 손실을 가져다 준다 .

즉 , 성능면으로만 보아도 goto 가 월등히 좋다는 것을 알 수 있다 .

임베디드 애플리케이션 분석
5.

포인터 크기

컴퓨터의 bit 에 따라서 포인터의 크기가 달라진다 .
예를 들어 , 64bit 컴퓨터의 경우 포인터의 크기는 8byte 이고
32bit 컴퓨터의 경우 포인터의 크기는 4byte 이다 .

임베디드 애플리케이션 분석

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

임베디드 애플리케이션 분석

8.

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

Return : void (*)(int)
Name : signal
Parameter : int signum, void (* handler)(int)

임베디드 애플리케이션 분석

9.

```
#include<stdio.h>

// void (*)(void)
// return : void
// name : aaa
// parameter : void

void aaa(void){

	printf("aaa called\n");
}
//void (*)( void (*)(void))
//void bbb( void (*p)(void))
//return : void
//name : bbb
//parameter : void (*)(void)
void bbb(void (*p)(void)){

	p();

	printf("bbb called\n");
}

//void (*)(void(*p)(void)) test(int a,
int b)
//void (* test(int a,int b))(void(*p)
(void))
// return void (*)( void (*)(void))
// name : test
// parameter : int , int
void (* test(int a,int b))(void (*p)
(void)){

	int sum = a+b;
	printf("test sum = %d\n", sum);

	return bbb;
}

int main(void){

	int sum =0;

	test(4,6)(aaa);

	return 0;
}
```

결과

```
test sum = 10
aaa called
bbb called
```

임베디드 애플리케이션 분석

10.

파이프라인은 성능 향상을 위해서 사용하는데 , 파이프라인이 깨진다면 cpu 의 clock 낭비가 생긴다 .

하지만 , 다음사이클의 명령어를 실행하지 못하는 경우나 파이프라인의 속도가 느려지는 경우처럼 파이프라인의 위험성이 존재한다 .

파이프라인이 깨진다면 , 클락낭비는 있겠지만 위와 같은 위험은 피해갈 수 있는 이점이 있다 .

12

```
#include<stdio.h>

void add_matrix_type_float(float (*a)[3], float (*b)[3]){

        int i,j;
        static float e[3][3] = {0};

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

                for(j=0;j<3;j++){
                        e[i][j] = a[i][j] + b[i][j];
                        printf("%f ", e[i][j]);
                }

                printf("\n");
        }
}
```

```
void add_matrix_type_int(int (*a)
[3], int (*b)[3]){

        int i,j;
        static int c[3][3] = {0};

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

                for(j=0;j<3;j++){
                        c [i][j] = a[i][j] + b[i][j];
                        printf("%d ", c[i][j]);
                }

                printf("\n");
        }
}
```

12

```
int main(void){

	int a[3][3] = { {1,2,3}, {2,3,4}, {5,6,7}};
	int b[3][3] = { {1,2,3}, {2,3,4}, {5,6,7}};

	float c[3][3] = { {1,2,3}, {2,3,4}, {5,6,7}};
	float d[3][3] = { {1,2,3}, {2,3,4}, {5,6,7}};

	printf("int type\n");
	add_matrix_type_int(a,b);

	printf("float type\n");
	add_matrix_type_float(c,d);

	return 0;
}
```

결과

```
int type
2 4 6
4 6 8
10 12 14
float type
2.000000 4.000000 6.000000
4.000000 6.000000 8.000000
10.000000 12.000000
14.000000
```

임베디드 애플리케이션 분석

13.

```
#include<stdio.h>
void odd_sum(int *arr){

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

                if ( arr[i] % 2 == 1)
                        sum += arr[i];
        }

        printf("odd_sum = %d\n", sum);
}

void even_sum(int *arr){

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

                if ( arr[i] % 2 == 0)
                        sum += arr[i];
        }

        printf("even_sum = %d\n", sum);
}

int fib(int num){

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

                return fib(num - 1) + fib (num - 2);
        }
}

int main(void){

        int arr[28] = {0};
        int i;

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

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

        even_sum(arr);
        odd_sum(arr);

        return 0;
}
```

결과

```
even_sum = 257114
odd_sum = 257114
```

임베디드 애플리케이션 분석
14.

```
#include<stdio.h>

int dif_fib(int num){

        if( num == 1 )
                return 1;
        else if(num ==2)
                return 4;
        else
                return dif_fib( num - 1) + dif_fib( num - 2);
}


int main(void){

        int num;

        scanf("%d", &num);

        printf("result = %d\n", dif_fib(num));

        return 0;
}
```

결과
23
result = 81790

임베디드 애플리케이션 분석
15.

모든 프로세서는 레지스터에서 레지스터로 연산이 가능 x86 은 메모리에서 메로리로 연산이 가능하지만

ARM 은 로드 / 스토어 아키텍처라고해서 메모리에서 메모리 불가능 (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 작업을 함 .

로드하고 스토어하는 방식이라고 해서 로드 / 스토어 아키텍처라고 함 .

임베디드 애플리케이션 분석

16. 네이버의 쓰레기같은 사다리 게임을 우리끼리 즐길 수 있는 것으로 대체하는 프로그램을 만들어보자 .

내용

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.
```
#include<stdio.h>

void add(int (*arr)[3]){

        arr[1][0] *=arr[0][0];
        arr[1][1] *=arr[0][1];
        arr[1][2] *=arr[0][2];

}

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

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

        int i,j;
        for(i=0;i<2;i++){

                for(j=0;j<3;j++)
                        printf("%d ",arr[i][j]);

                printf("\n");
        }

}
```

```
int main(void){

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

        print_arr(arr);

        printf(" 함수적용 후 \n");
        sapply(arr,add);
        print_arr(arr);

}
```

결과

```
2 4 6
2 4 6
함수적용 후
2 4 6
4 16 36
```

임베디드 애플리케이션 분석
18.

```
phw@phw-Z20NH-AS51B5U:~/test$ vi 18.c
phw@phw-Z20NH-AS51B5U:~/test$ cat 18.c
#include<stdio.h>

int find_capital(char *arr){

        char alp[] = {'a','b','c','d','e','f','g','h',
                'i','j','k','l','m','n','o','p','q','r','s','t','u',
                'v','w','x','y','z'};

        int count = 0;
        int i,j;

        for(i =0; arr[i];i++){

                for(j=0; j<26; j++){

                                if( arr[i] == alp[j] && arr[i] != ' '){
                                                count++;

                                                printf("%c ",arr[i]);
                        }

                }
        }

        return count;
}
int main(void){

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

        int result = 0;

        result = find_capital(str);

        printf("result = %d\n",result);
        return 0;
}
```

```
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
h e r e i s m y o i n t e r h e r e i s i t result = 22
phw@phw-Z20NH-AS51B5U:~/test$
```

임베디드 애플리케이션 분석
19.

int *p[3] 과 int (*p)[3] 은 다른 것이다 .

int *p[3] 은 포인터 배열이고

int (*p)[3] 은 배열 포인터이다 .

즉 int (*p)[3] 은 달리 쓰면 , int (*)[3] p 라고 쓸 수 있다 . 즉 int 형 포인터 3 개짜리를 받을 수 있는 변수 p 라는 뜻이다 .

임베디드 애플리케이션 분석
20.

```
phw@phw-Z20NH-AS51B5U: ~/test
phw@phw-Z20NH-AS51B5U:~/test$ vi 20.c
phw@phw-Z20NH-AS51B5U:~/test$ gcc 20.c
phw@phw-Z20NH-AS51B5U:~/test$ cat 20.c
#include<stdio.h>


int main(void){

        long num;

        scanf("%lu",&num);

        printf("%lu\n", num & ~134217727);

        return 0;
}
phw@phw-Z20NH-AS51B5U:~/test$
```

```
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
1111111111111
1111054352384
phw@phw-Z20NH-AS51B5U:~/test$
```

임베디드 애플리케이션 분석
21.

phw@phw-Z20NH-AS51B5U: ~/test

```
1 #include<stdio.h>
2
3 char changer(char alpha){
4
5     return alpha ^= 0x20;
6 }
7
8 int main(void){
9
10     char cap;
11
12     scanf("%c", &cap);
13
14     cap = changer(cap);
15
16     printf("%c\n", cap);
17 }
```

phw@phw-Z20NH-AS51B5U: ~/test

```
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
A
a
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
a
A
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
b
B
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
C
c
phw@phw-Z20NH-AS51B5U:~/test$
```

임베디드 애플리케이션 분석
22.

변수란 값을 저장할 수 있는 메모리 공간이다 .

임베디드 애플리케이션 분석
23.

포인터란 주소 값 저장할 수 있는 메모리 공간이다 .

임베디드 애플리케이션 분석
24.

함수 포인터란 함수의 주소를 저장할 수 있는 메모리 공간이다 .

임베디드 애플리케이션 분석
25.

파이프라인은 call 이나 jmp 등 분기 명령어가 너무 자주 호출될 때 , cpu 의 파이프라인에 치명적인 손실을 가져다 준다
이러한 경우 cpu clock 의 낭비로 인해 파이프라인이 깨질 수 있다 .

메모리 계층구조란 메모리를 필요에 따라 여러가지 종류로 나누어 둠을 의미한다 . 이때 필요란 대부분의 경우 CPU 가 메모리에 더 빨리 접근하기 위함이다 .

속도는 레지스터 > 캐시 > 메모리 > 하드디스크 순이다 .

레지스터와 캐시는 cpu 내부에 존재하고 메모리는 cpu 외부에 존재한다 . 그렇기 때문에 레지스터와 캐시보다는 느리다 . 반면 , 하드 디스크는 cpu 가 직접 접근할 방법이 없다 . cpu 가 하드웨어에 접근하기 위해서는 하드디스크의 데이터를 메모리로 이동시키고 , 메모리에서 접근해야하기 때문에 접근 속도가 아주 느리다 .

임베디드 애플리케이션 분석
27.

stack 은 지역 변수와 함수 호출시 변수가 저장되는 공간이며 stack 은 다른 영역과 달리 위에서 아래로 쌓이는 구조이다 .

data 는 전역변수 , 정적 변수가 저장되는 공간이다

text 는 프로그램의 코드 부분이 저장되는 영역으로 순차처리를 한다 .

heap 은 프로그래머가 동적 할당을 해서 생성한 변수가 저장되는 공간이다 .

임베디드 애플리케이션 분석
28.

디버깅시에 보는 주소는 가짜주소이다 . 메모리의 실제주소를 사용자가 건들면 안되기 때문에 물리적 주소가 아닌 가상주소를 보여준다 .

프로그램을 하다가 세그멘테이션 에러가 나는 이유는 사용자가 건들면 안되는 공간을 건들었기 때문이다 .

이처럼 디버깅시에도 실제주소가 아닌 가상 주소를 보여준다 .

임베디드 애플리케이션 분석
29.

phw@phw-Z20NH-AS51B5U: ~/test

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

typedef struct __person{

    char name[100];
    int salary;
}person;

void avg_salary(person *p, int num){

    int i, sum;
    double avg;

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

        sum += p[i].salary;
    }


    printf("salary avg = %lf\n",(double)(sum / num));
}

void who_is_best(person *p,int num){

    int tmp = 0;
    person best;
    int i, k;

    for(i=0; i<num;i++){
        if(p[i].salary > tmp){
            tmp = p[i].salary;
            k = i;
        }
    }

    printf("best person name: %s, salary: %d\n", p[k].name, p[k].salary);
}

```

```c

person *insert_person(person *p, int num){

    int i;
    char *name;

    p = (person *)malloc(sizeof(person) * num);

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

        scanf("%s", p[i].name);
        p[i].salary = rand();

        printf("name = %s\n", p[i].name);
        printf("salary = %d\n",p[i].salary);
    }

    return p;
}

int main(void){

    int number_of_people;

    person *p;

    scanf("%d",&number_of_people);

    p = insert_person(p,number_of_people);

    avg_salary(p,number_of_people);

    who_is_best(p, number_of_people);

    return 0;
}
```

76,1 Bot

```
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
3
aa
name = aa
salary = 1804289383
bb
name = bb
salary = 846930886
cc
name = cc
salary = 1681692777
salary avg = 12648583.000000
best person name: aa, salary: 1804289383
```

임베디드 애플리케이션 분석
30.

gcc -g -o0 -o debug aaa.c

임베디드 애플리케이션 분석
31.

v 를 누르고 코드를 블락을 만든 후 = 을 누르면 정렬된다 .

임베디드 애플리케이션 분석
31.

v 를 누르고 코드를 블락을 만든 후 = 을 누르면 정렬된다 .

임베디드 애플리케이션 분석
32.

gcc -o1 ~ o3 으로 최적화 옵션을 줄 수 있고 -o0 을 하면 아무런 옵션도 안준다 .

gdb 는 프로그램의 에러를 잡기 위해서 사용된다 .
또한 , 어셈블리어의 과정을 보기 위해서도 사용함 .

임베디드 애플리케이션 분석
34.

push pop jump ret

임베디드 애플리케이션 분석
35.

phw@phw-Z20NH-AS51B5U: ~/test

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
// (a,b) (c,d),(e,f)

void tri_area(int *a, int *b, int *c){

    double area;
    area =  ((a[0]*b[1] + b[0]*c[1] + c[0]*a[1]) - ( a[1]*b[0] + b[1]*c[0] + c[1]*a[0])) / 2;

    printf("%d\n",abs(area));
}

int main(void){

    int a[2] = {0};
    int b[2] = {3, 6};
    int c[2] = {4, 4};

    tri_area(a,b,c);

    return 0;
}
```

```
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
6
phw@phw-Z20NH-AS51B5U:~/test$
```

임베디드 애플리케이션 분석
36.

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

typedef enum _game{

    ROCK = 1,
    SCC,
    PAPER

}GAME;

void game(void){

    int com = rand()%3 +1;
    int num;

again:

    printf("가위 : 1, 바위 : 2, 보 : 3를 입력하세요 : ");

    scanf("%d", &num);

    switch(num){

        case ROCK:
                num = 0;
                if(com == 1){
                    printf("컴퓨터와 비겼습니다\n");
                    goto again;

                }else if(com ==2)
                    printf("플레이어가 이겼습니다\n");
                else
                    printf("컴퓨터가 이겼습니다\n");


                break;
        case SCC:
                num = 0;
                if(com == 2){
                    printf("컴퓨터와 비겼습니다\n");
                    goto again;
                }else if(com ==3)
                    printf("플레이어가 이겼습니다\n");
                else
                    printf("컴퓨터가 이겼습니다\n");
                break;

        case PAPER:
                num = 0;
                if(com == 3){
                    printf("컴퓨터와 비겼습니다\n");
                    goto again;
                }else if(com ==1)
                    printf("플레이어가 이겼습니다\n");
                else
                    printf("컴퓨터가 이겼습니다\n");
                break;
```

```c

        default :
                printf("숫자를 다시 입력해주세요\n");
                num = 0;
                    goto again;

    }

}

int main(void){

    int num = 0;

    game();
    return 0;
}
```

```
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
가위 : 1, 바위 : 2, 보 : 3를 입력하세요 : 1
플레이어가 이겼습니다
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
가위 : 1, 바위 : 2, 보 : 3를 입력하세요 : 2
컴퓨터와 비겼습니다
가위 : 1, 바위 : 2, 보 : 3를 입력하세요 : 3
컴퓨터가 이겼습니다
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
가위 : 1, 바위 : 2, 보 : 3를 입력하세요 : 1
플레이어가 이겼습니다
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
가위 : 1, 바위 : 2, 보 : 3를 입력하세요 : 3
컴퓨터가 이겼습니다
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
가위 : 1, 바위 : 2, 보 : 3를 입력하세요 : 2
컴퓨터와 비겼습니다
가위 : 1, 바위 : 2, 보 : 3를 입력하세요 : 4
숫자를 다시 입력해주세요
가위 : 1, 바위 : 2, 보 : 3를 입력하세요 : 3
컴퓨터가 이겼습니다
phw@phw-Z20NH-AS51B5U:~/test$
```

임베디드 애플리케이션 분석
40.

수업을 들으면서 c 언어라는 것에 대해서 오개념이 너무 많았고 대학교 수업이 너무나도 수박 겉핥기 공부라는 생각을 했다 .

지금 한달동안 c 언어에 대한 깊이 있는 공부를 할 수 있어서 너무나 만족한다 . 또한 , 자료구조를 공부하며 다른 코드를 보지 않고 내가 스택부터 트리까지 구현을 할 수 있게 된 점도 너무나도 좋다 .

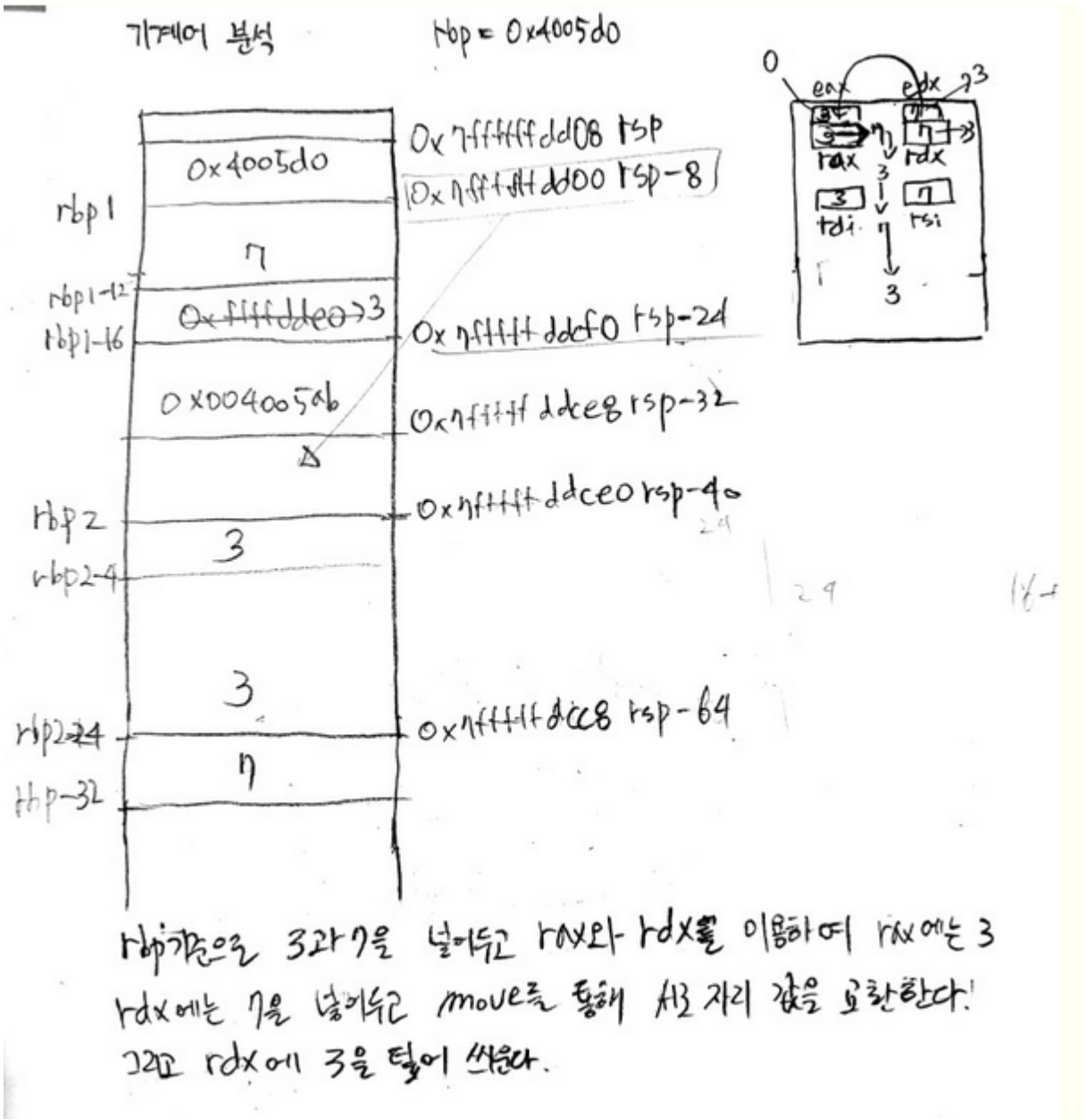
앞으로는 더욱더 어려워질 것이라고 생각한다 . 하지만 , 깊이있는 배움을 생각하면 앞으로 더 만족하고 힘들어도 재밌을 거라고 확신한다 .

앞으로 남은 기간에는 드론을 통한 레이저 타격과 같은 프로젝트를 진행하고 싶다 . 그리고 방산업체나 대기업 위주로 취업준비를 할 생각이다 .
또한 , 정처기 실기가 끝나면 , 틈틈히 영어회화 준비를 하면서 돈을 모아 외국 대학원을 진학해보고 싶다 .

그리고 최종적으로는 내가 만든 제품을 전세계 사람이 사용하는 개발자가 되고 싶다 .

임베디드 애플리케이션 분석

41.



자료구조

1.

phw@phw-Z20NH-AS51B5U: ~/test

```c
#include<stdio.h>
#include<math.h>
#include<stdlib.h>

void input_random(int *arr){

    int i;

    for(i=0;i<100;i++){

        arr[i] = rand()%4096 +1;
        printf("%d ",arr[i]);
    }

    printf("\n");

}

int main(void){

    int arr[100];

    input_random(arr);

}
```

```
phw@phw-Z20NH-AS51B5U:~/test$ ./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 2
62 792 2137 2794 1119 2261 3500 3251 206 199 1692 3765 1109 1554 1039 3459 2165 1602 1314 3390 3549
 136 2673 3562 2111 1186 578 2274 3581 872 575 3842 1663 2712 2539 2781 876 1943 1936 1081 2141 362
7 749 3249 1084 1788 2611 3248 3389 3925 2541 2841 4060 1117 2307 2075 2303 2884
```

자료구조

4.

phw@phw-Z20NH-AS51B5U: ~/test

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

typedef struct __tree
{
    int data;
    struct __tree *left;
    struct __tree *right;
} tree;

tree *get_node(void)
{
    tree *tmp;
    tmp = (tree *)malloc(sizeof(tree));
    tmp->left = NULL;
    tmp->right = NULL;
    return tmp;
}

void tree_ins(tree **root, int data)
{
    if(*root == NULL)
    {
        *root = get_node();
        (*root)->data = data;
        return;
    }
    else if((*root)->data > data)
        tree_ins(&(*root)->left, data);
    else if((*root)->data < data)
        tree_ins(&(*root)->right, data);
}


void print_tree(tree *root)
{
    if(root)
    {
        printf("data = %d, ", root->data);
        print_tree(root->left);
        print_tree(root->right);
    }
}



tree *chg_node(tree *root)
{
    tree *tmp = root;

    if(!root->right)
        root = root->left;
    else if(!root->left)
        root = root->right;

    free(tmp);

    return root;
}

tree *find_max(tree *root, int *data)
{
    if(root->right)
        root->right = find_max(root->right, data);
    else
    {
        *data = root->data;
        root = chg_node(root);
    }

    return root;
}

tree *delete_tree(tree *root, int data)
{
    int num;
    tree *tmp;
    if(root == NULL)
    {
        printf("Not Found\n");
        return NULL;
    }
    else if(root->data > data)
        root->left = delete_tree(root->left, data);
    else if(root->data < data)
        root->right = delete_tree(root->right, data);
    else if(root->left && root->right)
    {
        root->left = find_max(root->left, &num);
        root->data = num;
    }
    else
        root = chg_node(root);
    return root;
}

int main(void)
{
    int i;
    int data[14] = {50, 45, 73, 32, 48, 46, 16,
        37, 120, 47, 130, 127, 124};

    tree *root = NULL;

    for(i = 0; data[i]; i++)
        tree_ins(&root, data[i]);

    print_tree(root);

    delete_tree(root, 50);
    printf("After Delete\n");

    print_tree(root);

    return 0;
}
```

```
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
data = 50,
data = 45,
data = 32,
data = 16,
data = 37,
data = 48,
data = 46,
data = 47,
data = 73,
data = 120,
data = 130,
data = 127,
data = 124,
After Delete
data = 48,
data = 45,
data = 32,
data = 16,
data = 37,
data = 46,
data = 47,
data = 73,
data = 120,
data = 130,
data = 127,
data = 124,
phw@phw-Z20NH-AS51B5U:~/test$
```

자료구조

5.

phw@phw-Z20NH-AS51B5U: ~/test

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct __tree
{
    int data;
    struct __tree *left;
    struct __tree *right;
} tree;

typedef struct __stack
{
    void *data;
    struct __stack *link;
} stack;

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

tree *get_tree_node(void)
{
    tree *tmp;
    tmp = (tree *)malloc(sizeof(tree));
    tmp->left = NULL;
    tmp->right = NULL;
    return tmp;
}

void *pop(stack **top)
{
    stack *tmp = *top;
    void *data = NULL;

    if(*top == NULL)
    {
        printf("stack is empty!\n");
        return NULL;
    }

    data = (*top)->data;
    *top = (*top)->link;
    free(tmp);

    return data;
}
```

```c

void push(stack **top, void *data)
{
    if(data == NULL)
        return;

    stack *tmp = *top;
    *top = get_stack_node();
    (*top)->data = malloc(sizeof(void *));
    (*top)->data = data;
    (*top)->link = tmp;
}

void non_recur_tree_ins(tree **root, int data)
{
    tree **tmp = root;

    while(*tmp)
    {
        if((*tmp)->data > data)
            tmp = &(*tmp)->left;
        else if((*tmp)->data < data)
            tmp = &(*tmp)->right;
    }

    *tmp = get_tree_node();
    (*tmp)->data = data;
}

bool stack_is_not_empty(stack *top)
{
    if(top != NULL)
        return true;
    else
        return false;
}
```

```c

bool stack_is_not_empty(stack *top)
{
    if(top != NULL)
        return true;
    else
        return false;
}

void print_tree(tree **root)
{
    tree **tmp = root;
    stack *top = NULL;

    push(&top, *tmp);

    while(stack_is_not_empty(top))
    {
        tree *t = (tree *)pop(&top);
        tmp = &t;

        printf("data = %d, ", (*tmp)->data);

        if((*tmp)->left)
            printf("left = %d, ", (*tmp)->left->data);
        else
            printf("left = NULL, ");

        if((*tmp)->right)
            printf("right = %d\n", (*tmp)->right->data);
        else
            printf("right = NULL\n");

        push(&top, (*tmp)->right);
        push(&top, (*tmp)->left);

    }
}
```

자료구조

5.

```
120 tree *chg_node(tree *root)
121 {
122     tree *tmp = root;
123
124     if(!root->right)
125         root = root->left;
126     else if(!root->left)
127         root = root->right;
128
129     free(tmp);
130
131     return root;
132 }
133
134 void find_max(tree **root, int *data)
135 {
136     tree **tmp = root;
137
138     while(*tmp)
139     {
140         if((*tmp)->right)
141             tmp = &(*tmp)->right;
142         else
143         {
144             *data = (*tmp)->data;
145             *tmp = chg_node(*tmp);
146             break;
147         }
148     }
149 }
```

```
151 void non_recur_delete_tree(tree **root, int data)
152 {
153     tree **tmp = root;
154     int num;
155
156     while(*tmp)
157     {
158         if((*tmp)->data > data)
159             tmp = &(*tmp)->left;
160         else if((*tmp)->data < data)
161             tmp = &(*tmp)->right;
162         else if((*tmp)->left && (*tmp)->right)
163         {
164             find_max(&(*tmp)->left, &num);
165             (*tmp)->data = num;
166             return;
167         }
168         else
169         {
170             (*tmp) = chg_node(*tmp);
171             return;
172         }
173     }
174
175     printf("Not Found\n");
176 }
177
178 int main(void)
179 {
180     int i;
181     int data[14] = {50, 45, 73, 32, 48, 46, 16,
182         37, 120, 47, 130, 127, 124};
183
184     tree *root = NULL;
185
186     for(i = 0; data[i]; i++)
187         non_recur_tree_ins(&root, data[i]);
188
189     print_tree(&root);
190
191     non_recur_delete_tree(&root, 50);
192     printf("After Delete\n");
193
194     print_tree(&root);
195
196     return 0;
197 }
```

```
phw@phw-Z20NH-AS51B5U:~/test$ ./a.out
data = 50, left = 45, right = 73
data = 45, left = 32, right = 48
data = 32, left = 16, right = 37
data = 16, left = NULL, right = NULL
data = 37, left = NULL, right = NULL
data = 48, left = 46, right = NULL
data = 46, left = NULL, right = 47
data = 47, left = NULL, right = NULL
data = 73, left = NULL, right = 120
data = 120, left = NULL, right = 130
data = 130, left = 127, right = NULL
data = 127, left = 124, right = NULL
data = 124, left = NULL, right = NULL
After Delete
data = 48, left = 45, right = 73
data = 45, left = 32, right = 46
data = 32, left = 16, right = 37
data = 16, left = NULL, right = NULL
data = 37, left = NULL, right = NULL
data = 46, left = NULL, right = 47
data = 47, left = NULL, right = NULL
data = 73, left = NULL, right = 120
data = 120, left = NULL, right = 130
data = 130, left = 127, right = NULL
data = 127, left = 124, right = NULL
data = 124, left = NULL, right = NULL
phw@phw-Z20NH-AS51B5U:~/test$
```

자료구조
7.

AVL 과 RB TREE

AVL 은 검색은 빠르나 삭제가 많아지고 회전이 많아지면 속도가 점점 느려지고 데이터 손실이 발생할 수 있다 .

반면 , Rb 는 검색속도는 약간 희생했지만 , 삭제와 입력 둘다 빠르다 . 즉 , 빈번한 데이터 삽입과 삭제에 효율적이다 .

그래서 RB 는 대규모 서버 관리나 검색용으로 많이 사용된다 .

자료구조

8.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.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;
}

void enqueue(queue **head, int data)
{
    if(*head == NULL)
    {
        *head = get_node();
        (*head)->data = data;
        return;
    }

    enqueue(&(*head)->link, data);
}

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

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

```c
queue *dequeue(queue *head, int data)
{
    queue *tmp = head;

    if(tmp == NULL)
        printf("There are no data that you delete\n");

    if(head->data != data)
        head->link = dequeue(head->link, data);
    else
    {
        //queue *res = head->link;
        printf("Now you delete %d\n", data);
        free(tmp);
        return head->link;
    }

    return head;
}
bool is_dup(int *arr, int cur_idx)
{
        int i, tmp = arr[cur_idx];

            for(i = 0; i < cur_idx; i++)
                    if(tmp == arr[i])
                                return true;

                return false;
}
```

```c
void init_rand_arr(int *arr, int size)
{
    int i;

    for(i = 0; i < size; i++)
    {
redo:
        arr[i] = rand() % 16;

        if(is_dup(arr, i))
        {
            printf("%d dup! redo rand()\n", arr[i]);
            goto redo;
        }
    }
}


int main(void)
{
    int i;

    queue *head = NULL;


    int arr[16] = {0};
    init_rand_arr(arr,16);
    for(i = 0; i < 3; i++)
        enqueue(&head, arr[i]);

    print_queue(head);


    return 0;
```

```
3 dup! redo rand()
6 dup! redo rand()
12 dup! redo rand()
2 dup! redo rand()
11 dup! redo rand()
8 dup! redo rand()
7 dup! redo rand()
13 dup! redo rand()
6 dup! redo rand()
10 dup! redo rand()
3 dup! redo rand()
3 dup! redo rand()
15 dup! redo rand()
9 dup! redo rand()
10 dup! redo rand()
6 dup! redo rand()
2 dup! redo rand()
13 dup! redo rand()
7 dup! redo rand()
1 dup! redo rand()
8 dup! redo rand()
3 dup! redo rand()
10 dup! redo rand()
13 dup! redo rand()
5 dup! redo rand()
7 dup! redo rand()
8 dup! redo rand()
9 dup! redo rand()
14 dup! redo rand()
4 dup! redo rand()
11 dup! redo rand()
2 dup! redo rand()
13 dup! redo rand()
6 dup! redo rand()
11 dup! redo rand()
4 dup! redo rand()
4 dup! redo rand()
1 dup! redo rand()
14 dup! redo rand()
2 dup! redo rand()
4 dup! redo rand()
1 dup! redo rand()
1 dup! redo rand()
13 dup! redo rand()
12 dup! redo rand()
7 dup! redo rand()
head->data = 7
head->data = 6
head->data = 9
phw@phw-Z20NH-AS51B5U:~/test$
```

자료구조
10.

```c
#include<stdio.h>
#include<malloc.h>
#include<math.h>
#include<stdbool.h>
#include<stdlib.h>

#define EMPTY 0

struct node{

    int data;
    struct node *link;
};


typedef struct node Stack;

Stack *get_node(){

    Stack *tmp;
    tmp = (Stack *)malloc(sizeof(Stack));
    tmp->link=EMPTY;
    return tmp;
}

void push(Stack **top, int data){

    Stack *tmp;
    tmp = *top;
    *top = get_node();
    (*top)->data = data;
    (*top)->link = tmp;
}
```

```c
int pop(Stack **top){

    Stack *tmp;
    int num;
    tmp = *top;
    if(*top == EMPTY){

        printf("Stack is empty!!\n");
        return 0;
    }
    num = tmp->data;
    *top = (*top)->link;
    free(tmp);
    return num;
}

bool is_dup(int *arr, int cur_idx)
{
    int i, tmp = arr[cur_idx];

    for(i = 0; i < cur_idx; i++)
        if(tmp == arr[i])
            return true;

    return false;
}
```

```c
void init_rand_arr(int *arr, int size)
{
    int i;

    for(i = 0; i < size; i++)
    {
redo:
        arr[i] = rand() % 100 + 1;

        if(is_dup(arr, i))
        {
            printf("%d dup! redo rand()\n", arr[i]);
            goto redo;
        }
    }
}
int main(void){

    Stack *top = EMPTY;
    int arr[100];
    int i;
    init_rand_arr(arr, 100);
    for(i = 0;i<20;i++){
        push(&top, arr[i]);
    }

    for(i = 0;i<20;i++){
        printf("%d\n",pop(&top));
    }

    return 0;

}
```

```
61 dup! redo rand()
15 dup! redo rand()
22 dup! redo rand()
61 dup! redo rand()
5 dup! redo rand()
29 dup! redo rand()
28 dup! redo rand()
51 dup! redo rand()
49 dup! redo rand()
57 dup! redo rand()
3 dup! redo rand()
95 dup! redo rand()
98 dup! redo rand()
100 dup! redo rand()
44 dup! redo rand()
40 dup! redo rand()
3 dup! redo rand()
29 dup! redo rand()
4 dup! redo rand()
1 dup! redo rand()
82 dup! redo rand()
69
12
37
73
41
27
64
60
91
28
63
22
50
93
36
94
16
78
87
84
phw@phw-Z20NH-AS51B5U:~/practice$
```