자료구조 1번

자료구조	
문제	[복합문제 1.1] 값이 1 ~ 4096까지 무작위로 할당되어 배열에 저장되도록 프로그래밍 하시오.
내용	배열의 크기는 100 개 정도로 잡는다

howard@ubuntu: ~/HomeworkBackup/test

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

int main(void){
    srand(time(NULL));
    int a[100];
    int i = 0;
    while(i<100){
        a[i] = rand()%4096 + 1;
        printf("%d\n",*(a+i));
        i++;
    }
    return 0;
}
```

howard@ubuntu: ~/HomeworkBackup/test

```
1106
299
2088
3958
1476
3939
2826
1027
3719
438
626
399
3028
2386
3329
3186
2643
3096
2148
3432
1959
1505
1938
howard@ubuntu:~/HomeworkBackup/test$
```

자료구조 2번

자료구조	
문제	**[복합문제 1.2] 앞서 만든 코드를 아래 설명을 보고 개량 하도록 한다.**
내용	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.

howard@ubuntu: ~/HomeworkBackup/test

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
int main(void){
    srand(time(NULL));
    int space[100];         //물건을 담을 수 있는 공간
    int input[100] = {0,}; // 담긴 물건
    int waste[100]={0,};   //낭비된 공간
    int i = 0;
    while(i<100){
        space[i] = rand()%4096 + 1;
        printf("%d\n",*(space+i));
        i++;
    }
    for(i=0;i<100;i++){
        waste[i] = space[i] - input[i];//낭비된 공간 : 수용량 - 사용량
    }
    return 0;
}
```

자료구조 3번

자료구조	
문제	**[복합문제 1.3] 앞서 만든 코드를 보다 더 개량한다.**
내용	문제에서 확장하여 공간을 보다 효율적으로 관리하고 싶어서 4096, 8192, 16384 등의 4096 배수로 크기를 확장할 수 있는 시스템을 도입했다. 이제부터 공간의 크기는 4096의 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)

howard@ubuntu: ~/HomeworkBackup/test

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(void){
    srand(time(NULL));
    int space[100];         //물건을 담을 수 있는 공간
    int input[100] = {0,}; // 담긴 물건
    int waste[100]={0,};   //낭비된 공간
    int i = 0;
    while(i<100){
        space[i] = rand()%131072 + 1;
        if(space[i] >= 4096)              //최솟값 : 4096
            space[i] = space[i]&(-4096); //난수 : 0~131072 ->4096배수
        else
            space[i] = 4096;
        i++;
    }
    for(i=0;i<100;i++){
        waste[i] = space[i] - input[i];//낭비된 공간 : 수용량 - 사용량
    }
    return 0;
}
```

자료구조 4번

자료구조	
문제	**이진 트리를 재귀 호출을 사용하여 구현하도록 한다.**
내용	재귀 호출로 구현한다.

howard@ubuntu: ~/HomeworkBackup/data structu

```
#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);
}
```

howard@ubuntu: ~/HomeworkBackup/data structure

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

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

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

        print_tree(root->left);
        print_tree(root->right);
    }
}
tree *chg_node(tree *root)
{
    tree *tmp = root;
    printf("debug : %d\n",tmp->data);
    if(!root->right)
        root = root->left;
    else if(!root->left)
        root = root->right;

    free(tmp);

    return root;
}
```

howard@ubuntu: ~/HomeworkBackup/data structure

```
64 tree *find_max(tree *root, int *data)
65 {
66     if(root->right)
67         root->right = find_max(root->right, data);
68     else
69     {
70         *data = root->data;
71         root = chg_node(root);
72     }
73
74     return root;
75 }
76
77 tree *delete_tree(tree *root, int data)
78 {
79     int num;
80     tree *tmp;
81     if(root == NULL)
82     {
83         printf("Not Found\n");
84         return NULL;
85     }
86     else if(root->data > data)
87         root->left = delete_tree(root->left, data);
88     else if(root->data < data)
89         root->right = delete_tree(root->right, data);
90     else if(root->left && root->right)
91     {
92         root->left = find_max(root->left, &num);
93         root->data = num;
94     }
95     else{
96         root = chg_node(root);
97     }
98     return root;
99 }
```

howard@ubuntu: ~/HomeworkBackup/data structure

```
100 int main(void)
101 {
102     tree *root = NULL;
103     tree_ins(&root, 10);
104     tree_ins(&root, 20);
105     tree_ins(&root, 30);
106     tree_ins(&root, 5);
107     print_tree(root);
108
109     delete_tree(root, 20);
110     printf("After Delete\n");
111     print_tree(root);
112     return 0;
113 }
114
```

```
data = 10, left = 5, right = 20
data = 5, left = NULL, right = NULL
data = 20, left = NULL, right = 30
data = 30, left = NULL, right = NULL
debug : 20
After Delete
data = 10, left = 5, right = 30
data = 5, left = NULL, right = NULL
data = 30, left = NULL, right = NULL
```

자료구조 5번

자료구조	
문제	**이진 트리를 재귀 호출 없이 구현하도록 한다.**
내용	결과를 확인하는 print 함수(전위, 중위, 후위 순회) 또한 재귀 호출을 수행하면 안됨

howard@ubuntu: ~/HomeworkBackup/data structure

```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;
}
```

howard@ubuntu: ~/HomeworkBackup/data structure

```c
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;
}

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;
}
```

howard@ubuntu: ~/HomeworkBackup/data structure

```
63 void non_recur_tree_ins(tree **root, int data)
64 {
65     tree **tmp = root;
66
67     while(*tmp)
68     {
69         if((*tmp)->data > data)
70             tmp = &(*tmp)->left;
71         else if((*tmp)->data < data)
72             tmp = &(*tmp)->right;
73     }
74
75     *tmp = get_tree_node();
76     (*tmp)->data = data;
77 }
78
79 bool stack_is_not_empty(stack *top)
80 {
81     if(top != NULL)
82         return true;
83     else
84         return false;
85 }
86
```

howard@ubuntu: ~/HomeworkBackup/data structure

```
87 void print_tree(tree **root)
88 {
89     tree **tmp = root;
90     stack *top = NULL;
91
92     push(&top, *tmp);
93
94     while(stack_is_not_empty(top))
95     {
96         tree *t = (tree *)pop(&top);
97         tmp = &t;
98
99         printf("data = %d, ", (*tmp)->data);
100
101        if((*tmp)->left)
102            printf("left = %d, ", (*tmp)->left->data);
103        else
104            printf("left = NULL, ");
105
106        if((*tmp)->right)
107            printf("right = %d\n", (*tmp)->right->data);
108        else
109            printf("right = NULL\n");
110
111        push(&top, (*tmp)->right);
112        push(&top, (*tmp)->left);
113    }
114 }
115
116 tree *chg_node(tree *root)
117 {
118     tree *tmp = root;
119
120     if(!root->right)
121         root = root->left;
122     else if(!root->left)
123         root = root->right;
124
125     free(tmp);
126
127     return root;
128 }
```

howard@ubuntu: ~/HomeworkBackup/data structure

```
130 void find_max(tree **root, int *data)
131 {
132     tree **tmp = root;
133
134     while(*tmp)
135     {
136         if((*tmp)->right)
137             tmp = &(*tmp)->right;
138         else
139         {
140             *data = (*tmp)->data;
141             *tmp = chg_node(*tmp);
142             break;
143         }
144     }
145 }
146
147 void non_recur_delete_tree(tree **root, int data)
148 {
149     tree **tmp = root;
150     int num;
151
152     while(*tmp)
153     {
154         if((*tmp)->data > data)
155             tmp = &(*tmp)->left;
156         else if((*tmp)->data < data)
157             tmp = &(*tmp)->right;
158         else if((*tmp)->left && (*tmp)->right)
159         {
160             find_max(&(*tmp)->left, &num);
161             (*tmp)->data = num;
162             return;
163         }
164         else
165         {
166             (*tmp) = chg_node(*tmp);
167             return;
168         }
169     }
170     printf("Not Found\n");
171 }
```

```
173 int main(void)
174 {
175     tree *root = NULL;
176
177     non_recur_tree_ins(&root, 50);
178     non_recur_tree_ins(&root, 60);
179     non_recur_tree_ins(&root, 40);
180     non_recur_tree_ins(&root, 80);
181     non_recur_tree_ins(&root, 70);
182
183     print_tree(&root);
184
185     non_recur_delete_tree(&root, 50);
186     printf("After Delete\n");
187     print_tree(&root);
188
189     return 0;
190 }
```

```
data = 10, left = 5, right = 20
data = 5, left = NULL, right = NULL
data = 20, left = NULL, right = 30
data = 30, left = NULL, right = NULL
debug : 20
After Delete
data = 10, left = 5, right = 30
data = 5, left = NULL, right = NULL
data = 30, left = NULL, right = NULL
```

자료구조 6번.

자료구조	
문제	AVL 트리를 재귀 호출을 사용하여 구현하도록 한다.

howard@ubuntu: ~/HomeworkBackup/data structure

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
typedef enum __rot
{
    RR,
    RL,
    LL,
    LR
} rot;

typedef struct __avl_tree
{
    int lev;
    int data;
    struct __avl_tree *left;
    struct __avl_tree *right;
} avl;

void print_arr(int *arr, int size)
{
    int i;

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

avl *get_avl_node(void)
{
    avl *tmp;
    tmp = (avl *)malloc(sizeof(avl));
    tmp->lev = 1;
    tmp->left = NULL;
    tmp->right = NULL;
    return tmp;
}
```

howard@ubuntu: ~/HomeworkBackup/data structure

```
38 void print_tree(avl *root)
39 {
40     if(root)
41     {
42         printf("data = %d, lev = %d, ", root->data, root->lev);
43 
44         if(root->left)
45             printf("left = %d, ", root->left->data);
46         else
47             printf("left = NULL, ");
48 
49         if(root->right)
50             printf("right = %d\n", root->right->data);
51         else
52             printf("right = NULL\n");
53 
54         print_tree(root->left);
55         print_tree(root->right);
56     }
57 }
58 
59 int update_level(avl *root)
60 {
61     int left = root->left ? root->left->lev : 0;
62     int right = root->right ? root->right->lev : 0;
63 
64     if(left > right)
65         return left + 1;
66 
67     return right + 1;
68 }
69 
70 int rotation_check(avl *root)
71 {
72     int left = root->left ? root->left->lev : 0;
73     int right = root->right ? root->right->lev : 0;
74 
75     return right - left;
76 }
```

howard@ubuntu: ~/HomeworkBackup/data structure

```
78 int kinds_of_rot(avl *root, int data)
79 {
80     printf("data = %d\n", data);
81 
82     // for RR and RL
83     if(rotation_check(root) > 1)
84     {
85         if(root->right->data > data)
86             return RL;
87 
88         return RR;
89     }
90     // for LL and LR
91     else if(rotation_check(root) < -1)
92     {
93         if(root->left->data < data)
94             return LR;
95 
96         return LL;
97     }
98 }
99 
100 avl *rr_rot(avl *parent, avl *child)
101 {
102     parent->right = child->left;
103     child->left = parent;
104     parent->lev = update_level(parent);
105     child->lev = update_level(child);
106     return child;
107 }
108 
109 avl *ll_rot(avl *parent, avl *child)
110 {
111     parent->left = child->right;
112     child->right = parent;
113     parent->lev = update_level(parent);
114     child->lev = update_level(child);
115     return child;
116 }
```

howard@ubuntu: ~/HomeworkBackup/data structure

```
118 avl *rl_rot(avl *parent, avl *child)
119 {
120     child = ll_rot(child, child->left);
121     return rr_rot(parent, child);
122 }
123
124 avl *lr_rot(avl *parent, avl *child)
125 {
126     child = rr_rot(child, child->right);
127     return ll_rot(parent, child);
128 }
129
130 avl *rotation(avl *root, int ret)
131 {
132     switch(ret)
133     {
134         case RL:
135             printf("RL Rotation\n");
136             return rl_rot(root, root->right);
137         case RR:
138             printf("RR Rotation\n");
139             return rr_rot(root, root->right);
140         case LR:
141             printf("LR Rotation\n");
142             return lr_rot(root, root->left);
143         case LL:
144             printf("LL Rotation\n");
145             return ll_rot(root, root->left);
146     }
147 }
148
```

howard@ubuntu: ~/HomeworkBackup/data structure

```
149 void avl_ins(avl **root, int data)
150 {
151     if(!(*root))
152     {
153         (*root) = get_avl_node();
154         (*root)->data = data;
155         return;
156     }
157
158     if((*root)->data > data)
159         avl_ins(&(*root)->left, data);
160     else if((*root)->data < data)
161         avl_ins(&(*root)->right, data);
162     (*root)->lev = update_level(*root);
163     if(abs(rotation_check(*root)) > 1)
164         *root = rotation(*root, kinds_of_rot(*root, data));
165 }
166
167 avl *chg_node(avl *root)
168 {
169     avl *tmp = root;
170
171     if(!root->right)
172         root = root->left;
173     else if(!root->left)
174         root = root->right;
175
176     free(tmp);
177
178     return root;
179 }
```

howard@ubuntu: ~/HomeworkBackup/data structure

```
181 avl *find_max(avl *root, int *data)
182 {
183     if(root->right)
184         root->right = find_max(root->right, data);
185     else
186     {
187         *data = root->data;
188         root = chg_node(root);
189     }
190
191     return root;
192 }
193
194 void avl_del(avl **root, int data)
195 {
196     if(*root == NULL)
197     {
198         printf("There are no data that you find %d\n", data);
199         return;
200     }
201     else if((*root)->data > data)
202         avl_del(&(*root)->left, data);
203     else if((*root)->data < data)
204         avl_del(&(*root)->right, data);
205     else if((*root)->left && (*root)->right)
206         (*root)->left = find_max((*root)->left, &(*root)->data);
207     else
208     {
209         *root = chg_node(*root);
210         return;
211     }
212
213     (*root)->lev = update_level(*root);
214
215     if(abs(rotation_check(*root)) > 1)
216         *root = rotation(*root, kinds_of_rot(*root, data));
217 }
```

howard@ubuntu: ~/HomeworkBackup/data structure

```
219 int main(void)
220 {
221     avl *root = NULL;
222
223     avl_ins(&root, 50);
224     avl_ins(&root, 60);
225     avl_ins(&root, 70);
226     avl_ins(&root, 80);
227     avl_ins(&root, 90);
228     avl_ins(&root, 100);
229
230     printf("Before :\n");
231     print_tree(root);
232
233     printf("After delete 50 \n");
234     avl_del(&root,50);
235     print_tree(root);
236     return 0;
237 }
```

```
data = 70
RR Rotation
data = 90
RR Rotation
data = 100
RR Rotation
Before :
data = 80, lev = 3, left = 60, right = 90
data = 60, lev = 2, left = 50, right = 70
data = 50, lev = 1, left = NULL, right = NULL
data = 70, lev = 1, left = NULL, right = NULL
data = 90, lev = 2, left = NULL, right = 100
data = 100, lev = 1, left = NULL, right = NULL
After delete 50
data = 80, lev = 3, left = 60, right = 90
data = 60, lev = 2, left = NULL, right = 70
data = 70, lev = 1, left = NULL, right = NULL
data = 90, lev = 2, left = NULL, right = 100
data = 100, lev = 1, left = NULL, right = NULL
```

자료구조 7번

자료구조	
문제	Red Black 트리와 AVL 트리를 비교해보도록 한다.

자료구조 8번

자료구조	
문제	**난수를 활용하여 Queue 를 구현한다.**
내용	중복되는 숫자를 허용하지 않도록 프로그래밍 하시오. 제일 좋은 방법은 배열을 16 개 놓고 rand() % 16 을 해서 숫자가 겹치지 않는지 확인하면 된다.

howard@ubuntu: ~/HomeworkBackup/test

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
bool chk_dup(int* a, int i, int num);
typedef struct __queue{
    int data;
    struct __queue *link;
}queue;
queue* get_node(void);
void ins_queue(queue** head,int data);
void print_queue(queue** head);
int de_queue(queue** head);
int main(void){
    srand(time(NULL));
    int a[16];
    int i;
    queue* head = NULL;
    while(i<16){
        a[i] = rand()%16;
        if(chk_dup(a,i,a[i]))
            continue;
        ins_queue(&head,a[i]);
        i++;
    }
    print_queue(&head);
    return 0;
}
bool chk_dup(int* a, int i, int num){
    int j;
    bool chk = false;
    for(j=0;j<i;j++){
        if(a[j] == num){
            chk = true;
            break;
        }
    }
    return chk;
}
```

howard@ubuntu: ~/HomeworkBackup/test

```
40 queue* get_node(void){
41     queue* tmp;
42     tmp = (queue*)malloc(sizeof(queue)*1);
43     tmp->link = NULL;
44     return tmp;
45 }
46 void ins_queue(queue** head,int data){
47     queue** tmp = head;
48     while(*tmp)
49         tmp = &(*tmp)->link;
50     *tmp = get_node();
51     (*tmp)->data = data;
52 }
53 void print_queue(queue** head){
54     queue* tmp = *head;
55     while(tmp){
56         printf("%d\n",tmp->data);
57         tmp = tmp->link;
58     }
59 }
60 int de_queue(queue** head){
61     queue* tmp = *head;
62     int data;
63     if(!(*head)){
64         printf("Error : There's no data in queue\n");
65         return 0;
66     }
67     data = (*head)->data;
68     *head = (*head)->link;
69     free(tmp);
70     return data;
71 }
```

```
9
5
6
11
14
15
13
2
7
4
12
8
10
1
3
0
```

자료구조 9번.

자료구조	
문제	**재귀호출을 사용하여 queue 를 구현하고 10, 20 을 집어넣는다.**
내용	enqueue 과정에 대한 기계어 분석을 수행하여 동작을 파악하도록 한다. 그림과 함께 자세하게 설명하시오.

자료구조 10번.

자료구조	
문제	**[복합문제 2.1] 난수를 활용해서 Stack을 구성한다.**
내용	같은 숫자가 들어가지 않게 하고 20 개를 집어넣는다. 이때 들어가는 숫자는 1 ~ 100 사이의 숫자로 넣는다. (마찬가지로 중복되지 않게 한다)

howard@ubuntu: ~/HomeworkBackup/test

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

#define element int
typedef struct __stack{
    element data;
    struct __stack* p_node;
}stack;

stack* get_node(void);
void push(stack** top,element data);
int pop(stack** top);
void disp_stack(stack **top);
void input_stack(stack** top);
bool chk_dup(int* a,int i,int num);
int main(void){
    stack* top = NULL;
    input_stack(&top);
    disp_stack(&top);

    return 0;
}

stack* get_node(void){
    stack* tmp;
    tmp = (stack*)malloc(sizeof(stack)*1);
    tmp->p_node = NULL;
    return tmp;
}
```

howard@ubuntu: ~/HomeworkBackup/test

```c
void push(stack** top,element data){
    stack* tmp;
    if(!(*top)){
        *top = get_node();
        (*top)->data = data;
    }
    else{
        tmp = *top;
        (*top) = get_node();
        (*top)->data = data;
        (*top)->p_node = tmp;
    }
}
int pop(stack** top){
    element data;
    if(!(*top)){
        printf("there's no stack!!\n");
        return 0;
    }
    data = (*top)->data;
    *top = (*top)->p_node;
    return data;
}
void disp_stack(stack **top){
    stack* tmp = *top;
    while(tmp){
        printf("%d\n",tmp->data);
        tmp = tmp->p_node;
    }
}
```

howard@ubuntu: ~/HomeworkBackup/test

```
63 void input_stack(stack** top){
64
65     srand(time(NULL));
66     int a[20];
67     int i = 0;
68     while(i<20){
69         a[i]=rand()%100+1;
70         if(chk_dup(a,i,a[i]))
71             continue;
72         push(top,a[i]);
73         i++;
74     }
75 }
76 bool chk_dup(int* a,int i,int num){
77     int j;
78     bool chk = false;
79     for(j=0;j<i;j++){
80         if(a[j] == num){
81             chk = true;
82             break;
83         }
84     }
85     return chk;
86 }
```

```
58
14
70
28
66
53
76
96
99
16
21
25
98
71
73
100
47
90
31
19
```

자료구조 11.

자료구조	
문제	**[복합문제 2.2] 2.1 에서 만든 내용중 홀수만 빼내서 AVL 트리를 구성하도록 한다.**
내용	시험 당일 학생들에게 별도로 답안지 배포

자료구조 12번

자료구조	
문제	[복합문제 2.3] 2.1 에서 짝수만 빼내서 RB 트리를 구성하도록 한다.

자료구조 13번

자료구조	
문제	**최적화 프로세스를 기술하도록 한다.**
내용	정상적인 회사에서 프로그램을 아주 잘 최적화 하는 방법에 대해 기술하시오.

먼저, 재귀호출은 모두 반복문으로 대체한다. 재귀호출을 사용 시,
간단하게 구현할 수 있다는 장점이 있지만, 함수의 호출이 매우 많아지므로
러닝타임이 곱절정도 늘어난다.

그 외에도, 코드 내 클럭을 최소화한다. 가령, If(A&&B){} 라는 구문이 있다고 하자.
B가 거짓인 빈도수가 높다면, If(B&&A)가 더 최적화 된 코드라고 할 수 있다.

생산성보다는 성능이 압도적으로 중요한 분야라면(ex.방산, 항공 등), 고급 언어가
아닌 어셈블리어로 코딩하는 것도 최적화의 한 방안이다.

자료구조 14번.

자료구조	
문제	**기존에는 숫자만 받아봤다.**
내용	이제 Queue 에서 데이터로서 숫자 값이 아닌 문자열을 받아보도록 하자.

howard@ubuntu: ~/HomeworkBackup/test

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

typedef struct __queue
{
    char* 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, char* 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 = %s\n", head->data);
        head = head->link;
    }
}
```

howard@ubuntu: ~/HomeworkBackup/test

```
int main(void)
{
    queue *head = NULL;
    srand(time(NULL));

    enqueue(&head, "aaa");
    enqueue(&head, "bbb");

    print_queue(head);

    return 0;
}
```

```
head->data = aaa
head->data = bbb
```

자료구조 15번.

자료구조	
문제	AVL 트리에 데이터로서 숫자가 아닌 문자열을 입력하도록 프로그램하시오.

자료구조 16번.

자료구조	
문제	**Binary Tree 에 문자열을 입력한다.**
내용	기존에 Data 에 숫자로 들어오던 정보에 문자열을 입력하도록 한다.

자료구조 17번.

자료구조	
문제	**성적 관리 프로그램을 만들어보자.**
내용	여태까지 배운 학습 내용들을 활용하여 성적 관리 프로그램을 설계하고 구현해보자. 1. 통계 기능(총 합산, 평균, 표준 편차 계산) 2. 성적순 정렬 기능 3. 성적 입력 기능 4. 학생 정보 삭제 기능

howard@ubuntu: ~/HomeworkBackup/test

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

typedef struct __student{
    int index;
    int score;
    char name[20];
}student;
typedef struct __queue{
    student data;
    struct __queue* p_node;
}queue;

queue* get_node(void);
queue* enqueue(queue** top,student data);
void dequeue(queue** top,int index);
void disp_queue(queue** top);
char _getch(void);
int main(void){
    queue* top = NULL;
    char ch;
    student data;
    int index;

    while(1){

        printf("명령어 입력(quit('q'),insert('i'),display('p'),dequeue('d')) : ");
        ch = _getch();
        system("clear");
        switch(ch){
            case 'q':
                printf("quit!!\n");
                break;
            case 'i':
                printf("input(name) : ");
                scanf("%s",data.name);
                printf("input(score) : ");
                scanf("%d%*c",&data.score);
                top = enqueue(&top,data);
                printf("name : %s, score : %d inserted\n",data.name,data.score);
                break;
            case 'p':
                                                                    1,1
```

```
43                  printf("current queue :\n");
44                  disp_queue(&top);
45                  break;
46              case 'd':
47                  printf("input deleted data's index : ");
48                  scanf("%d%*c",&index);
49                  dequeue(&top,index);
50                  printf("%d index deleted!!\n",index);
51                  break;
52              default:
53                  printf("wrong instruction!!\n");
54                  break;
55          }
56          if(ch == 'q')
57              break;
58          printf("\n\n");
59      }
60      return 0;
61  }
62  queue* get_node(void){
63      queue* tmp;
64      tmp = (queue*)malloc(sizeof(queue)*1);
65      tmp->p_node = NULL;
66      return tmp;
67  }
68  queue* enqueue(queue** top,student data){
69      queue* prev;
70      queue* b_top;
71      int i = 0;
72      if(!(*top)){
73          b_top = get_node();
74          b_top->data = data;
75      }
76      else{
77          b_top = *top;
78          while(1){
79              if(*top){
80                  prev = (*top);
81                  (*top) = (*top)->p_node;
82              }
83              else{
84                  *top = get_node();
                                                    84,1-4
```

howard@ubuntu: ~/HomeworkBackup/test

```
84                  *top = get_node();
85                  (*top)->data = data;
86                  (*top)->data.index = i;
87                  prev->p_node = *top;
88                  break;
89              }
90              i++;
91          }
92      }
93      return b_top;
94  }
95  void dequeue(queue** top,int index){
96      if(!(*top)){
97          printf("there's no queue!!\n");
98      }
99      (*top)= (*top)->p_node;
100 }
101 void disp_queue(queue** top){
102     queue* tmp = *top;
103     while(tmp){
104         printf("%d. name : %s, score : %d\n",tmp->data.index,tmp->data.name,tmp->data.score);
105         tmp = tmp->p_node;
106     }
107 }
108 char _getch(void){
109     char buf,ch;
110     while((buf = getchar())!=10)
111         ch = buf;
112     return ch;
113 }
```

howard@ubuntu: ~/HomeworkBackup/test

```
current queue :
0. name : KIM, score : 100
1. name : LEE, score : 80
2. name : Choi, score : 16

명령어 입력(quit('q'),insert('i'),display('p'),dequeue('d')) :
```

임베디드 애플리케이션 분석 1번.

임베디드 애플리케이션 분석	
문제	**이것이 없으면 사실상 C 언어를 사용할 수 없다.**
내용	C 언어에서 함수를 호출하기 위해서도 이것은 반드시 필요하다. 이와 같은 이유로 운영체제의 부팅 코드에서도이것을 설정하는 작업이 진행되는데 이것은 무엇일까 ?

헤더파일 (stdio.h stdlib.h 등..)

임베디드 애플리케이션 분석 2번.

임베디드 애플리케이션 분석 3번.

임베디드 애플리케이션 분석	
문제	**12 비트 ADC 를 가지고 있는 장치가 있다.**
내용	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. 모든 문제는 기능별로 함수를 분리해야함 (통 메인, 통 함수 전부 감점임)

howard@ubuntu: ~/HomeworkBackup/test

```c
#include <stdio.h>
double dac(int sig);
int main(void){
    int sig;
    double v;
    sig = 2077; // ADC value : 2077
    v = dac(sig);

    printf("adc : %d, voltage : %.3lf\n",sig,v);
}
double dac(int sig){
    int minV = -5, maxV = 5, bit = 12;
    int maxSig = 2047; //-2^11 ~ 2^11-1
    int minSig = -2048;
    double v;

    v = (double)maxV/maxSig*sig;
    return v;

}
```

```
adc : 2077, voltage : 5.073
howard@ubuntu:~/HomeworkBackup/test$
```

임베디드 애플리케이션 분석 4번.

임베디드 애플리케이션 분석	
문제	**goto 의 정체성은 ?**
내용	전세계 각지의 천재들이 모여서 개발하는 리눅스 운영체제 코드에는 엄청 많은 양의 goto 가 사용되고 있다. goto 를 도대체 왜 사용해야만 할까 ?

다중 루프를 생각해보자

```
while(1){
    while(1){
    }
}
```

이 루프를 빠져나오기 위해서는, break문을 루프 수만큼 써야한다.

goto를 이용하면, 1번에 모든 루프를 빠져나올 수 있다.

프로그래밍하기 편할 뿐만 아니라, 어셈블리어 jmp문의 사용이 줄어들어

파이프라인이 덜 깨지기 때문에, 성능면에서도 뛰어나다.

임베디드 애플리케이션 분석 5번.

임베디드 애플리케이션 분석	
문제	**포인터 크기에 대해 알고 있는대로 기술하시오.**

포인터는 주소를 저장하는 메모리 공간이다.
저장하는 변수가 주소이기 때문에, 데이터 타입에 관계 없이 포인터의 크기는 모두 같다.
운영체제에 따라 크기가 달라진다.
32bit os에서는 4byte의 크기를, 64bit os에서는 8byte의 크기를 갖는다.
확인 방법은 printf("%d",sizeof(*p));

임베디드 애플리케이션 분석 6번.

임베디드 애플리케이션 분석	
문제	**위의 문장에서 Safety MCU 가 몇 번째 부터 시작하는지 찾아내도록 프로그래밍 해보자.**
내용	TI Cortex-R5F Safety MCU is very good to Real-Time System. (이정도는 가볍게 해야 파싱 같은것도 쉽게 할 수 있다) 무능력자가 되지 않기 위해서라도 이정도는 가볍게 할 수 있어야 한다. 모든 문제는 기능별로 함수를 분리해야함 (통 메인, 통 함수 전부 감점임)

howard@ubuntu: ~/HomeworkBackup/test

```
#include <stdio.h>
#include <string.h>
int compStr(char* str,char* search);
int main(void){
    char* str = "TI Cortex-R5F Safety MCU is very good to Real-Time System.";
    char* search = "Safety MCU";
    int index;
    index = compStr(str,search);
    if(index == -1)
        printf("일치하는 문장 없음\n");
    else
        printf("%d번째부터 시작\n",index+1);
}
int compStr(char* str,char* search){
    int index = -1, len, chk;
    for(index = 0;index < len; index++){
        chk = strncmp(str+index,search,strlen(search));
        if(!chk)
            break;
    }
    return index;

}
```

```
15번째부터 시작
howard@ubuntu:~/HomeworkBackup/test$
```

임베디드 애플리케이션 분석 7번.

임베디드 애플리케이션 분석	
문제	**이중 배열을 함수의 인자로 입력 받아 3 by 3 행렬의 곱셈을 구현하시오.**
내용	모든 문제는 기능별로 함수를 분리해야함 (통 메인, 통 함수 전부 감점임)

howard@ubuntu: ~/HomeworkBackup/test

```c
#include <stdio.h>
void multiple(int (*mat1)[3],int (*mat2)[3],int (*res)[3]);
void print_mat(int (*mat)[3]);
int main(void){
    int mat1[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int mat2[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int res[3][3];
    multiple(mat1,mat2,res);
    print_mat(mat1);
    printf("\n\n");
    print_mat(mat2);
    printf("\n\n");
    print_mat(res);
    return 0;
}
void multiple(int (*mat1)[3],int (*mat2)[3],int (*res)[3]){
    int i = 0, j = 0, k = 0;
    for(i = 0;i < 3;i++){
        for(j = 0;j < 3;j++){
            res[i][j] = 0;
            for(k = 0;k < 3;k++){
                res[i][j] += mat1[i][k] + mat2[k][j];
            }
        }
    }
}
void print_mat(int (*mat)[3]){
    int i = 0, j = 0;
    for(i = 0;i < 3; i++){
        for(j = 0;j < 3; j++){
            printf("%4d",mat[i][j]);
        }
        printf("\n");
    }
}
```

```
   1   2   3
   4   5   6
   7   8   9


   1   2   3
   4   5   6
   7   8   9


  18  21  24
  27  30  33
  36  39  42
```

임베디드 애플리케이션 분석 8번.

임베디드 애플리케이션 분석	
문제	**다음 프로토타입을 기술하시오.**
내용	void (* signal(int signum, void (* handler)(int)))(int)

return : void (*)(int)
name : signal
parameter : int, void(*)(int)

임베디드 애플리케이션 9번.

임베디드 애플리케이션 분석

문제	**다음 코드를 작성하라.**
내용	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) 모든 문제는 기능별로 함수를 분리해야함 (통 메인, 통 함수 전부 감점임)

howard@ubuntu: ~/HomeworkBackup/test

```c
#include <stdio.h>
//parameter : void
//name : fp_1
//return : void
void fp_1(void){
    printf("fp_1 called!\n");
}


//parameter : void(*)(void)
//name : fp_2
//return : void
void fp_2(void (*p)(void)){

    printf("fp_2 called!\n");
    p();
}


//parameter : int, int
//name : fp_main
//return : void (*)(void(*)(void))
void (*fp_main(int x,int y))(void(*)(void)){

    printf("fp_main called!\n");
    return fp_2;
}

int main(void){
    fp_main(1,2)(fp_1);
}

```

```
fp_main called!
fp_2 called!
fp_1 called!
```

임베디드 애플리케이셔 분석 10번.

임베디드 애플리케이션 분석	
문제	파이프라인이 깨지는 경우 어떠한 이점이 있는지 기술하시오.

결론부터 말하면, 이점이 전혀 없다.
가장 간단한 3단계 파이프라인을 살펴보자
fetch discode execute
jmp나 callq 가 execute가 되면, fetch와 discode중이던 명령어는 깨지고 처음부터 실행된다. 이는 2clock만큼의 성능 손실이 있다.
그렇기 때문에, callq가 기하급수적으로 늘어나는 재귀호출 형태는 되도록 사용하지 않는 것이 좋다.

임베디드 애플리케이션 분석 11번.

임베디드 애플리케이션 분석	
문제	**아래 프로그램을 작성한다.**
내용	4x^2 + 5x + 1 을 1 ~ 3 까지 정적분하는 프로그램을 구현하도록 한다. 모든 문제는 기능별로 함수를 분리해야함 (통 메인, 통 함수 전부 감점임)

howard@ubuntu: ~/HomeworkBackup/test

```c
#include <stdio.h>
int f_x(int x);
double integral(int x1,int x2);
int main(void){
    double res;
    res = integral(1,3);
    printf("%.5lf\n",res);
    return 0;
}
int f_x(int x){
    int y;
    y = 4*x*x + 5*x + 1; //4x^2+5x+1

    return y;
}
double integral(int x1,int x2){
    int n = 100000, i;
    double dx = (double)(x2 - x1)/n;
    double res = 0, x = x1;
    for(i = x1; x <= x2; i++){
        res+= f_x(x)*dx;
        x = x + dx;
    }
    return res;
}
```

```
36.99966
```

임베디드 애플리케이션 12번.

임베디드 애플리케이션 분석	
문제	**아래 프로그램을 작성하시오.**
내용	함수 포인터를 활용하여 float 형 3 by 3 행렬의 덧셈과 int 형 3 by 3 행렬의 덧셈을 구현하도록 하시오. 모든 문제는 기능별로 함수를 분리해야함 (통 메인, 통 함수 전부 감점임)

임베디드 애플리케이션 분석 13번.

임베디드 애플리케이션 분석	
문제	**아래 설명을 보고 프로그래밍 하시오.**
내용	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. 모든 문제는 기능별로 함수를 분리해야함 (통 메인, 통 함수 전부 감점임)

howard@ubuntu: ~/HomeworkBackup/test

```c
#include <stdio.h>
int fib(int num);
int main(void){
    int num = 27, i = 1;
    int val, odd_sum = 0,even_sum = 0;
    while(i<=num){
        val = fib(i);
        if(val % 2)
            odd_sum += val;
        else
            even_sum += val;
        i++;
    }
    printf("odd_sum : %d\n",odd_sum);
    printf("even_sum : %d\n",even_sum);
    printf("odd_sum - even_sum = %d\n",odd_sum-even_sum);
    return 0;
}
int fib(int num){
    int num1 = 1,num2 = 1;
    int res = 0;
    int n = 2;
    if(num == 1 || num == 2)
        return 1;
    while(n!=num){
        res = num1 + num2;
        num1 = num2;
        num2 = res;
        n++;
    }
    return res;
}
```

```
odd_sum : 257114
even_sum : 257114
odd_sum - even_sum = 0
```

임베디드 애플리케이션 분석 14번.

임베디드 애플리케이션 분석	
문제	**아래 설명을 보고 프로그래밍 하시오.**
내용	1, 4, 5, 9, 14, 23, 37, 60, 97, ... 형태로 숫자가 진행된다. 23번째 숫자는 무엇일까 ? (프로그래밍 하시오)

howard@ubuntu: ~/HomeworkBackup/test

```c
#include <stdio.h>
int fib(int num);
int main(void){
    int num = 23, i = 1;
    int val;
    while(i<=num){
        val = fib(i);
        i++;
    }
    printf("%d번째 : %d\n",i-1,val);
    return 0;
}
int fib(int num){
    int num1 = 1,num2 = 4, num3 = 5;
    int res = 0;
    int n = 2;
    if(num == 1)
        return 1;
    else if(num == 2)
        return 4;
    while(n!=num){
        res = num1 + num2;
        num1 = num2;
        num2 = res;
        n++;
    }
    return res;
}
```

```
23번째 : 81790
```

임베디드 애플리케이션 분석 15번.

임베디드 애플리케이션 분석	
문제	Intel Architecture와 ARM Architecture의 차이점은 ?

1.Data Ordering 방식 :
Intel Architecture는 Little Endian 방식을 사용하고, ARM Architecture는 선택 옵션이 있다.
2.용도 :
Intel 프로세서는 주로 pc에 사용하고, Arm 프로세서는 모바일 용으로 주로 사용한다.

임베디드 애플리케이션 분석 16번.

임베디드 애플리케이션 분석

문제	**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.

howard@ubuntu: ~/HomeworkBackup/test

```
 2 #include <stdlib.h>
 3 #include <time.h>
 4 #include <stdbool.h>
 5 int* sadari(int people,int getnum);
 6 bool chk_dup(int* a, int i, int num);
 7 int main(void){
 8     int people, getnum, i; //people 참여인원수, getnum 사다리 당첨자수
 9     int* get;
10     printf("사람수 : ");
11     scanf("%d",&people);
12     printf("사다리 당첨자 수  : ");
13     scanf("%d",&getnum);
14     get = sadari(people,getnum);
15     //당첨자 출력
16     printf("%d명 당첨!\n",getnum);
17     for(i=0;i<getnum;i++)
18         printf("%d번 사람\n",get[i]);
19
20     return 0;
21 }
22 int* sadari(int people,int getnum){
23     int* get;                //당첨자 저장 변수
24     int i = 0;
25     get = (int*)malloc(sizeof(int)*getnum);
26     srand(time(NULL));
27     while(i<getnum){
28         get[i] = rand()%people + 1;
29         if(chk_dup(get,i,get[i]))
30             continue;
31         i++;
32     }
33     return get;
34
35 }
36 bool chk_dup(int* a,int i, int num){
37     int j;
38     bool chk = false;
39     for(j = 0; j < i; j++){
40         if(a[j] == num){
41             chk = true;
42             break;
43         }
44     }
45     return chk;
46 }
```

```
사람수 : 25
사다리 당첨자 수  : 5
5명 당첨!
13번 사람
1번 사람
18번 사람
9번 사람
22번 사람
```

임베디드 애플리케이션 분석 17번.

임베디드 애플리케이션 분석	
문제	**아래 설명을 보고 프로그래밍 해보자.**
내용	아래와 같은 행렬을 생각해보자! 2 4 6 2 4 6 sapply(arr, func) 으로 위의 행렬을 아래와 같이 바꿔보자! 2 4 6 4 16 36 sapply 함수를 위와 같이 구현하라는 의미다. (R 이나 python 같은 언어에서 지원되는 기법중 하나에 해당한다) 모든 문제는 기능별로 함수를 분리해야함 (통 메인, 통 함수 전부 감점임)

howard@ubuntu: ~/HomeworkBackup/test

```c
#include <stdio.h>
void func(int (*arr)[3]);
void sapply(int (*arr)[3],void(*p)(int (*array)[3]));
void print_mat(int (*mat)[3]);
int main(void){
    int arr[2][3] = {{2,4,6},{2,4,6}};
    printf("before : \n");
    print_mat(arr);
    sapply(arr,func);
    printf("after : \n");
    print_mat(arr);
}
void print_mat(int (*mat)[3]){
    int i = 0, j = 0;
    for(i = 0;i < 2;i++){
        for(j=0;j<3;j++){
            printf("%4d",mat[i][j]);
        }
        printf("\n");
    }

}
void func(int (*arr)[3]){
    int j;
    for(j=0;j<3;j++){
        arr[1][j] = arr[1][j] * arr[1][j];
    }
}


void sapply(int (*arr)[3],void(*p)(int (*array)[3])){
    p(arr);
}
```

```
before :
   2   4   6
   2   4   6
after :
   2   4   6
   4  16  36
```

임베디드 애플리케이션 18번.

임베디드 애플리케이션 분석

문제	**아래 설명을 보고 프로그램을 만들어보자.**
내용	char *str = "WTF, Where is my Pointer ? Where is it ?" 라는 문자열이 있다. 여기에 소문자가 총 몇 개 사용되었는지 세는 프로그램을 만들어보자! 모든 문제는 기능별로 함수를 분리해야함 (통 메인, 통 함수 전부 감점임)

howard@ubuntu: ~/HomeworkBackup/test

```c
#include <stdio.h>
#include <string.h>
int cntSmallLetter(char* str);
int main(void){
    char* str = "WTF, Where is my Pointer ? Where is it ?";
    int cnt;
    cnt = cntSmallLetter(str);
    printf("Small Letter : %d(EA)\n",cnt);

}
int cntSmallLetter(char* str){
    int len = strlen(str);
    int i,cnt = 0;
    for(i = 0;i < len;i++){
        if((str[i] > 96)&&(str[i] < 123 ))
            cnt++;
    }
    return cnt;
}
```

```
Small Letter : 22(EA)
```

임베디드 애플리케이션 분석 19번.

임베디드 애플리케이션 분석	
문제	int *p[3] 와 int (*p)[3] 는 같은 것일까 ? 다른 것일까 ? 이유를 함께 기술하도록 한다.

다른 것이다.
int *p[3]는 포인터 배열이고, int (*p)[3]은 배열 포인터이다.
포인터 배열은 주소를 저장하는 메모리 공간을 여러 개 할당하는 것이다.
배열 포인터는 배열 크기만큼의 주소를 저장하는 포인터이다.
따라서, 2중 배열 등을 인자로 받아 올 때, 배열 포인터를 사용한다.

임베디드 애플리케이션 분석 20번.

임베디드 애플리케이션 분석	
문제	**아래 설명을 보고 프로그램을 작성한다.**
내용	임의의 값 x가 있는데, 이를 134217728 단위로 정렬하고 싶다면 어떻게 해야할까 ? 어떤 숫자를 넣던 134217728 의 배수로 정렬이 된다는 뜻임 (힌트 : 134217728 = 2^27)

howard@ubuntu: ~/HomeworkBackup/test

```
1 #include <stdio.h>
2
3 int main(void){
4     int x;
5     printf("input x: ");
6     scanf("%d",&x);
7     printf("arranged value (134217728) : %d\n",x&(-134217728));
8 }
```

```
input x: 10000000000000
arranged value (134217728) : 1207959552
```

임베디드 애플리케이션 분석 21번.

임베디드 애플리케이션 분석	
문제	**아래 설명을 보고 프로그램을 작성한다.**
내용	단 한 번의 연산으로 대소문자 전환을 할 수 있는 연산에 대해 기술하시오. 덧셈 혹은 뺄셈 같은 기능이 아님 모든 문제는 기능별로 함수를 분리해야함 (통 메인, 통 함수 전부 감점임)

howard@ubuntu: ~/HomeworkBackup/test

```
1 #include <stdio.h>
2 int main(void){
3     char ch;
4     printf("input 1 letter(char) : ");
5     scanf("%c",&ch);
6     printf("res : %c\n",ch^32);
7 }
```

```
input 1 letter(char) : a
res : A
```

임베디드 애플리케이션 22번.

임베디드 애플리케이션 분석	
문제	**변수의 정의를 기술하시오.**

변수란, 데이터를 저장하는 메모리 공간을 말한다.
int형 변수의 경우 정수형 데이터를 저장하는 메모리 공간이고,
int*형 변수의 경우, int형 데이터의 주소를 저장하는 메모리 공간이 된다.

임베디드 애플리케이션 23번.

임베디드 애플리케이션 분석	
문제	포인터의 정의를 기술하시오.

변수란, 데이터를 저장하는 메모리 공간이라면,
포인터는 주소를 저장하는 메모리 공간이다.
주소를 저장하기 때문에, data type에 관계 없이 그 크기가 일정하다.(OS에 따른다)

임베디드 애플리케이션 24번.

임베디드 애플리케이션 분석

문제	**함수 포인터의 정의를 기술하시오.**

모든 변수가 주소를 가지고 있듯이, 함수 또한 주소가 있다.
함수의 이름이 주소가 된다.
이러한 함수의 주소를 저장하는 메모리 공간이 함수 포인터이다.

임베디드 애플리케이션 25번

함수를 호출하거나 건너뛰는 코드(if문 등)을 사용할 때, 어셈블리어를 살펴보면 callq나 jmp je 등이 있다, 이 때 파이프라인이 깨진다.

임베디드 애플리케이션 분석 26번.

임베디드 애플리케이션 분석	
문제	메모리 계층 구조에 대해 기술하시오.

임베디드 애플리케이션 분석 27번.

임베디드 애플리케이션 분석	
문제	C 언어의 기본 메모리 구조에 대해 기술하시오.

기본적으로, c언어의 모든 동작은 스택 메모리 내에서 이루어진다.
스택 메모리 중에서도, 시스템 영역을 제외한 유저 영역에서 이루어진다.
동적 할당의 경우, Heap메모리에서 배열을 할당하여, 그 주소를 반환해준다.
예외적으로 시스템 영역의 메모리를 사용할 때가 있다.
문자열을 선언하면, 문자열은 시스템 영역에서 할당이 되고, 그 주소를 반환한다.

임베디드 애플리케이션 분석 28번.

임베디드 애플리케이션 분석	
문제	**디버깅하면서 보는 변수의 주소는 무엇일까 ?**
내용	우리가 사용하는 윈도우나 리눅스, 맥(유닉스)에서 보는 변수 주소는 진짜 주소일까 아닐까 ? 알고 있는대로 기술하시오.

우리가 사용하거나 보는 모든 주소는, 가상의 주소이다.
사용하는 메모리 자체가 가상 메모리이기 때문이다.
페이지 테이블을 사용하여, 접근해야 하는 물리 주소에 접근한다.
각각의 테이지 테이블은 서로 독립적이어서, 다른 프로그램에 의한 간섭을 차단하는 효과가 있다.

임베디드 애플리케이션 분석 29번

임베디드 애플리케이션 분석	
문제	**아래 설명을 보고 프로그램을 작성하시오.**
내용	이름과 급여를 저장하도록 만든다. 이름은 마음대로 고정시키도록 하고 급여는 rand() 를 활용 급여의 평균을 출력하고 가장 높은 한 사람의 이름과 급여를 출력하시오. (값이 같을 수 있음에 유의해야 한다) 모든 문제는 기능별로 함수를 분리해야함 (통 메인, 통 함수 전부 감점임)

howard@ubuntu: ~/HomeworkBackup/test

```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
typedef struct __employee{
    char* name;
    int salary;
}employee;
void init_salary(employee* p);
double avr_salary(employee *p);
void print_best(employee *p);
int main(void){
    employee p[5]={{"Mark",},{"Jason",},{"Rena",},{"Jay",},{"Sheldon",}};
    int i;
    double avr;
    srand(time(NULL));

    init_salary(p);
    for(i=0;i<5;i++)
        printf("%s, %d\n",p[i].name,p[i].salary);
    avr = avr_salary(p);
    printf("average : %.2lf\n",avr);
    print_best(p);
    return 0;
}
void init_salary(employee* p){
    int i;
    for(i=0;i<5;i++)
        p[i].salary = rand()%5000000;
}
double avr_salary(employee *p){
    double res = 0;
    int i;
    for(i = 0;i<5;i++)
        res += p[i].salary;
    res /= 5;
    return res;
}
void print_best(employee *p){
    int tmp = 0,i, index;
    for(i=0;i<5;i++){
        if(tmp < p[i].salary){
            tmp = p[i].salary;
            index = i;
        }
    }
    printf("The best salaryman : %s, %d\n",p[index].name,p[index].salary);
```

```
Mark, 393520
Jason, 2354526
Rena, 3714020
Jay, 4918855
Sheldon, 152541
average : 2306692.40
The best salaryman : Jay, 4918855
```

임베디드 애플리케이션 분석 30번.

임베디드 애플리케이션 분석	
문제	**리눅스에서 디버깅을 수행하기 위한 프로그램 컴파일 방법을 기술하시오.**

먼저, c 파일에 디버깅 옵션을 주어 컴파일한다.
gcc -g -o debug 1.c
break point를 main에 주고, 실행한다.
s,n,si,ni 등의 명령어를 통해 프로그램을 1단계씩 실행한다.
s,si 명령어는 함수가 있을 때, 함수 안으로 들어가기 때문에,
printf 등 디버깅할 필요가 없는 함수는 n,ni를 통해 건너뛴다.
p/x 나 x 명령어 등으로 데이터나 주소값을 확인한다.
오동작을 점검한다.

임베디드 애플리케이션 분석 31번

임베디드 애플리케이션 분석	
문제	**vi 활용 방법**
내용	vi 에서 코드가 정렬이 잘 안되어 있다. 이 상황에서 어떻게 해야 예쁘고 깔끔하게 정렬되는가 ?

편집 모드에서 v를 누르면, 범위를 선택할 수 있다.
정렬하고자 하는 범위를 선택한 후에 =을 누르면 예쁘게 정렬된다.

임베디드 애플리케이션 32번.

임베디드 애플리케이션 분석	
문제	**최적화 기법에 대한 문제다.**
내용	프로그램을 최적화하는 컴파일 옵션을 적고 반대로 어떠한 최적화도 수행하지 않는 컴파일 옵션을 적도록 한다.

-O, -O1, -O2, -O3, -Os 등의 옵션을 넣으면, 컴파일 시 최적화를 수행한다.
최적화를 원하지 않을 경우(코드 분석 등의 이유로) -O0 옵션을 넣으면 된다.

임베디드 애플리케이션 33번

임베디드 애플리케이션 분석	
문제	**gdb 를 사용하는 이유를 기술하시오.**

segmentation fault 등의 에러가 났을 경우,
어느 부분에서 문제가 있는 지 파악할 수 있다.
또한, 동작은 잘 되나 결과값이 원하지 않는 결과가 나올 때,
한 단계씩 실행해보면 잘못된 논리를 찾아낼 수 있다.

임베디드 애플리케이션 분석 34번.

임베디드 애플리케이션 분석	
문제	**기계어 레벨에서 스택을 조정하는 명령어는 어떤것들이 있는가 ?**
내용	직접 연산을 하는 경우를 제외하고 기술하도록 한다. (4 가지)

push, mov, jmp, callq 등.

임베디드 애플리케이션 분석 35번.

임베디드 애플리케이션 분석	
문제	**a 좌표(3, 6), b 좌표(4, 4) 가 주어졌다.**
내용	원점으로부터 이 두 좌표가 만들어내는 삼각형의 크기를 구하는 프로그램을 작성하라.

임베디드 애플리케이션 분석 36번.

임베디드 애플리케이션 분석	
문제	**가위 바위 보 게임을 만들어보자.**
내용	프로그램을 만들고 컴퓨터랑 배틀 뜨자!

howard@ubuntu: ~/HomeworkBackup/test

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int game(int com,int player);
int main(void){
    srand(time(NULL));
    int com, player,chk;
    while(1){
        printf("가위 바위 보!! (1:가위, 2:바위, 3:보, 그 외 종료) : ");
        scanf("%d",&player);
        com = rand()%3 + 1;
        chk = game(com,player);
        if(chk == 1)
            break;
    }
    printf("게임 종료!\n");
    return 0;
}
int game(int com,int player){

    int res = player - com;
    int chk = 0;
    if(res == 1 || res == -2)
        printf("You win\n");
    else if(res == 0)
        printf("Draw\n");
    else if(res == -1|| res == 2)
        printf("You lose\n");
    switch(com){
        case 1:
            printf("com : 가위\n");
            break;
        case 2:
            printf("com : 바위\n");
            break;
        case 3:
            printf("com : 보\n");
            break;
        default:
            chk = 1;
            break;
    }
    return ckk;
```

```
com : 가위
가위 바위 보!! (1:가위, 2:바위, 3:보, 그 외 종료) : 2
Draw
com : 바위
가위 바위 보!! (1:가위, 2:바위, 3:보, 그 외 종료) : 2
Draw
com : 바위
가위 바위 보!! (1:가위, 2:바위, 3:보, 그 외 종료) : 2
Draw
com : 바위
가위 바위 보!! (1:가위, 2:바위, 3:보, 그 외 종료) : 2
You win
com : 가위
가위 바위 보!! (1:가위, 2:바위, 3:보, 그 외 종료) : 2
You lose
com : 보
가위 바위 보!! (1:가위, 2:바위, 3:보, 그 외 종료) : 2
Draw
com : 바위
가위 바위 보!! (1:가위, 2:바위, 3:보, 그 외 종료) : 2
You lose
com : 보
가위 바위 보!! (1:가위, 2:바위, 3:보, 그 외 종료) : 2
Draw
com : 바위
```