Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 장성환
redmk1025@gmail.com

2. 배열에 아래와 같은 정보들이 들어있다.
여기서 가장 빈도수가 높은 3 개의 숫자를 찾아 출력하시오!
함수에서 이 작업을 한 번에 찾을 수 있도록 하시오.
(찾는 작업을 여러번 분할하지 말란 뜻임)

```
#include <stdio.h>

typedef struct map
{
        int freq;
        int cnt;
} hash;

int dup_cnt;
hash dup[100];

int check_dup(int num)
{
        int i;

        for (i = 0; i < dup_cnt; i++)
                if (dup[i].freq == num)
                        return i;

        return -1;
}

int find_kinds_freq(int *arr, int size)
{
        int i;

        /*
          1. 값을 넣고
```

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

typedef struct _queue{
  int data;
  int cnt;
  struct _queue* link;
}queue;
queue* get_node(){
  queue *tmp = (queue*)malloc(sizeof(queue));
  tmp->cnt = 0;
  tmp->link = NULL;
}
queue* enqueue(queue *head, int data){

  if(head==NULL){
        head=get_node();
        head->data=data;
        return head;
  }
  if(head->data == data){
        head->cnt ++;
        return head;
  }
  else{
        head->link=enqueue(head->link,data);
  }
  return head;
}

void printf_queue(queue *head){
  if(head==NULL)
```

```
            2. 다음에 들어오는 값이 이전 값중 같은게 있다면
            3. cnt 를 증가시킨다.
          */
        for (i = 0; i <= size; i++)
        {
                int dup_loc;
                if ((dup_loc = check_dup(arr[i])) != -1)
                {
                        if (arr[i] == 1)
                                printf("Here\n");
                        dup[dup_loc].cnt++;
                }
                else
                {
                        dup[dup_cnt].freq = arr[i];
                        dup[dup_cnt].cnt++;
                        dup_cnt++;
                }
        }

        return dup_cnt;
}

void print_arr(hash *arr)
{
        int i;

        for (i = 0; i < dup_cnt - 1; i++)
                printf("freq = %d, cnt = %d\n", arr[i].freq, arr[i].cnt);

        puts("");
}

void sort(hash *arr, int len)
```

```
        return;
  printf("%d %d\n",head->data,head->cnt);
  printf_queue(head->link);
}

void printf_major(queue *head){
  int arr[3]={0};
  int tmp;

  if(head == NULL)
        return;



}

int main(void){

  int arr[]={2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400,
2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400, 2400,
2400, 2400, 2400, 2400, 2400, 2400, 2400, 5000, 5000, 5000, 5000, 5000,
5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000,
5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000, 5000,
5000, 5000, 5000, 5000, 5000, 5000, 5000, 500, 500, 500, 500, 500, 500,
500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500, 500,
500, 500, 500, 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, 978, 456, 234756 , 234 ,4564 ,3243, 876,645, 534, 423,
312, 756, 235 ,75678};
  queue *head = NULL;

  for(int i=0;i<sizeof(arr)/sizeof(int);i++){
```

```
{
	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].cnt > 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;
	}
}

void print_many_freq(void)
{
	int i;

	for(i = dup_cnt - 2; i > dup_cnt - 5; i--)
		printf("dup[%d].freq = %d\n", i, dup[i].freq);
}

int main(void)
{
	int arr[] = {
		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,
```

```
		head=enqueue(head,arr[i]);
  }

  printf_queue(head);

  return 0;
}
```

내가 짠 코드는 큐에 입력할때, 이전에 저장된 정보를 검색하여 같은 값이 있으면 **cnt** 의 값을 하나 올려주는 것이었고, 시간이 없어서 가장 많은 **cnt** 를 찾아 소팅하여 출력을 완성시키지 못했다.

답안은,

구조체 **hash** 를 선언하고 값을 저장하는 **freq** 와 몇개가 같은지 나타내는 **cnt** 로 구성된다.

find_kinds_freq()함수를 통하여

원래 배열과 배열의 크기만큼 반복하여 숫자를 집어넣고
check_dup()함수를 통하여 저장이 되었는지 아닌지 판단하고, 저장이 되어있다면,
저장된 값의 위치를 반환하여 해당 **cnt** 를 올려준다.
저장이 되지 않았다면 값을 저장하고 **cnt =1** 로 맞춘다.
그리고 새로 생성된 **hash** 구조체의 크기를 리턴한다.

sort()함수를 통하여 해쉬 구조체의 크기만큼 반복이 되며
확인하고자 하는 값의 **cnt** 와 이전에 저장된 **cnt** 를 반복 비교하여
cnt 가 큰 값을 인덱스가 뒤로 가도록 보내게 된다.

```
            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
      };
```

```
	int size = sizeof(arr) / sizeof(int);
	printf("size = %d\n", size);

	hash *map_arr = NULL;

	size = find_kinds_freq(arr, size);
	printf("dup_cnt = %u\n", dup_cnt);
	print_arr(dup);

	sort(dup, size);
	print_arr(dup);

	print_many_freq();

	return 0;
}
```

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.

```
#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;
}
```

입력값을 0~4096 이라고 할때,
출력값을 -5~5 V 로 만드는 변환

즉 slice()함수는 입력값을 0~10V 의 출력으로 변환 (값의 구간)하고
adc 함수는 -5 를 통하여 출력을 -5 ~ 5 로 변환한다.

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.

```
#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;
}
```

strncpy 를 통하여 찾고자 하는 문자의 길이와 문자열을 넘겨주고

리턴값을 이용하여 인덱스를 알아내도록 한다.

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

```
#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;
	}
```

적분은 dx 를 dy 만큼 곱한 넓이를 구간만큼 더해주면 나오는 넓이라는 개념이므로,

dx(0.000002504)만큼씩 1~3 까지의 구간을 나눈것을 int_e
dy 는 1~3 구간까지의 y 의 값이다.

따라서 넓이는 dx*dy 이므로
sum 에 int_e 만큼 반복하여 넣으면
근사치로 적분값이 저장된다.

```
	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;
}
```

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.

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

void set_rand_mem(int *mem)
{
	int i;

	for(i = 0; i < 100; i++)
		mem[i] = rand() % 4096 + 1;
}

void print_mem(int *mem)
{
	int i;

	for(i = 0; i < 100; i++)
		printf("mem[%d] = %d\n", i, mem[i]);
}

void check_waste_mem(int *mem, int *waste)
{
	int i;

	for(i = 0; i < 100; i++)
		waste[i] = 4096 - mem[i];
}

int sum_waste(int *waste)
{
```

해당 코드는
0~99 까지 인덱스를 가진 배열에 랜덤 수(1~4096)를 집어넣고
4096 에서 해당 배열을 뺀 waste_mem 을 저장한다.

각각의 waste_mem 의 인자를 모두 더하여
최대 4096 에서 얼마나 차이가 났는지
확인한다.

```
	int i, tot = 0;

	for(i = 0; i < 100; i++)
		tot += waste[i];

	return tot;
}

int main(void)
{
	int dissipation;
	int mem[100] = {0};
	int waste_mem[100] = {0};

	srand(time(NULL));
	set_rand_mem(mem);

	print_mem(mem);

	check_waste_mem(mem, waste_mem);

	print_mem(waste_mem);

	dissipation = sum_waste(waste_mem);
	printf("dissipation = %d\n", dissipation);

	return 0;
}
```

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)

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

```
#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);
}
```

원점에서 한점 사이의 거리

sqrt(x^2 + y^2)

점과 직선 사이의 거리

$(|x1 * y2 - x2 * y1|) / (\sqrt{x2^2 + y2^2})$

따라서 0.5 * sqrt(x1 * y2 - x2 * y1)으로 삼각형의 넓이를 구할 수 있다.

```
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;
}
```

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

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

double rad_2_deg(double rad)
{
	return rad * 180.0 / M_PI;
}

double deg_2_rad(double deg)
{
	return deg * M_PI / 180.0;
```

```
}

double fact(double val)
{
	double first = 1;

	while(val > 0)
		first = first * val--;

	return first;
}

double taylor_sin(double rad)
{
	double res = 0;

	res = rad - pow(rad, 3.0) / fact(3) + pow(rad, 5.0) / fact(5)
		- pow(rad, 7.0) / fact(7) + pow(rad, 9.0) / fact(9);

	return res;
}

bool check_sin(void)
{
	double math_sin = sin(M_PI/4.0);
	double my_sin = taylor_sin(M_PI/4.0);

	printf("math_sin = %lf, pi/4 = %lf degree\n", math_sin,
rad_2_deg(M_PI/4.0));
	printf("my_sin = %lf\n", my_sin);

	if((int)math_sin * 1000000 == (int)my_sin * 1000000)
		return true;
	else
```

```
                return false;
}

int main(void)
{
        bool res = check_sin();

        if(res)
                printf("It's similar\n");
        else
                printf("It's not similar\n");

        return 0;
}
```