

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

1. 단 한 번의 연산으로 대소문자 전환을 할 수 있는 연산에 대해 기술하시오.
- 비트 연산을 통하여 한번에 대소문자 전환이 가능하다.

```
sunghwan@HWAN: ~/Documents
1 #include <stdio.h>
2
3 int main(void){
4     char a;
5     scanf("%c",&a);
6     a=a^32;
7
8     printf("%c\n",a);
9
10    return 0;
11 }
```

2. Stack 및 Queue 외에 Tree 라는 자료구조가 있다. 이 중에서 Tree 는 Stack 이 나 Queue 와는 다르게 어떠한 이점이 있는가 ?

스택이나 큐는 N 개가 저장되어 있을 때, 최대 N-1 번 자료를 탐색하게 된다.
하지만 트리구조는 어떤 기준점을 가지고 구분하여 저장하기 때문에 자료의 탐색시 훨씬 빠른 탐색속도를 보이게 된다.
완전 이진트리라고 가정할 경우 $N < 2^x$ 라면 최대 x 번 탐색하게 된다.

문제 임의의 값 x 가 있는데, 이를 4096 단위로 정렬하고 싶다면 어떻게 해야할까 ?
C 언어로 프로그래밍 하시오.

무슨 뜻인지...

int p[7] 와 int (*p)[7] 가 있다.
이 둘의 차이점에 대해 기술하시오.

Int p[7] 은 int 형 타입의 변수가 7 개 나열되어 있는 포인터 상수 p
(*p)[7] int 형 타입의 변수 7 개가 나열되어 있는 자료구조의 주소값을 저장할 수 있는 포인터 변수 p 이다. 따라서 2 중 배열을 지정할 때 쓰게 된다.

이번 문제의 힌트를 제공하자면 함수 포인터와 관련된 문제다.

아래와 같은 행렬을 생각해보자!

1 2 3

1 2 3

sapply(arr, func) 이라는 함수를 만들어서 위의 행렬을 아래와 같이 바꿔보자!

sapply 에 func 함수가 연산을 수행하는 함수로 만들어야 한다.

1 2 3

1 4 9

```
sunghwan@HWAN: ~/Documents
1 #include <stdio.h>
2
3 int func(int data){
4     int res;
5     res=data*data;
6     return res;
7 }
8
9 void sapply(int (*p)[3],int (*f)(int data)){
10     for(int i=0;i<3;i++){
11         p[1][i]=f(p[0][i]);
12     }
13 }
14
15 int main(void){
16     int arr[][3]={{1,2,3},{1,2,3}};
17     sapply(arr,func);
18
19     for(int i=0;i<2;i++){
20         for(int j=0;j<3;j++){
21             printf("%d ",arr[i][j]);
22         }
23         printf("\n");
24     }
25     return 0;
26 }
```

Intel Architecture 와 ARM Architecture 의 차이점은 무엇인가 ?	빅 엔디안 리틀 엔디안 바이트의 저장 순서가 다르다 빅엔디안은 1,2,3,4 순으로 배치가 되면 리틀 엔디안은 4,3,2,1 순으로 배치가 된다.
---	--

이것이 없으면 C 언어의 함수를 호출할 수 없다. 여기서 이야기하는 이것은 무엇일까 ?	JMP ? 데이터의 주소?
---	----------------

<p>$3x^2 + 7x$ 를 1 ~ 2 까지 정적분하는 프로그램을 구현해보라.</p> <p>$3x^2$ 에서 x^2 는 제곱을 의미한다.</p> <p>예로 x 에 1 이 들어가면 $3x^2 = 9$ 가 된다.</p>	 <pre> 1 #include <stdio.h> 2 #include <stdlib.h> 3 4 5 double make_pre(double a, double upper){ 6 double res; 7 res = a/upper; 8 return res; 9 } 10 11 double make_upp(double upper){ 12 return upper+1; 13 } 14 15 double poww(double x, double y){ 16 if(y==1){ 17 return x; 18 } 19 return x*poww(x,y-1); 20 } 21 22 double calc(int x){ 23 double res; 24 res = make_pre(3,make_upp(2))*poww(x,make_upp(2))+make_pre(7,make_upp(1))*poww(x,make_upp(1)) 25 ; 26 return res; 27 } 28 29 int main(void){ 30 double result; 31 32 printf("%lf\n",calc(2)-calc(1)); 33 34 return 0; 35 } </pre>
--	--

Memory Hierarchy(메모리 계층 구조)에 대해 기술하시오.	<p>Register</p> <ul style="list-style-type: none"> - CPU 의 연산을 위한 저장소 레지스터에서 레지스터로 연산 <p>cache</p> <ul style="list-style-type: none"> - CPU 저장장치 CPU 와 메모리의 속도 차이를 줄이기 위하여 쓴다. 버퍼라고도 한다. 메모리의 자주 쓰이는 데이터를 캐시로 이동시켜서 저장한다. <p>memory</p> <ul style="list-style-type: none"> -외부 저장장치 전원이 나가면 데이터의 내용이 지워진다. 필요한 디스크의 데이터를 메모리로 이동시켜 저장한다. <p>disk</p> <ul style="list-style-type: none"> -외부 저장장치 전원이 나가도 데이터의 내용이 지워지지 않는다.
<p>C 언어에서 중요시하는 메모리 구조에 대해 기술하시오.</p> <p>(힌트: Stack, Heap, Data, Text 에 대해 기술하시오.)</p>	<p>STACK</p> <ul style="list-style-type: none"> -컴파일 시간에 결정되는 메모리 구조 <p>HEAP</p> <ul style="list-style-type: none"> -프로그램 실행 중 사용자가 메모리 동적 할당을 통하여 값들을 저장 <p>DATA</p> <ul style="list-style-type: none"> -전역 변수들을 저장하는 공간 <p>TEXT</p> <ul style="list-style-type: none"> -모르겠음.
파이프라인이 깨지는 경우에 대해 기술하시오.	<p>분기명령어가 쓰일 경우에 파이프 라인이 깨지게 된다.</p> <p>예로 단순한 3 단계 파이프라인이라고 하면,</p> <p>Fetch-decoding-execute 방식의 3 단계 로 CPU 의 동작이 되는데 빠른 실행을 위해 각 단계가 끝나면 다음에 실행할 명령어를 읽어 들인다.</p> <p>분기 명령어가 완료될 경우 지금까지 미리 읽어 들였던 명령어를 버리고 새롭게 fetch 해야 하므로 파이프 라인이 깨지게 된다.</p>
void (* signal(int signum, void (* handler)(int)))(int)라는 signal 함수의 프로토타입을 기술하시오.	void (*)(int) signal(int signum, void (* handler)(int))

<p>프로토타입을 기술하라는 의미는 반환형(리턴 타입)과 함수의 이름, 그리고 인자(파라미터)가 무엇인지 기술하라는 뜻임.</p>	<p>반환형: void 를 리턴하고 int 를 인자로 받는 함수</p> <p>이름 : signal</p> <p>매개변수: int 형 인자와 void 반환 하고 int 를 인자로 받는 함수 .</p>
<p>goto 를 사용하는 이유에 대해 기술하시오.</p>	<p>2 중 이상의 반복문일 경우 GOTO 를 사용하여 내부 반복문에서 벗어나면 효율이 좋다. Break 사용시 이전 반복이 계속 진행되기 때문이다.</p>

TI Cortex-R5F Safety MCU is very good to Real-Time System.

위의 문장에서 Safety MCU 가 몇 번째 부터 시작하는지 찾아내보자!

(배열의 인덱스로 표기해도 상관없고, 실제 위치로 표기해도 상관없으며 둘 중 무엇인지 표기하시오)

```
sunghwan@HWAN: ~/Documents
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 void make_str(char **str, int size){
6
7     *str = (char*)malloc(size+1);
8 }
9
10
11 int main(void){
12     char str[]="TI Cortex-R5F Safety MCU is very good to Real-Time System";
13     char *tmp=(char*)malloc(sizeof(str));
14     int i;
15     int cnt=0;
16
17     for(i=0;i<sizeof(str);i++){
18         if(str[i]==' '|| str[i]=='\0'){
19             tmp[cnt+1]='\0';
20             //printf("%s\n",tmp);
21             if(strcmp(tmp,"Safety")==0){
22                 printf("string array[%d] is Starting point!\n",i-6);
23             }
24             make_str(&tmp,sizeof(str));
25             cnt=0;
26
27             continue;
28         }
29         tmp[cnt]=str[i];
30         cnt++;
31     }
32
33     return 0;
34 }
```

```
sunghwan@HWAN: ~/Documents
sunghwan@HWAN:~/Documents$ ./a.out
string array[14] is Starting point!
sunghwan@HWAN:~/Documents$
```

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,
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, 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, 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

[illegible]

<p>아래 질문에 적절한 C 코드를 구현해보자. 내용</p> <p>16 비트 ADC 를 가지고 있는 장치가 있다.</p> <p>보드는 12 V 로 동작하고 있고 ADC 는 -5 ~ 5 V 로 동작한다.</p> <p>ADC 에서 읽은 값이 12677 일 때 이 신호를 전압으로 변환해보자!</p> <p>(실제 보드가 없으므로 단순히 C 로 구현하면 된다)</p>	
---	--

<p>설명을 읽고 C 코드를 구현해보자. 내용</p> <p>24 비트 DAC 장치가 있다.</p> <p>이 장치는 -12 V ~ 12 V 로 동작하며 보드는 5 V 로 동작한다.</p> <p>DAC 에서 나온 전압이 9.7 V 일 때 어떤 디지털 신호를 입력 받은것인지 파악해보자!</p>	
---	--

<p>다음 질문에 적절한 답을 적으시오(주관식) 내용</p> <p>운영체제의 5 대 요소 5 가지를 적으시오.</p>	
---	--

Stack 자료구조를 아래와 같은 포맷에 맞춰 구현해보시오.

(힌트: 이중 포인터)

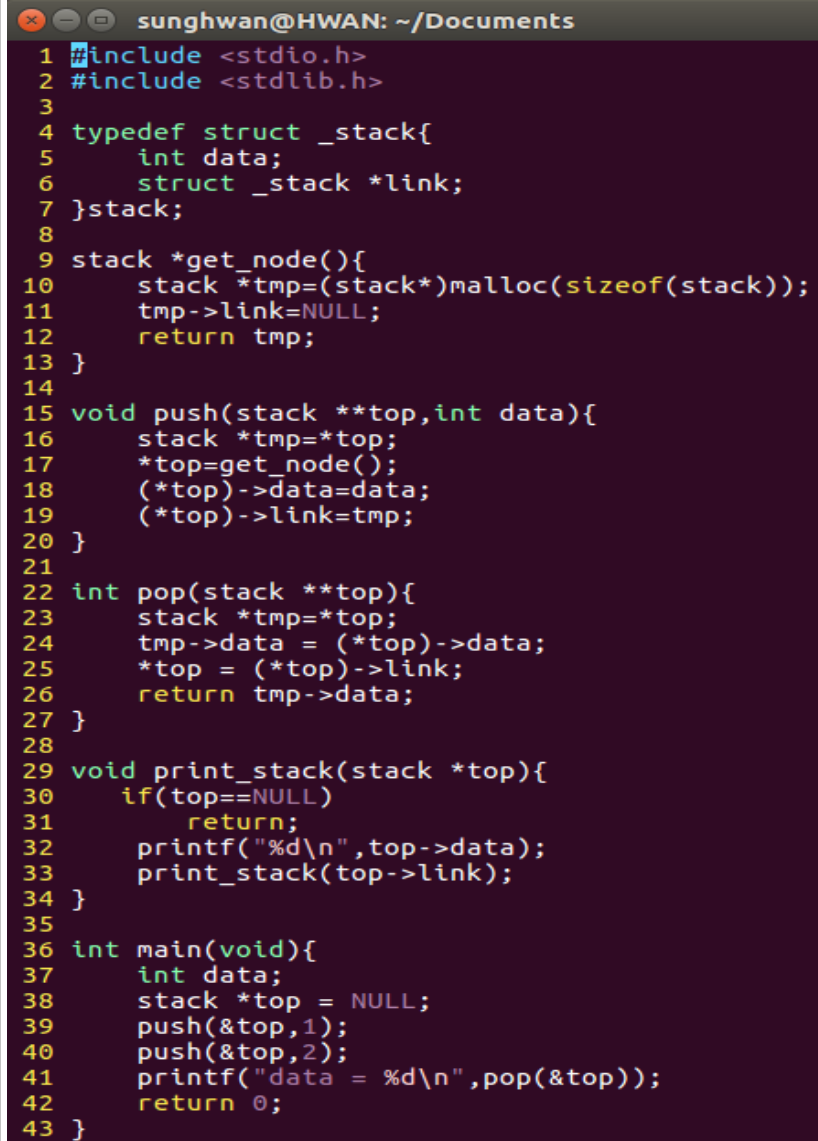
ex)

```
int main(void)
{
    stack *top = NULL;

    push(&top, 1);

    push(&top, 2);

    printf("data = %d\n", pop(&top));
}
```



```
sunghwan@HWAN: ~/Documents
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct _stack{
5     int data;
6     struct _stack *link;
7 }stack;
8
9 stack *get_node(){
10     stack *tmp=(stack*)malloc(sizeof(stack));
11     tmp->link=NULL;
12     return tmp;
13 }
14
15 void push(stack **top,int data){
16     stack *tmp=*top;
17     *top=get_node();
18     (*top)->data=data;
19     (*top)->link=tmp;
20 }
21
22 int pop(stack **top){
23     stack *tmp=*top;
24     tmp->data = (*top)->data;
25     *top = (*top)->link;
26     return tmp->data;
27 }
28
29 void print_stack(stack *top){
30     if(top==NULL)
31         return;
32     printf("%d\n",top->data);
33     print_stack(top->link);
34 }
35
36 int main(void){
37     int data;
38     stack *top = NULL;
39     push(&top,1);
40     push(&top,2);
41     printf("data = %d\n",pop(&top));
42     return 0;
43 }
```

Binary Tree 나 AVL Tree, Red-Black Tree 와 같이 Tree 계열의 자료구조를 재귀 호출 없이 구현하고자 한다.

이 경우 반드시 필요한 것은 무엇인가?

더블 포인터?

Binary Tree 를 구현하시오.

초기 데이터를 입력 받은 이후 다음 값이 들어갈 때 작으면 왼쪽 크면 오른쪽으로 보내는 방식으로 구현하시오.

삭제 구현이 가능하다면 삭제도 구현하시오.

```
60 tree *delete_tree(tree *root, int data){
61     int num;
62     tree *tmp;
63     if(root == NULL){
64         printf("Not Found\n");
65         return NULL;
66     }
67     else if(root->data > data){
68         root->left = delete_tree(root->left,data);
69     }
70     else if(root->data < data){
71         root->right = delete_tree(root->right,data);
72     }
73     else if(root->left && root->right){
74         root->left = find_max(root->left,&num);
75         root->data = num;
76     }
77     else
78         root = chg_node(root);
79     return root;
80 } // 트리 삭제 함수
81
82
83
84 int main(void){
85     int i;
86     int data[14]={50,45,73,32,48,46,16,37,120,47,130,127,124};
87     tree *root = NULL;
88     for(i=0;data[i];i++){
89         tree_ins(&root,data[i]);
90     }
91
92     print_tree(root);
93     printf("\n");
94
95     delete_tree(root,50);
96     print_tree(root);
97
98     return 0;
99 }
```

```
sunghwan@HWAN: ~/Documents
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct _tree{
5     int data;
6     struct _tree *left;
7     struct _tree *right;
8 }tree;
9
10 tree* get_node(){
11     tree * tmp=(tree*)malloc(sizeof(tree));
12     tmp->left=NULL;
13     tmp->right=NULL;
14     return tmp;
15 }
16
17
18 void tree_ins(tree **root, int data){
19     if(*root==NULL){
20         *root = get_node();
21         (*root)->data =data;
22         return;
23     }
24     else if((*root)->data >data)
25         tree_ins(&(*root)->left,data);
26     else if((*root)->data < data)
27         tree_ins(&(*root)->right , data);
28 } //트리 삽입
29
30 void print_tree(tree *root){
31     if(root){
32         printf("%d\n",root->data);
33         print_tree(root->left);
34         print_tree(root->right);
35     }
36 } // 트리 표현
37
38 tree *chg_node(tree *root){
39     tree *tmp =root;
40
41     if(!root->right)
42         root = root->left;
43     else if(!root->left)
44         root = root->right;
45     free(tmp);
46
47     return root;
48 } //chg_node 함수
49
50 tree *find_max(tree *root, int *data){
51     if(root->right)
52         root->right = find_max(root->right, data);
53     else{
54         *data = root->data;
55         root = chg_node(root);
56     }
57     return root;
58 } //find max 함수
59
```

<p>AVL 트리는 검색 속도가 빠르기로 유명하다.</p> <p>Red-Black 트리도 검색 속도가 빠르지만 AVL 트리보다 느리다.</p> <p>그런데 어째서 SNS 솔루션등에서는 AVL 트리가 아닌 Red-Black 트리를 사용할까?</p>	<p>AVL 트리가 RB 트리보다 탐색속도가 더 빠르지만 삽입등의 작업에서 더 많은 컴퓨터 자원을 소모한다. 삽입 삭제가 빈번하게 일어날 경우 RB 트리가 AVL 트리보다 훨씬 효율이 좋기에 RB 트리를 사용한다.</p>
---	---

AVL 트리를 C 언어로 구현하시오.

AVL 트리는 밸런스 트리로 데이터가 1, 2, 3, 4, 5, 6, 7, 8, 9 와 같이

순서대로 쌓이는 것을 방지하기 위해 만들어진 자료구조다

```
sunghwan@HWAN: ~/Documents
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef enum _rot{
5     RR=1,
6     RL=2,
7     LL=3,
8     LR=4
9 }rot;
10
11 typedef struct _avl{
12     int lev;
13     int data;
14     struct _avl *left;
15     struct _avl *right;
16 }avl;
17
18 avl* get_node(){
19     avl * node = (avl*)malloc(sizeof(avl));
20     node->left=NULL;
21     node->right=NULL;
22     node->lev=1;
23     return node;
24 }
25
26 int update_level(avl *root){
27     int left = root->left ? root->left->lev : 0;
28     int right = root->right ? root->right->lev : 0;
29
30     if(left>right)
31         return left+1;
32     return right+1;
33 }
34
35 int rotation_check(avl *root){
36     int left = root->left ? root->left->lev : 0;
37     int right = root->right ? root->right->lev : 0;
38
39     return right - left;
40 }
41
42 int kinds_of_rot(avl *root, int data){
43     printf("data = %d\n",data);// 1
44
45     if(rotation_check(root)>1){
46         if(root->right->data > data)
47             return RL;
48         return RR;
49     }
50     else if(rotation_check(root) < -1){
51         if(root->left->data < data)
52             return LR;
53         return LL;
54     }
55 }
```

```
sunghwan@HWAN: ~/Documents
57 avl *rr_rot(avl *parent, avl *child) //input grand, and parent, not son
58 {
59     parent->right = child->left;
60     child->left = parent;
61     parent->lev = update_level(parent);
62     child->lev = update_level(child);
63     return child;
64 }
65
66 avl *ll_rot(avl *parent, avl *child)
67 {
68     parent->left = child->right;
69     child->right = parent;
70     parent->lev = update_level(parent);
71     child->lev = update_level(child);
72     return child;
73 }
74
75 avl *rl_rot(avl *parent, avl *child)
76 {
77     child = ll_rot(child, child->left);
78     return rr_rot(parent, child);
79 }
80
81 avl *lr_rot(avl *parent, avl *child)
82 {
83     child = rr_rot(child, child->right);
84     return ll_rot(parent, child);
85 }
86
87
88 avl* rotation(avl *root, int ret){ //input-parents root, rot_status
89     switch(ret){
90         case RL:
91             printf("RL Rotation\n");
92             return rl_rot(root, root->right);
93         case RR:
94             printf("RR Rotation\n");
95             return rr_rot(root, root->right);
96         case LR:
97             printf("LR Rotation\n");
98             return lr_rot(root, root->left);
99         case LL:
100             printf("LL Rotation\n");
101             return ll_rot(root, root->left);
102     }
103 }
104
```

```

105 void avl_ins(avl **root, int data){
106     if(!(*root)){
107         (*root) = get_node();
108         (*root)->data = data;
109         return;
110     }
111
112     if((*root)->data > data)
113         avl_ins(&(*root)->left,data);
114     else if((*root)->data < data)
115         avl_ins(&(*root)->right,data);
116
117     (*root)->lev = update_level(*root);
118
119     if(abs(rotation_check(*root))>1){
120         printf("Rotation !\n");
121         *root = rotation(*root, kinds_of_rot(*root,data));
122     }
123 }
124
125 void printf_avl(avl *root){
126     if(root==NULL)
127         return;
128     printf("%d\n",root->data);
129     printf_avl(root->left);
130     printf_avl(root->right);
131 }
132 int main(void){
133
134     int i;
135     int arr[]={1,2,3,4,5,6};
136     avl *root=NULL;
137
138     for(i=0;i<sizeof(arr)/sizeof(int);i++){
139         avl_ins(&root,arr[i]);
140     }
141     printf_avl(root);
142
143
144
145     return 0;
146 }

```