TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – GJ (박현우)
uc820@naver.com

1. 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 ?

정답

```
1  #if NO1
2  스택
3  #endif
```

내가 쓴 답안

임베디드 애플리케이션 분석

1. 이것이 없으면 사실상 C 언어를 사용할 수 없다 .

어셈블리 언어로 call 하는 것이 필요하다 .

오답노트

스택이 물론 필요한 걸 알았지만, 함수 호출할 때 기계어로 call을 하기 때문에 call이라고 생각했다...

1. 임베디드 애플리케이션 분석

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)

```
void sort(hash *arr, int len)
{
	int i, j, key1, key2;

	for (i = 1; i < len - 1; i++)
	{
		key1 = arr[i].freq;
		key2 = arr[i].cnt;

		for (j = i - 1; arr[j].freq > key2 && j > -1; j--)
		{
			arr[j + 1].freq = arr[j].freq;
			arr[j + 1].cnt = arr[j].cnt;
		}

		arr[j + 1].freq = key1;
		arr[j + 1].cnt = key2;
	}
}
```

오답노트

시간도 부족하고, 여러 측면에서 풀지 못함.

정답을 보고 숫자와 빈도수를 저장할 구조체 배열과

입력된 숫자와 기존 숫자를 비교할 함수가 필요하다는

점을 알게 되었다. 하지만 위에 sort함수는 key2만 비교하는 것이 아니고 freq를 비교하는지 아직 이해가 되지 않는다.

1. 임베디드 애플리케이션 분석

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.

정답

```c
#include <stdio.h>

#define RESOLUTION      1 << 12
#define MINUS_VOLT      -5
#define PLUS_VOLT       5

float get_slice(void)
{
        //return (float)(PLUS_VOLT - MINUS_VOLT) / (1 << 16);
        return (float)(PLUS_VOLT - MINUS_VOLT) / (RESOLUTION);
}

float adc(int bit, float slice)
{
                return (float)(MINUS_VOLT) + bit * slice;
}

int main(void)
{
                float slice, volt;
                int bit = 2077;

                slice = get_slice();
                printf("slice = %f\n", slice);

                volt = adc(bit, slice);
                printf("volt = %f\n", volt);

                return 0;
}
```

오답노트

아직 디지털 관점과 보드에 관한 지식이 부족하여

어떻게 코딩해야 하는 것인지 감조차 못 잡았습니다.

공부 후 다시 풀어보겠습니다.

1. 임베디드 애플리케이션 분석

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

정답

```
#if NO4
goto 는 리눅스 커널에서 밥먹듯이 사용하는 매우 좋은 녀석이다.
왜 좋냐하면 에러가 발생했을 경우
flag 변수를 사용하고 if 문에 break 를 사용하면
다중 루프 구조에서 if 문에 break 를 여러번 사용해야 한다.
또한 메모리 관점에서 쓸대 없는 변수 선언이 한 번 있게 된다.
if 문은 branch 에 해당하며 break 또한 branch 에 해당한다.
branch 가 발생하면 CPU 의 파이프라인이 깨지기 때문에
성능이 하락하게 되는데 goto 는 단 한 번의 jmp(branch) 만으로
위의 작업을 수행하지 않고 작업을 수행할 수 있기 때문에
성능상 이점 메모리 공간 효율성상의 이점으로 goto 를 사용하는 것이다.
#endif
```

내가 쓴 답안

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 .

오답노트

이 문제는 그래도 정답을 맞춘 것 같다.

그래도 if와 break등이 branc라는 점과 branch가 발생할 때 마다 cpu의 파이프라인이 깨진다는 점은 추가해야겠다.

1. 임베디드 애플리케이션 분석

5. 포인터 크기에 대해 알고 있는대로 기술하시오.

정답

```
#if NO5
32 비트 및 64 비트 컴퓨터의 비트 수에 따라서
4 byte 혹은 8 byte 로 결정됨
4 byte 는 32 비트로 만약 바이트 단위로 카운팅을 한다면
2^32 byte 가 되고 이것은 32 비트 운영체제의
가상메모리 주소인 4 GB 의 공간을 의미하게 된다.
2^64 byte 로 계산하면 이것이 64 비트 운영체제의 가상 메모리 공간이다.
즉 낮은 주소와 높은 주소를 모두 표현하기 위해
최대 사이즈를 커버하는 구조로 포인터의 크기를 결정하게 되는 것이다.
#endif
```

내가 쓴 답안

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 .

오답노트

5번 문제의 경우 단순히 4byte이냐 8byte냐만 쓰고 더 정확한 이유에 대해서 쓰지 못한 것 같다.

낮은 주소와 높은 주소를 모두 커버하기 위해 최대사이즈로 결정한다는 점을 보완해야겠다.

1. 임베디드 애플리케이션 분석

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)

정답

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

int where_is_it(char *text, char *find)
{
        int i;

        for(i = 0; text[i]; i++)
                if(!(strncmp(&text[i], find, strlen(find))))
                        return i;
}

int main(void)
{
        int idx;
        char text[100] = "TI Cortex-R5F Safety MCU is very good to Real-Time System.";

        idx = where_is_it(text, "Safety MCU");
        printf("idx = %d\n", idx);

        return 0;
}
```

내가 쓴 답안

오답노트

이 문제는 stncmp를 쓸 생각을 못했다.

함수를 사용하지 않고 풀어야 되는지 알고

그냥 pass를 했는데... 풀걸 그랬다...는 아쉬움이 남는다.

1. 임베디드 애플리케이션 분석

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

내가 쓴 답안

정답

```
void mult_mat(int (*A)[3], int (*B)[3], int (*R)[3])
{
        int a = A[0][0], b = A[0][1], c = A[0][2],
                d = A[1][0], e = A[1][1], f = A[1][2],
                g = A[2][0], h = A[2][1], i = A[2][2],
                j = B[0][0], k = B[0][1], l = B[0][2],
                m = B[1][0], n = B[1][1], o = B[1][2],
                p = B[2][0], q = B[2][1], r = B[2][2];

        R[0][0] = a * j + b * m + c * p;
        R[0][1] = a * k + b * n + c * q;
        R[0][2] = a * l + b * o + c * r;
        R[1][0] = d * j + e * m + f * p;
        R[1][1] = d * k + e * n + f * q;
        R[1][2] = d * l + e * o + f * r;
        R[2][0] = g * j + h * m + i * p;
        R[2][1] = g * k + h * n + i * q;
        R[2][2] = g * l + h * o + i * r;
}
```

오답노트

이 문제 또한, 다른 문제들 해결하고 풀려고 냅뒀는데,

깜빡하고 못 풀었다...

정답을 보니, 답안처럼 a,b,c 등으로 배열마다

매칭을 시켜 푸는 방법은 정말 좋은 것 같다.

1. 임베디드 애플리케이션 분석

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)

정답

```c
#include <stdio.h>

// int (*)(int)
int t1(int a)
{
        printf("t1 function\n");
        return a * 3;
}

// int (*)(int) tp1(int (*p)(int))
// int (* tp1(int (*p)(int)))(int)
int (* tp1(int (*p)(int)))(int)
{
        printf("tp1 function\n");
        p(3);
        return p;
}

// int (*)(int) (*)(int (*p)(int)) tpp1 (int a, int b)
// int (*)(int) (* tpp1 (int a, int b))(int (*p)(int))
// int (* (* tpp1 (int a, int b))(int (*p)(int)))(int)
int (* (* tpp1(int a, int b))(int (*p)(int)))(int)
{
        printf("tpp1 function a = %d, b = %d\n", a, b);
        return tp1;
}

int main(void)
{
        int num;
        num = tpp1(3, 7)(t1)(33);
        printf("num = %d\n", num);
        return 0;
}
```

내가 쓴 답안

임베디드 애플리케이션 분석

```c
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");
}
```

```c
//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
```

오답노트

아... 이건 조금 잘못 생각한 것 같다.

함수 포인터를 반환하고 함수 포인터를 인자로 취하는 함수의 주소를 반환인데,

말뜻의 이해를 제대로 못했다...

1. 임베디드 애플리케이션 분석

10. 파이프라인이 깨지는 경우 어떠한 이점이 있는지 기술하시오.

정답

```
#if NO10
파이프라인이 깨지는 경우는 좋은 경우가 아니다.
branch 명령어를 만날 경우 파이프라인이 깨지게 되고
파이프라인에 쌓아왔던 정보들이 무의미해져
클록을 쓸모없이 소비한 케이스가 되어버린다.
#endif
```

내가 쓴 답안

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 .

오답노트

파이프라인이 깨지면 무조건 나쁘다는 생각을 하고

이점이 그나마 무엇일까 생각을 해봤는데

파이프라인 해저드라는 게 있길래 써봤다.

파이프라인이 깨지면 좋은 경우는 없다는 것에 대해

다시 한번 생각해봐야겠다.

1. 임베디드 애플리케이션 분석

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

정답

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

int calc_piece(float interval, float dx)
{
        return interval / dx;
}
float calc_int_4x_2_5x_1(float start, float end)
{
        int i, int_s, int_e;
        float sum = 0.0f;
        float temp = 0.0f;
        const float exp = 2.0;
      const float dx = 0.000002504f;

      float dy = pow(dx, exp);

      printf("dx = %.3f\n", dx);
      printf("dy = %.6f\n", dy);

      int_s = 0.0f;
      int_e = calc_piece(end - start, dx);

      printf("int_e = %d\n", int_e);

      for (i = int_s; i < int_e; i++)
      {
            temp += dx;
            dy = 4 * pow(temp + start, exp) + 5 * (temp + start) + 1;
            sum += dx * dy;
      }

      printf("sum = %f\n", sum);

      return sum;
}

int main(void)
{
      float res;
      res = calc_int_4x_2_5x_1(1.0f, 3.0f);
      printf("res = %f\n", res);
      return 0;
}
```

내가 쓴 답안

오답노트

정적분이야 할 줄 알지만, 코딩으로 하는 법은

처음 봤다. 좀 생각을 해보고 다시 풀어봐야겠다.

1. 임베디드 애플리케이션 분석

24. Intel Architecture와 ARM Architecture의 차이점은 ?

정답

```
#if NO15
Intel 은 CISC 구조로 반도체 다이 사이즈가 커서
다양한 Functional Unit 들이 많이 올라갈 수 있고
그로 인하여 전력 소모가 커질 수 밖에 없다.
ARM 은 RISC 구조로 다이 사이즈가 작아서
CISC 만큼 많은 양의 Functional Unit 이 올라가지 못하지만
전력 소모가 적다는 장점과 Load/Store 아키텍처로 구성된다.
#endif
```

내가 쓴 답안

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 .

오답노트

CISC 구조와 RISC 구조에 대한 차이를 더 명확하게 이해하지 못한 것 같다.

그리고 Intel에 대한 건 쓰지 않은 점이 부족했다.

1. 임베디드 애플리케이션 분석

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.

정답

Homework / sanghoonlee / answer / c / 16.c

내가 쓴 답안

오답노트

정답을 봤지만, 이해가 더 필요하다...

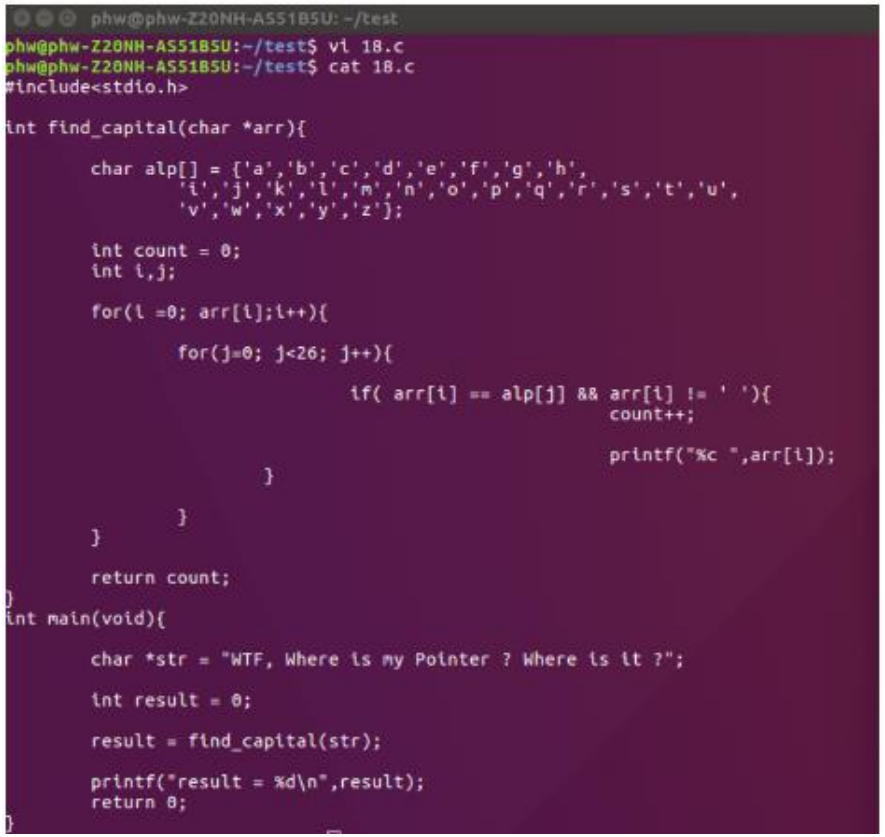
1. 임베디드 애플리케이션 분석

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

정답

```c
#include <stdio.h>

int check_lower(char *str)
{
        int i = 0, cnt = 0;

        for(; str[i]; i++)
                if(str[i] >= 97 && str[i] < 123)
                        cnt++;

        return cnt;
}

int main(void)
{
        int cnt;
        char *str = "WTF, Where is my Important Pointer ?";
        cnt = check_lower(str);
        printf("cnt = %d\n", cnt);
        return 0;
}
```

내가 쓴 답안



오답노트

와... 정답을 보니 아스키코드 써서 그냥 비교할 껄...
정말 무식하게 코딩한 것 같다...아휴...

1. 임베디드 애플리케이션 분석

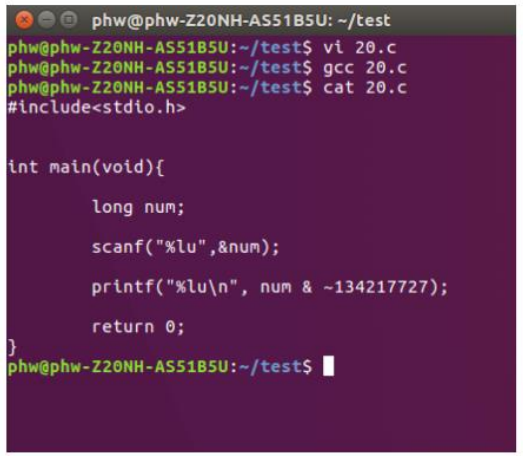
29. 임의의 값 x가 있는데, 이를 134217728 단위로 정렬하고 싶다면 어떻게 해야할까 ?
어떤 숫자를 넣던 134217728 의 배수로 정렬이 된다는 뜻임
(힌트 : 134217728 = 2^27)

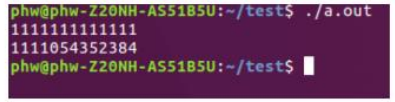
정답

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

int get_bit_order(int x, int order)
{
        return x & ~(int)(powl(2, order) - 1);
}

int main(void)
{
        int res, x = 394217728;
        printf("x = 0x%x, x(int) = %d\n", x, x);
        res = get_bit_order(x, 27);
        printf("res = 0x%x, res(int) = %d\n", res, res);
        return 0;
}
```

내가 쓴 답안

임베디드 애플리케이션 분석
20.





오답노트

답은 맞았지만, 너무 쉽게 생각한 것 같다.

답은 맞았지만, 2의 제곱근 방식으로도 할 수 있다는 것을 배웠고

문제를 풀 때 더 생각할 필요가 있다.

1. 임베디드 애플리케이션 분석

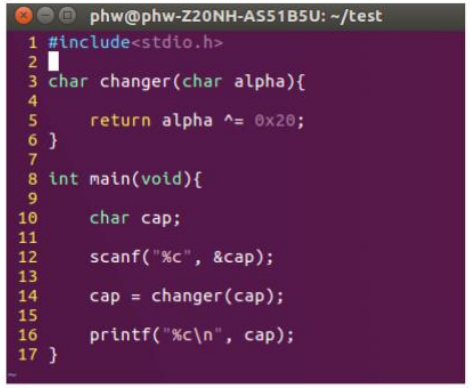
30. 단 한 번의 연산으로 대소문자 전환을 할 수 있는 연산에 대해 기술하시오.
(프로그래밍 하시오), 덧셈 혹은 뺄셈 같은 기능이 아님

정답

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

void chg_order(char *str)
{
        int i, len = strlen(str);

        printf("len = %d\n", len);

        for(i = 0; i < len; i++)
                if((str[i] >= 97 && str[i] < 123) || (str[i] >= 65 && str[i] < 91))
                        str[i] ^= 0x20;
}

int main(void)
{
        char test[10] = "Who am I";
        printf("test = %s\n", test);
        chg_order(test);
        printf("test = %s\n", test);

        return 0;
}
```

내가 쓴 답안

임베디드 애플리케이션 분석
21.



오답노트

이 문제도 답은 맞았지만,

한 문자가 아니라 문장을 바꿔야 했는데,

너무 쉬운 단일 문자로 잘못한 것 같다.

1. 임베디드 애플리케이션 분석

52. 기계어 레벨에서 스택을 조정하는 명령어는 어떤것들이 있는가 ?

정답

```
#if NO34
push, pop, ret, call
#endif
```

내가 쓴 답안

임베디드 애플리케이션 분석
34.

push pop jump ret

오답노트

와... jump를 왜 쓴거지?

Call 이 push & jump인데,,,

다음부터 이런 실수하면 안되겠다.

1. 임베디드 애플리케이션 분석

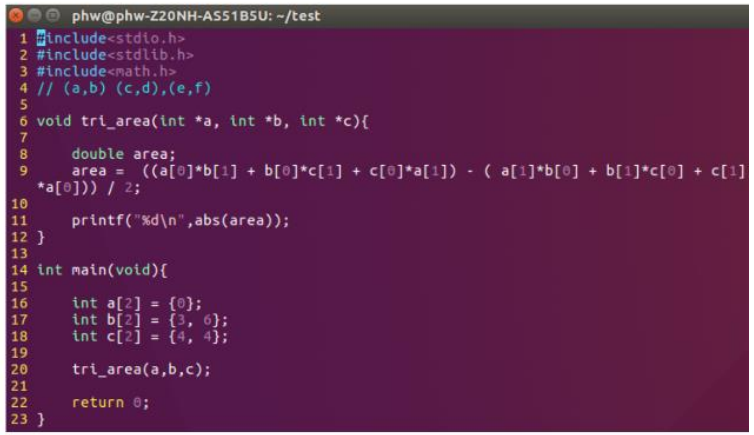
53. a 좌표(3, 6), b 좌표(4, 4) 가 주어졌다.
원점으로부터 이 두 좌표가 만들어내는 삼각형의 크기를 구하는 프로그램을 작성하라.

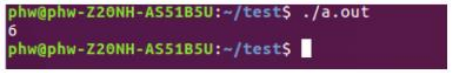
정답

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

typedef struct __coord
{
        float x;
        float y;
} coord;

void init_coord(coord *A, int x, int y)
{
        A->x = x;
        A->y = y;
}

float calc_vec_area(coord *A, coord *B)
{
        return 0.5 * fabs(A->x * B->y - B->x * A->y);
}

int main(void)
{
        coord A;
        coord B;
        float area;

        init_coord(&A, 3, 6);
        init_coord(&B, 4, 4);

        area = calc_vec_area(&A, &B);
        printf("area = %f\n", area);

        return 0;
}
```

내가 쓴 답안

임베디드 애플리케이션 분석
35.





오답노트

좌표 3개를 알 때 사용하는 공식을 사용했다.

나는 배열을 이용했지만, 풀이에서는 구조체를 사용하여

문제를 풀었다. 또한,

Math함수에 fabs 실수형 절대값이 있다는 것도 알게 되었다.

1. 임베디드 애플리케이션 분석

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.

정답

```c
#include <stdio.h>

void find_scaling(float xm, float xp, float ym, float yp,
                                float *xs, float *ys, float xmon, float ymon)
{
        *xs = xmon / xp;
        *ys = ymon / yp;
}

int main(void)
{
        float x_minus = -12.0;
        float x_plus = 12.0;
        float y_minus = -8.0;
        float y_plus = 8.0;
        float x_scale, y_scale;
        float x_mon = 800, y_mon = 600;

        find_scaling(x_minus, x_plus, y_minus, y_plus,
                            &x_scale, &y_scale, x_mon, y_mon);

        printf("x_scale = %f\n", x_scale);
        printf("y_scale = %f\n", y_scale);
}
```

내가 쓴 답안

오답노트

이해하고 싶으나 무엇을 하는 것인지 잘 몰르겠어서 pass한 문제이다.

질문을 통해 해결해야겠다.

1. 임베디드 애플리케이션 분석

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

내가 쓴 답안

정답

오답노트

이 문제도... 지금 풀기는 너무 어렵다.

2. 오답노트 후기

오답노트를 하면서 오답노트를 할 수 있는 문제보다 할 수 없는 어려운 문제들이 더 많았다.

내가 생각한 것 보다 부족한 부분들도 너무 많은 것 같다.

문제를 풀면서 막상 코딩을 해보니, 문법적인 부분의 이해적인 측면도 많이 부족했고

이론적인 측면에서도 띄엄띄엄 알고 있다는 것을 느꼈다.

그렇지만,

선생님의 답안을 보면서 C언어에서 부족한 문법적인 측면을 조금이나마 해소한 것 같고,

이론적인 부분과 용어적인 이해도가 조금은 높아진 것 같다.

아직은 푸는 문제 반, 못 푸는 문제 반이지만 하드웨어, 수학적인 코딩과 자료구조

부분을 더 보완하면 다음에는 80%까지는 풀 수 있다고 생각한다.

앞으로 더 공부하고 채워서 더 잘해지고 싶다...하... 할게 너무 많다...