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

강사 – Innova Lee (이상훈)
gcccompil3r@gmail.com
학생-김민주
alswnqodrl@naver.com

01

[복합문제 1.1] 값이 1 ~ 4096까지 무작위로 할당되어 배열에 저장되도록 프로그래밍 하시오.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define randomize() srand((unsigned)time(NULL))
#define random(num) (rand()%(num))

int main()
{
     int data[100];
     int i, j, temp;

      randomize(); // 랜덤 초기화

     for(i=0; i < 100; i++)
          data[i] = random(4096);  // 랜덤함수 사용해서 데이터 생성


     // 소트하기전 데이터 출력

     printf("Before sort : ");
     for(i=0; i < 100; i++)
          printf("%-3d", data[i]);
     printf("\n");


     // 버블소트 사용하여 정렬
     for(i=0; i < 99; i++)
     {
          for(j=1; j < 100-i; j++)
          {
               if(data[j-1] > data[j])
               {
                    temp = data[j-1];
                    data[j-1] = data[j];
                    data[j] = temp;
               }
          }
     }

     // 소트 후 데이터 출력

     printf("After sort  : ");
     for(i=0; i < 100; i++)
          printf("%-3d", data[i]);
     printf("\n");



     return 0;
}
```

```
alswnqodrl@alswnqodrl-900X3K:~$ ./a.out
Before sort : 40512305352218924107733376516138887892714582418104339598089916384157042922641368111626182638318621906845881531639289395725313303173018113820311826906514811012755387619931654141820342225184720235932964282121352054915282026422446345914393404189564610383706370610612300102254233131777322121034321740324415613587344710582455217231934133087191730551438128139874631761045178427867878899915927957980101210221038104510581116121012811368138814181438143914581531156116381654173017401777178418111847189218951917199320342054213521722190222522642300230524182446245525312618263826422690278628202821289329643055308731183176318631933244330333133376339534043432344734593522358735933706382038764051alswnqodrl@alswnqodrl-900X3K:~$ vi data1.calswnqodrl@alswnqodrl-900X3K:~$ vi data1.calswnqodrl@alswnqodrl-900X3K:~$
```

02

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.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define randomize() srand((unsigned)time(NULL))
#define random(num) (rand()%(num))

int main()
{
     int data[100];
     int i, j, temp;

      randomize(); // 랜덤 초기화

     for(i=0; i < 100; i++)
          data[i] = random(4096);  // 랜덤함수 사용해서 데이터 생성


     // 소트하기전 데이터 출력

     printf("Before sort : ");
     for(i=0; i < 100; i++)
          printf("%-3d", data[i]);
     printf("\n");


     // 버블소트 사용하여 정렬
     for(i=0; i < 99; i++)
     {
          for(j=1; j < 100-i; j++)
          {
               if(data[j-1] > data[j])
               {
                    temp = data[j-1];
                    data[j-1] = data[j];
                    data[j] = temp;
               }
          }
     }

     // 소트 후 데이터 출력

     printf("After sort  : ");
     for(i=0; i < 100; i++)
          printf("%-3d", data[i]);
     printf("\n");


     return 0;
}
```

```
alswnqodrl@alswnqodrl-900X3K:~$ ./a.out
Before sort : 405123053522189241077333765161388878927145824181043395980899163841
 5704292264136811162618263831862190684588153163928939572531330317301811382031182
69065148110127553876199316541418203422251847202359329642821213520549152820264224
4634591439340418956461038370637061 230010225423313177732212103432174032441561358
734471058245521723193413308719173055143812813987463176104517842786
After sort  : 41 61 104202322370398410413429481516542570588639646651684746755773
87889991592795798010121022103810451058111612101281136813881418143814391458153115
61163816541730174017771784181118471892189519171993203420542135217221902225226423
00230524182446245525312618263826422690278628202821289329643055308731183176318631
933244330333133376339534043432344734593522358735933706382038764051
alswnqodrl@alswnqodrl-900X3K:~$ vi data1.c
alswnqodrl@alswnqodrl-900X3K:~$ vi data1.c
alswnqodrl@alswnqodrl-900X3K:~$
```

03

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.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define randomize() srand((unsigned)time(NULL))
#define random(num) (rand()%(num))

int main()
{
     int data[100];
     int i, j, temp;

      randomize(); // 랜덤 초기화

     for(i=0; i < 100; i++)
          data[i] = random(131072);  // 랜덤함수 사용해서 데이터 생성


     // 소트하기전 데이터 출력

     printf("Before sort : ");
     for(i=0; i < 100; i++)
          printf("%-3d", data[i]);
     printf("\n");


     // 버블소트 사용하여 정렬
     for(i=0; i < 99; i++)
     {
          for(j=1; j < 100-i; j++)
          {
               if(data[j-1] > data[j])
               {
                    temp = data[j-1];
                    data[j-1] = data[j];
                    data[j] = temp;
               }
          }
     }

     // 소트 후 데이터 출력

     printf("After sort  : ");
     for(i=0; i < 100; i++)
          printf("%-3d", data[i]);
     printf("\n");



     return 0;
}
```

```
alswnqodrl@alswnqodrl-900X3K:~$ gcc data1.c
alswnqodrl@alswnqodrl-900X3K:~$ ./a.out
Before sort : 95641108717716871078024780177532705982712912472997412107620866056653231793117803104872110568115242955791211519090601108056530231254061100792421610252651522112191774451609189836180601238946565955926342033695892502976010243447839629242036315147009321532430399965022445271575917913050180180535131095091043972496729959855161024134605144281120473388735084684994102293845414317298194785879559727457491194703629527282314374873497275859464052491567702117565275931120991146157553665441138741036041108251032751140530599571971136983Before sort : 95641108717716871078024780177532705982712912472997412107620866056653231793117803104872110568115242955791211519090601108056530231254061100792421610252651522112191774451609189836180601238946565955926342033695892502976010243447839629242036315147009321532430399965022445271575917913050180180535131095091043972496729959855161024134605144281120473388735084684994102293845414317298194785879559727457491194703629527282314374873497275859464052491567702117565275931120991146157553665441138741036041108251032751140530599571971136983After sort  : 91298148735749656577021024311405114611152416091180602153222445242162496727129271572728227593295572976029959305993143731514317933369536295388734203643039431724428144783460514780149156497275084651522530235351357197575535859459179634206405266532665447009370598716877445775328018084541849948551686605879558925089836906019478595592956419629297274974129965010229310241310252610327510360410439710487210762010780210805610871710950911007911056811082511209911219111369811387411519011756511780311947012047312389412472912540613050 1
alswnqodrl@alswnqodrl-900X3K:~$
```

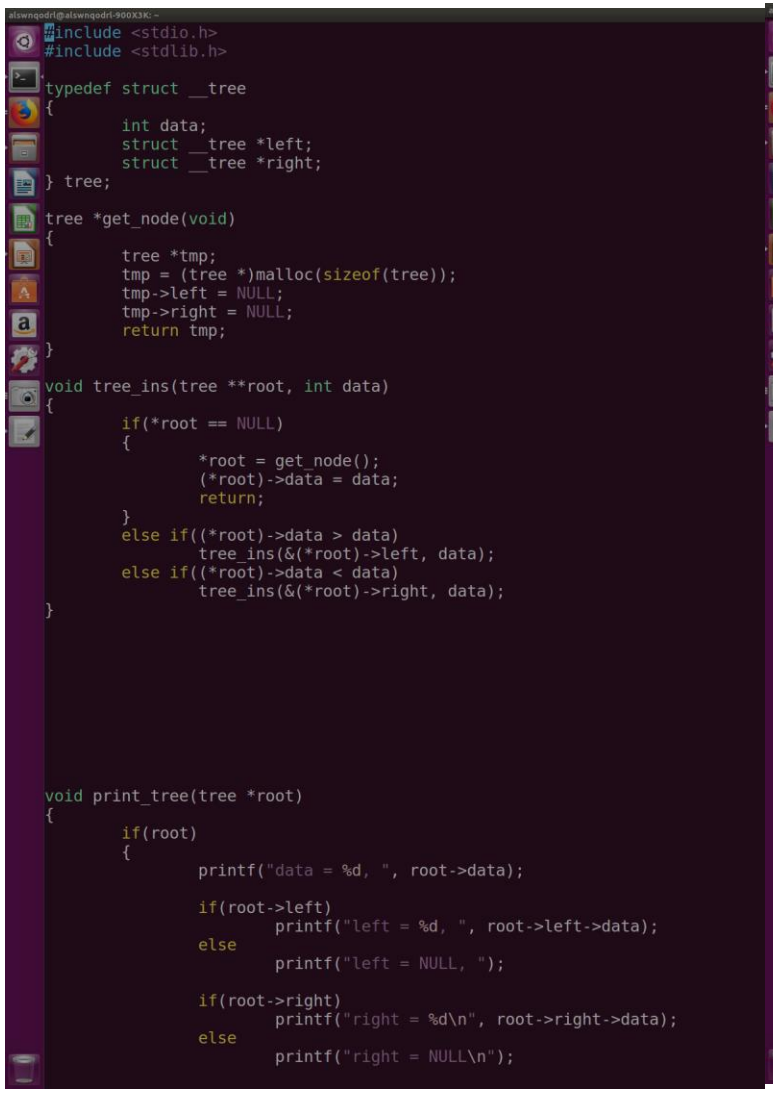
04

이진 트리를 재귀 호출을 사용하여 구현하도록 한다.

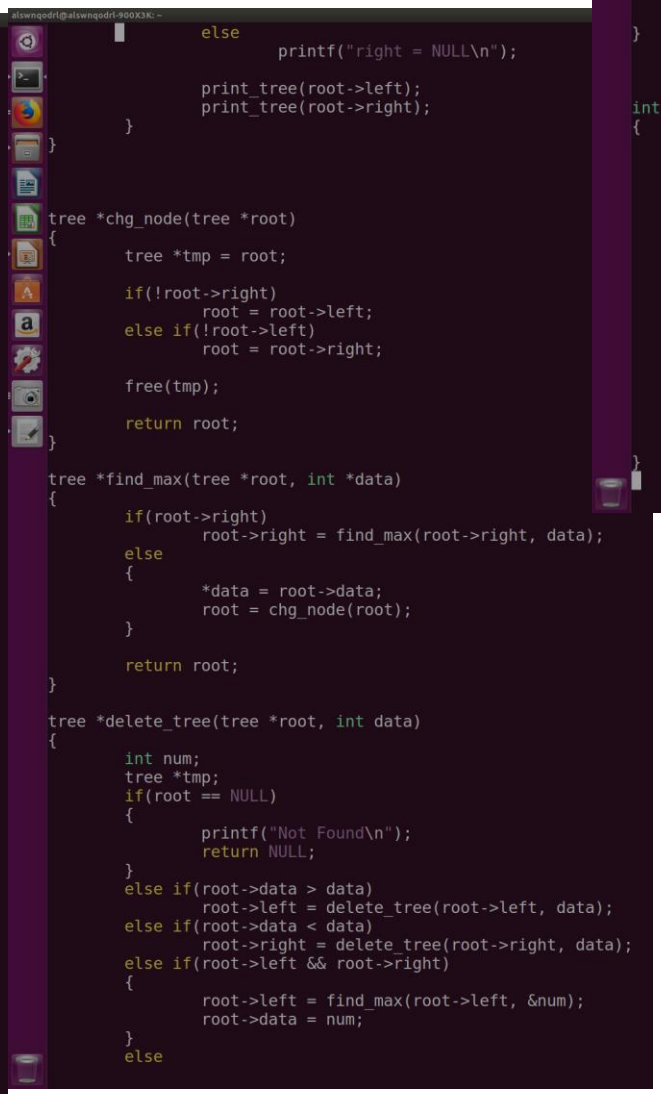
```
  ^
data4.c:131:2: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
alswnqodrl@alswnqodrl-900X3K:~$ vi data4.c
alswnqodrl@alswnqodrl-900X3K:~$ gcc data4.c
alswnqodrl@alswnqodrl-900X3K:~$ ./a.out
data = 50, left = 45, right = 73
data = 45, left = 32, right = 48
data = 32, left = 16, 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
alswnqodrl@alswnqodrl-900X3K:~$
```

04

이진 트리를 재귀 호출을 사용하여 구현하도록 한다.



```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);

		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");
```



```c
		else
			printf("right = NULL\n");

		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
```

```c
		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;
}
```

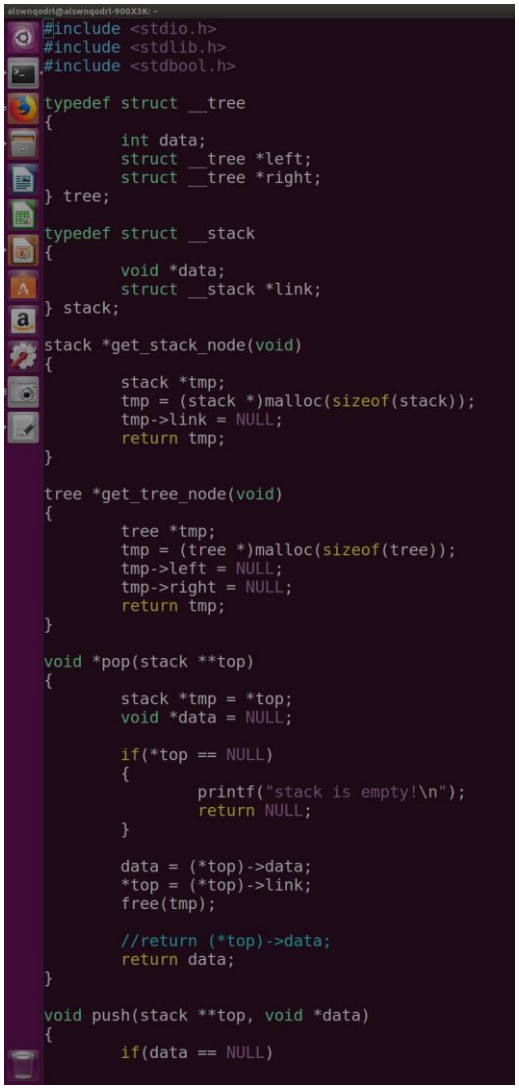
05

이진 트리를 재귀 호출 없이 구현하도록 한다.

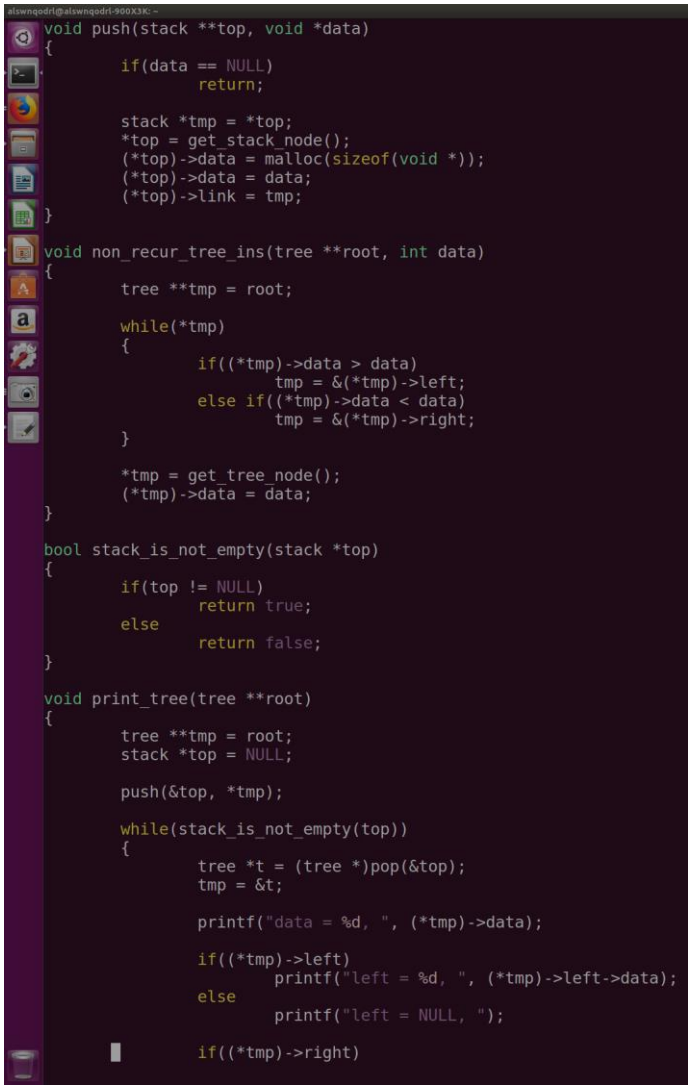
```
alswnqodrl@alswnqodrl-900X3K:~$ vi data4.c
alswnqodrl@alswnqodrl-900X3K:~$ vi data4.c
alswnqodrl@alswnqodrl-900X3K:~$ vi data5.c
alswnqodrl@alswnqodrl-900X3K:~$ gcc data5.c
alswnqodrl@alswnqodrl-900X3K:~$ ./a.out
data = 50, left = 45, right = 73
data = 45, left = 32, right = 48
data = 32, left = 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
alswnqodrl@alswnqodrl-900X3K:~$
```

05

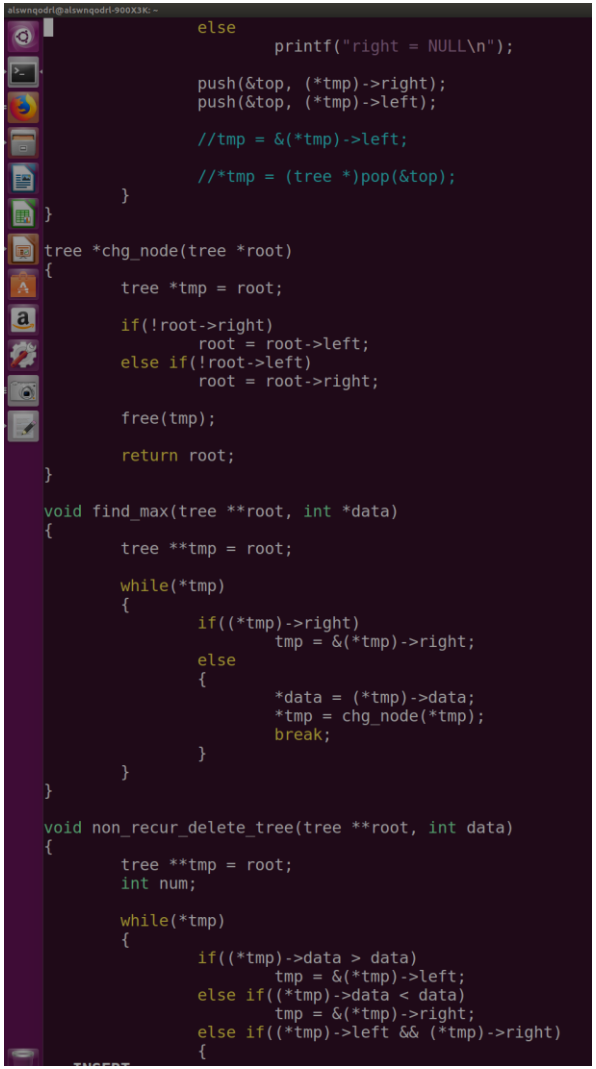
이진 트리를 재귀 호출 없이 구현하도록 한다.



```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 (*top)->data;
        return data;
}

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



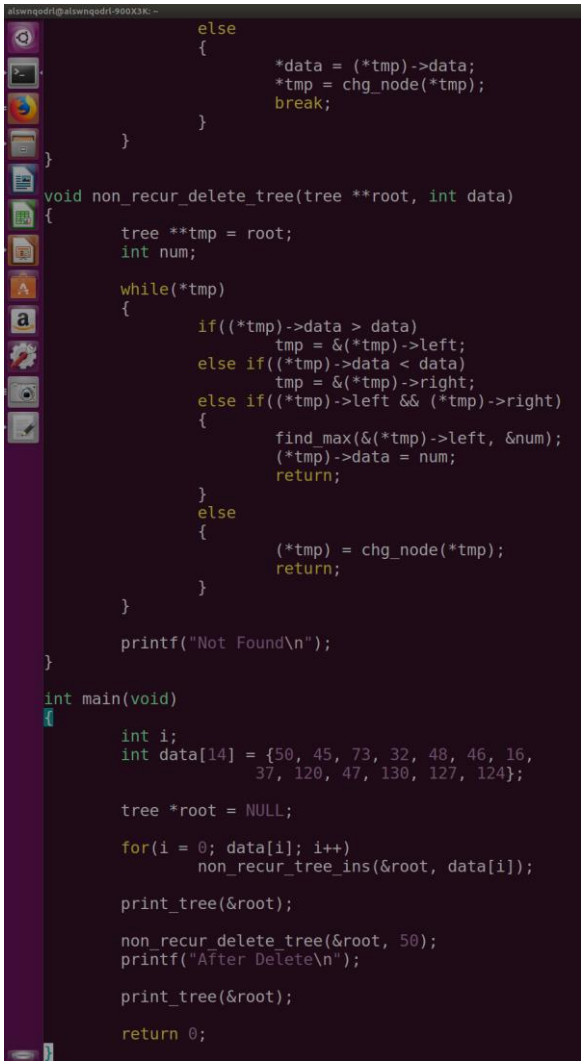
```c
void push(stack **top, void *data)
{
        if(data == NULL)
                return;

        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;
}

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)
```



```c
                else
                        printf("right = NULL\n");

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

                //tmp = &(*tmp)->left;

                //*tmp = (tree *)pop(&top);
        }
}

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;
}

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

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

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

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

05

이진 트리를 재귀 호출 없이 구현하도록 한다.

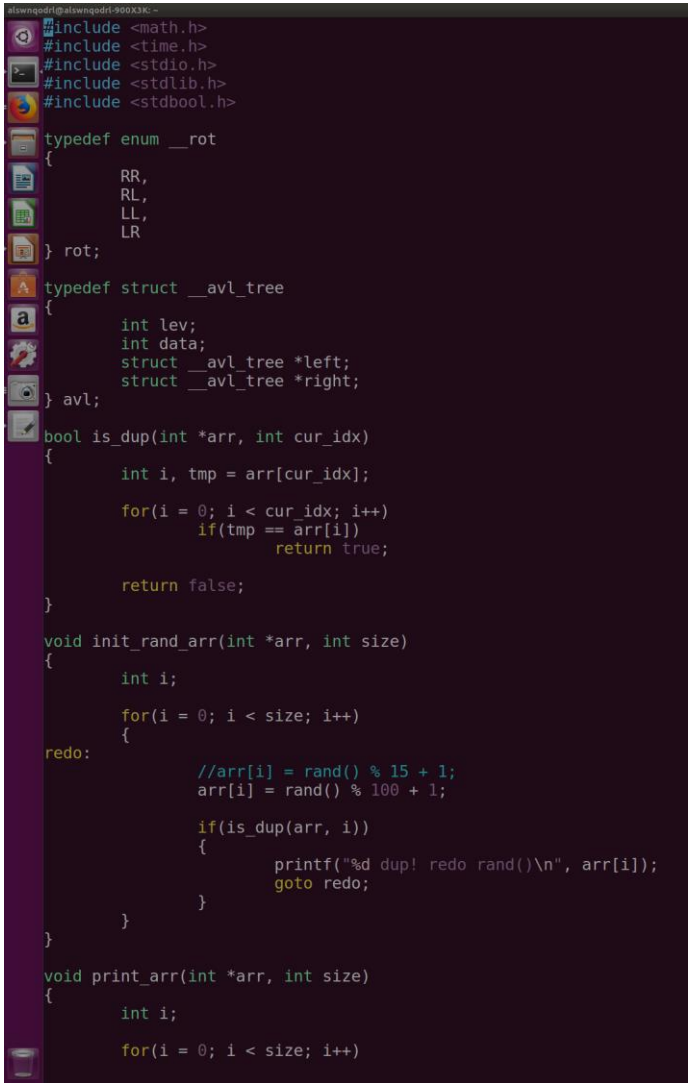


```
                else
                {
                        *data = (*tmp)->data;
                        *tmp = chg_node(*tmp);
                        break;
                }
        }
}

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

        while(*tmp)
        {
                if((*tmp)->data > data)
                        tmp = &(*tmp)->left;
                else if((*tmp)->data < data)
                        tmp = &(*tmp)->right;
                else if((*tmp)->left && (*tmp)->right)
                {
                        find_max(&(*tmp)->left, &num);
                        (*tmp)->data = num;
                        return;
                }
                else
                {
                        (*tmp) = chg_node(*tmp);
                        return;
                }
        }

        printf("Not Found\n");
}

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++)
                non_recur_tree_ins(&root, data[i]);

        print_tree(&root);

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

        print_tree(&root);

        return 0;
}
```

06

AVL 트리를 재귀 호출을 사용하여 구현하도록 한다.

```
alswnqodrl@alswnqodrl-900X3K:~$ vi data6.c
alswnqodrl@alswnqodrl-900X3K:~$ gcc data6.c
alswnqodrl@alswnqodrl-900X3K:~$ ./a.out
45 dup! redo rand()
arr[0] = 8
arr[1] = 51
arr[2] = 47
arr[3] = 67
arr[4] = 45
arr[5] = 28
arr[6] = 21
arr[7] = 23
arr[8] = 37
arr[9] = 15
arr[10] = 92
arr[11] = 6
arr[12] = 58
arr[13] = 90
arr[14] = 16

Debug AVL
Insert Rotation!
data = 70
LR Rotation
data = 75, lev = 3, left = 50, right = 100
data = 50, lev = 2, left = 25, right = 70
data = 25, lev = 1, left = NULL, right = NULL
data = 70, lev = 1, left = NULL, right = NULL
data = 100, lev = 2, left = NULL, right = 200
data = 200, lev = 1, left = NULL, right = NULL
alswnqodrl@alswnqodrl-900X3K:~$
```



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

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;
}

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

        for(i = 0; i < size; i++)
        {
redo:
                //arr[i] = rand() % 15 + 1;
                arr[i] = rand() % 100 + 1;

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

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

        for(i = 0; i < size; i++)
```

06

AVL 트리를 재귀 호출을 사용하여 구현하도록 한다.

```c
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;
}

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

                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);
        }
}

int update_level(avl *root)
{
        int left = root->left ? root->left->lev : 0;
        int right = root->right ? root->right->lev : 0;

        if(left > right)
                return left + 1;

        return right + 1;
}

int rotation_check(avl *root)
{
        int left = root->left ? root->left->lev : 0;
        int right = root->right ? root->right->lev : 0;
```

```c
        return right - left;
}

int kinds_of_rot(avl *root, int data)
{
        printf("data = %d\n", data);

        // for RR and RL
        if(rotation_check(root) > 1)
        {
                if(root->right->data > data)
                        return RL;

                return RR;
        }
        // for LL and LR
        else if(rotation_check(root) < -1)
        {
                if(root->left->data < data)
                        return LR;

                return LL;
        }
}

avl *rr_rot(avl *parent, avl *child)
{
        //parent->right = child->left ? child->left : child->right;
        parent->right = child->left;
        child->left = parent;
        parent->lev = update_level(parent);
        child->lev = update_level(child);
        return child;
}

avl *ll_rot(avl *parent, avl *child)
{
        //parent->left = child->right ? child->right : child->left;
        parent->left = child->right;
        child->right = parent;
        parent->lev = update_level(parent);
        child->lev = update_level(child);
        return child;
}

#if 0
avl *rl_rot(avl *parent, avl *child, int data)
{
}
#endif

#if 0
avl *lr_rot(avl *parent, avl *child, int data)
{
        avl *tmp;
```

06

AVL 트리를 재귀 호출을 사용하여 구현하도록 한다.

```
#if 0
avl *lr_rot(avl *parent, avl *child, int data)
{
        avl *tmp;

        if(child->right->data > data)
        {
                tmp = child->right->left;
                child->right->right = parent;
                child->right->left = child;
                child->right = tmp;
                parent->left = NULL;
        }
        else
        {
                tmp = child->right->right;
                child->right->right = parent;
                child->right->left = child;
                parent->left = tmp;
                tmp = parent->left;
                child->right = NULL;
        }

        return tmp;
}
#endif

avl *rl_rot(avl *parent, avl *child, int data)
{
        child = ll_rot(child, child->left);
        //child = ll_rot(child, child->left);
        return rr_rot(parent, child);
}

#if 1
avl *lr_rot(avl *parent, avl *child, int data)
{
        child = rr_rot(child, child->right);
        //child = rr_rot(child, child->left);
        return ll_rot(parent, child);
}
#endif

//void rotation(avl *root, int ret)
avl *rotation(avl *root, int ret, int data)
{
        switch(ret)
        {
                case RL:
                        printf("RL Rotation\n");
                        return rl_rot(root, root->right, data);
                case RR:
                        printf("RR Rotation\n");
                        return rr_rot(root, root->right);
                case LR:
                        printf("LR Rotation\n");
```

```
                        return lr_rot(root, root->left, data);
                case LL:
                        printf("LL Rotation\n");
                        return ll_rot(root, root->left);
        }
}


void avl_ins(avl **root, int data)
{
        if(!(*root))
        {
                (*root) = get_avl_node();
                (*root)->data = data;
                return;
        }

        if((*root)->data > data)
                avl_ins(&(*root)->left, data);
        else if((*root)->data < data)
                avl_ins(&(*root)->right, data);

        //update_level(root);
        (*root)->lev = update_level(*root);

        if(abs(rotation_check(*root)) > 1)
        {
                printf("Insert Rotation!\n");
                *root = rotation(*root, kinds_of_rot(*root, data), data);
                //rotation(*root, kinds_of_rot(*root, data));
        }
}

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

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

        free(tmp);

        return root;
}

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

06

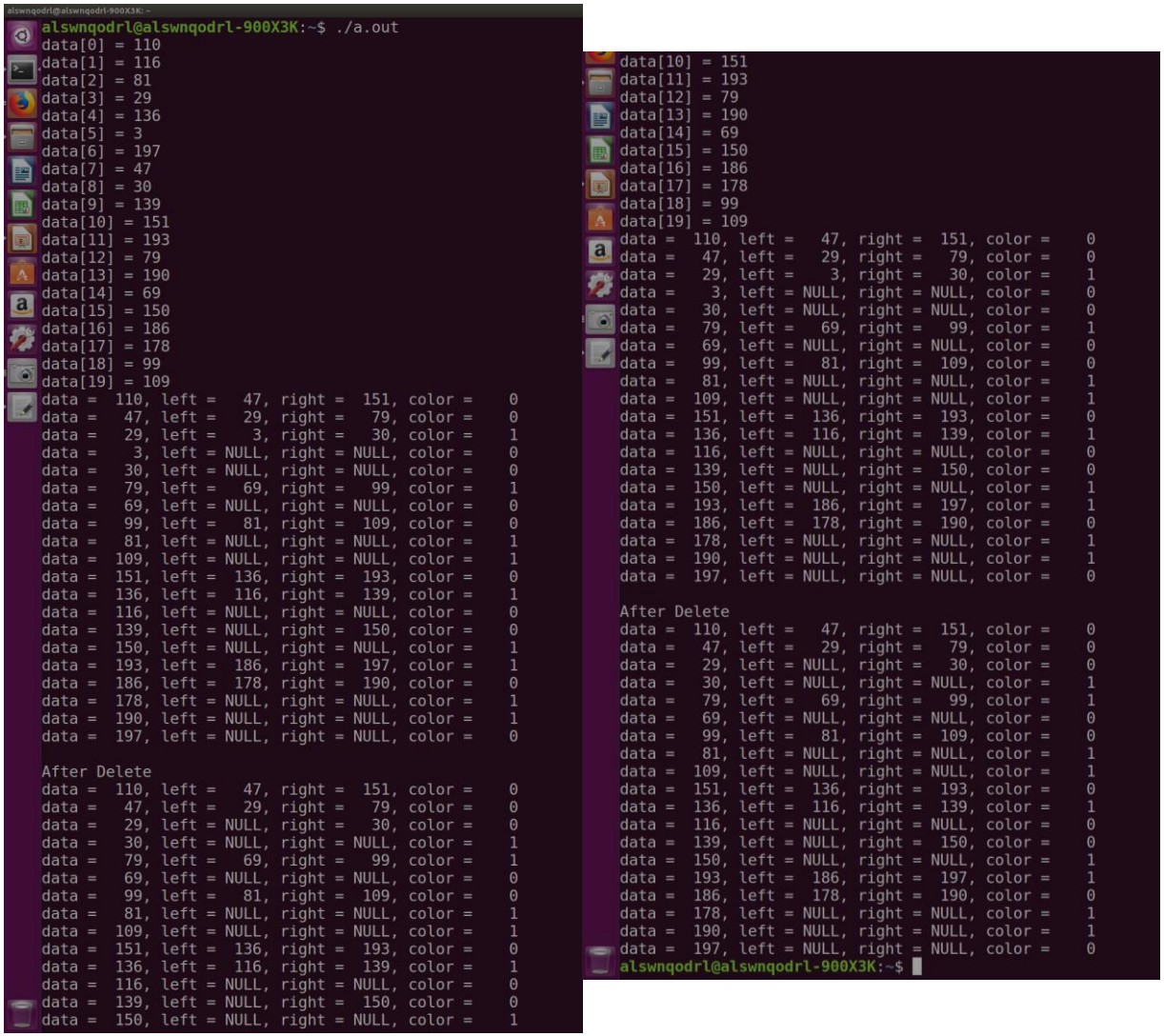
AVL 트리를 재귀 호출을 사용하여 구현하도록 한다.

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

        return root;
}

void avl_del(avl **root, int data)
{
        if(*root == NULL)
        {
                printf("There are no data that you find %d\n", data);
                return;
        }
        else if((*root)->data > data)
                avl_del(&(*root)->left, data);
        else if((*root)->data < data)
                avl_del(&(*root)->right, data);
        else if((*root)->left && (*root)->right)
                (*root)->left = find_max((*root)->left, &(*root)->data);
        else
        {
                *root = chg_node(*root);
                return;
        }

        (*root)->lev = update_level(*root);

        if(abs(rotation_check(*root)) > 1)
        {
                printf("Delete Rotation!\n");
                *root = rotation(*root, kinds_of_rot(*root, data), data);
                //rotation(*root, kinds_of_rot(*root, data));
        }
}

int main(void)
{
        int i;
        avl *root = NULL;
        avl *test = NULL;
        int arr[16] = {0};
        int size = sizeof(arr) / sizeof(int) - 1;

        //int data[] = {100, 50, 200, 25, 75, 80};
        int data[] = {100, 50, 200, 25, 75, 70};

        srand(time(NULL));

        init_rand_arr(arr, size);
```

```
                (*root)->left = find_max((*root)->left, &(*root)->data);
        else
        {
                *root = chg_node(*root);
                return;
        }

        (*root)->lev = update_level(*root);

        if(abs(rotation_check(*root)) > 1)
        {
                printf("Delete Rotation!\n");
                *root = rotation(*root, kinds_of_rot(*root, data), data);
                //rotation(*root, kinds_of_rot(*root, data));
        }
}

int main(void)
{
        int i;
        avl *root = NULL;
        avl *test = NULL;
        int arr[16] = {0};
        int size = sizeof(arr) / sizeof(int) - 1;

        //int data[] = {100, 50, 200, 25, 75, 80};
        int data[] = {100, 50, 200, 25, 75, 70};

        srand(time(NULL));

        init_rand_arr(arr, size);
        print_arr(arr, size);

#if 0
        for(i = 0; i < size; i++)
                avl_ins(&root, arr[i]);

        print_tree(root);

        printf("\nAfter Delete\n");
        avl_del(&root, arr[3]);
        avl_del(&root, arr[6]);
        avl_del(&root, arr[9]);

        print_tree(root);
#endif

        printf("\nDebug AVL\n");

        for(i = 0; i < 6; i++)
                avl_ins(&test, data[i]);

        print_tree(test);

        return 0;
}
```

07

Red Black 트리와 AVL 트리를 비교해보도록 한다.



```
alswnqodrl@alswnqodrl-900X3K:~$ ./a.out
data[0] = 110
data[1] = 116
data[2] = 81
data[3] = 29
data[4] = 136
data[5] = 3
data[6] = 197
data[7] = 47
data[8] = 30
data[9] = 139
data[10] = 151
data[11] = 193
data[12] = 79
data[13] = 190
data[14] = 69
data[15] = 150
data[16] = 186
data[17] = 178
data[18] = 99
data[19] = 109
data =  110, left =   47, right =  151, color =    0
data =   47, left =   29, right =   79, color =    0
data =   29, left =    3, right =   30, color =    1
data =    3, left = NULL, right = NULL, color =    0
data =   30, left = NULL, right = NULL, color =    0
data =   79, left =   69, right =   99, color =    1
data =   69, left = NULL, right = NULL, color =    0
data =   99, left =   81, right =  109, color =    0
data =   81, left = NULL, right = NULL, color =    1
data =  109, left = NULL, right = NULL, color =    1
data =  151, left =  136, right =  193, color =    0
data =  136, left =  116, right =  139, color =    1
data =  116, left = NULL, right = NULL, color =    0
data =  139, left = NULL, right =  150, color =    0
data =  150, left = NULL, right = NULL, color =    1
data =  193, left =  186, right =  197, color =    1
data =  186, left =  178, right =  190, color =    0
data =  178, left = NULL, right = NULL, color =    1
data =  190, left = NULL, right = NULL, color =    1
data =  197, left = NULL, right = NULL, color =    0

After Delete
data =  110, left =   47, right =  151, color =    0
data =   47, left =   29, right =   79, color =    0
data =   29, left = NULL, right =   30, color =    0
data =   30, left = NULL, right = NULL, color =    1
data =   79, left =   69, right =   99, color =    1
data =   69, left = NULL, right = NULL, color =    0
data =   99, left =   81, right =  109, color =    0
data =   81, left = NULL, right = NULL, color =    1
data =  109, left = NULL, right = NULL, color =    1
data =  151, left =  136, right =  193, color =    0
data =  136, left =  116, right =  139, color =    1
data =  116, left = NULL, right = NULL, color =    0
data =  139, left = NULL, right =  150, color =    0
data =  150, left = NULL, right = NULL, color =    1
data[10] = 151
data[11] = 193
data[12] = 79
data[13] = 190
data[14] = 69
data[15] = 150
data[16] = 186
data[17] = 178
data[18] = 99
data[19] = 109
data =  110, left =   47, right =  151, color =    0
data =   47, left =   29, right =   79, color =    0
data =   29, left =    3, right =   30, color =    1
data =    3, left = NULL, right = NULL, color =    0
data =   30, left = NULL, right = NULL, color =    0
data =   79, left =   69, right =   99, color =    1
data =   69, left = NULL, right = NULL, color =    0
data =   99, left =   81, right =  109, color =    0
data =   81, left = NULL, right = NULL, color =    1
data =  109, left = NULL, right = NULL, color =    1
data =  151, left =  136, right =  193, color =    0
data =  136, left =  116, right =  139, color =    1
data =  116, left = NULL, right = NULL, color =    0
data =  139, left = NULL, right =  150, color =    0
data =  150, left = NULL, right = NULL, color =    1
data =  193, left =  186, right =  197, color =    1
data =  186, left =  178, right =  190, color =    0
data =  178, left = NULL, right = NULL, color =    1
data =  190, left = NULL, right = NULL, color =    1
data =  197, left = NULL, right = NULL, color =    0

After Delete
data =  110, left =   47, right =  151, color =    0
data =   47, left =   29, right =   79, color =    0
data =   29, left = NULL, right =   30, color =    0
data =   30, left = NULL, right = NULL, color =    1
data =   79, left =   69, right =   99, color =    1
data =   69, left = NULL, right = NULL, color =    0
data =   99, left =   81, right =  109, color =    0
data =   81, left = NULL, right = NULL, color =    1
data =  109, left = NULL, right = NULL, color =    1
data =  151, left =  136, right =  193, color =    0
data =  136, left =  116, right =  139, color =    1
data =  116, left = NULL, right = NULL, color =    0
data =  139, left = NULL, right =  150, color =    0
data =  150, left = NULL, right = NULL, color =    1
data =  193, left =  186, right =  197, color =    1
data =  186, left =  178, right =  190, color =    0
data =  178, left = NULL, right = NULL, color =    1
data =  190, left = NULL, right = NULL, color =    1
data =  197, left = NULL, right = NULL, color =    0
alswnqodrl@alswnqodrl-900X3K:~$
```

색 구별(0과 1)에 의한 라벨링이 붙어 효율성이 높아졌다.
소스코드는 별첨

08

난수를 활용하여 Queue 를 구현한다.

```
alswnqodrl@alswnqodrl-900X3K:~$ vi data8.c
alswnqodrl@alswnqodrl-900X3K:~$ gcc data8.c
data8.c:11:1: warning: useless storage class specifier in empty declaration
 };
 ^
data8.c: In function 'addq':
data8.c:21:54: error: 'queue' undeclared (first use in this function)
   queue_pointer temp = (queue_pointer) malloc(sizeof(queue));
                                                      ^
data8.c:21:54: note: each undeclared identifier is reported only once for each f
unction it appears in
alswnqodrl@alswnqodrl-900X3K:~$ vi data8.c
alswnqodrl@alswnqodrl-900X3K:~$ gcc data8.c
data8.c:11:1: warning: useless storage class specifier in empty declaration
 };
 ^
alswnqodrl@alswnqodrl-900X3K:~$ vi data8.c
alswnqodrl@alswnqodrl-900X3K:~$ gcc data8.c
alswnqodrl@alswnqodrl-900X3K:~$ ./a.out
 68338718 -> 81505847 -> 88064959 -> 149628710 -> 169882828 -> 316532354 -> 3670
74628 -> 385791464 -> 409164176 -> 431879518 -> 473974412 -> 508220259 -> 543424
855 -> 567322464 -> 574852688 -> 589464582 -> 596106216 -> 633968604 -> 65654774
7 -> 675051094 -> 680390468 -> 681948651 -> 692077785 -> 701687040 -> 706792891
-> 707873043 -> 807789525 -> 839112812 -> 843862322 -> 853045356 -> 855143373 ->
 873512322 -> 921495458 -> 993491032 -> 999201983 -> 1014665490 -> 1024169433 ->
 1029225816 -> 1033929169 -> 1058044432 -> 1065711923 -> 1074996879 -> 107605864
0 -> 1101676882 -> 1122953321 -> 1123058421 -> 1144656860 -> 1199108645 -> 12314
00435 -> 1285942313 -> 1287048172 -> 1308217173 -> 1366276611 -> 1385127173 -> 1
408479931 -> 1409218104 -> 1413316637 -> 1462976904 -> 1463082928 -> 1516469871
-> 1522235306 -> 1606976774 -> 1620636542 -> 1634491406 -> 1672164856 -> 1712778
797 -> 1713626536 -> 1722538533 -> 1726197233 -> 1733162287 -> 1769905202 -> 178
2191585 -> 1783625534 -> 1828374991 -> 1840359450 -> 1857970161 -> 1872570275 ->
 1904265457 -> 1911568523 -> 1919834752 -> 1925634047 -> 1926308879 -> 192855202
8 -> 1937168896 -> 1962112358 -> 1971197163 -> 1971303188 -> 1984494656 -> 19975
54098 -> 2006898170 -> 2010057875 -> 2015163727 -> 2047285242 -> 2052154001 -> 2
057956320 -> 2058354396 -> 2100588966 -> 2102697836 -> 2128980301 -> 2137723911
-> NULL
alswnqodrl@alswnqodrl-900X3K:~$ vi data8.c
alswnqodrl@alswnqodrl-900X3K:~$
```

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

typedef struct queue *queue_pointer;
typedef struct queue {
  int item;
  queue_pointer link;
}queue;

queue_pointer front = NULL, rear = NULL;

void addq(int item)
{
  queue_pointer temp = (queue_pointer) malloc(sizeof(queue));
  temp->item = item;
  temp->link = NULL;
  if (front)
    rear->link = temp;
  else
    front = temp;
  rear = temp;
}

queue_pointer find_minimum()
{
  queue_pointer temp = front->link, mintemp = front, premintemp = NULL, pre = front;
  for (; temp != NULL; temp = temp->link){
    if (mintemp->item > temp->item) {
      premintemp = pre;
      mintemp = temp;
    }
    if ( temp->link != NULL )
      pre = pre->link;
  }
  if ( mintemp == front )
    return NULL;
  else
    return premintemp;
}

int delete_node(queue_pointer temp)
{
  int item;
  queue_pointer delnode;
  if ( temp == NULL ) {
    item = front->item;
    delnode = front;
    front = front->link;
  }
  else {
    item = temp->link->item;
    delnode = temp->link;
    temp->link = delnode->link;
  }
  free(delnode);
-- INSERT --
```

08

난수를 활용하여 Queue 를 구현한다.

```
    if ( temp->link != NULL )
      pre = pre->link;
  }
  if ( mintemp == front )
    return NULL;
  else
    return premintemp;
}

int delete_node(queue_pointer temp)
{
  int item;
  queue_pointer delnode;
  if ( temp == NULL ) {
    item = front->item;
    delnode = front;
    front = front->link;
  }
  else {
    item = temp->link->item;
    delnode = temp->link;
    temp->link = delnode->link;
  }
  free(delnode);
  return item;
}

int main(void)
{
  int count;
  queue_pointer p;

  srand(time(NULL));
  for (count=0; count<100; count++)
    addq(rand());

  for (count=0; count<100; count++){
    p = find_minimum();
    printf(" %d ->", delete_node(p));
  }
  printf(" NULL\n");
}
```

09

재귀호출을 사용하여 queue 를 구현하고 10, 20 을 집어넣는다.

```
alswnqodrl@alswnqodrl-900X3K:~$ gcc da
alswnqodrl@alswnqodrl-900X3K:~$ ./a.ou
head->data = 10
head->data = 20
head->data = 30
Now you delete 20
head->data = 10
head->data = 30
alswnqodrl@alswnqodrl-900X3K:~$
```

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

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

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

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

int main(void)
{
        int i;

        queue *head = NULL;

        srand(time(NULL));

        for(i = 0; i < 3; i++)
                enqueue(&head, (i + 1) * 10);

        print_queue(head);

        dequeue(&head, 20);

        print_queue(head);

        return 0;
}
```

09

재귀호출을 사용하여 queue 를 구현하고 10, 20 을 집어넣는다.
enqueue그림 설명

Main 0
Head 1000->2000

enqueue head(1000)	data(10)
Enqueue head(2004)	data(20)
enqueue head(3004)	data(30)

Heap

10 data(2000)	0->3000 link
20 data(3000)	4000 link
30 data(4000)	0 link

10

10. 파이프라인이 깨지는 경우 어떠한 이점이 있는지 기술하시오.

11

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

alswnqodrl@alswnqodrl-900X3K: ~/Downloads

```
}

void rb_tree_del_fixup(rb_tree *tree, rb_node *x)
{
        rb_node *root = tree->root->left;
        rb_node *w;

        while((!x->color) && (root != x))
        {
                if(x->parent->left == x)
                {
                        w = x->parent->right;

                        if(w->color)
                        {
                                w->color = BLACK;
                                x->parent->color = RED;
                                rb_left_rotate(&tree, x->parent);
                                w = x->parent->right;
                        }

                        if((!w->right->color) && (!w->left->color))
                        {
                                w->color = RED;
                                x = x->parent;
                        }
                        else
                        {
                                if(!w->right->color)
                                {
                                        w->left->color = BLACK;
                                        w->color = RED;
                                        rb_right_rotate(&tree, w);
                                        w = x->parent->right;
                                }

                                w->color = x->parent->color;
                                x->parent->color = BLACK;
                                w->right->color = BLACK;
                                rb_right_rotate(&tree, x->parent);
                                x = root;
                        }
                }
                else
                {
                        w = x->parent->left;

                        if(w->color)
                        {
                                w->color = BLACK;
                                x->parent->color = 1;
                                rb_right_rotate(&tree, x->parent);
                                w = x->parent->left;
                        }

                        if((!w->right->color) && (!w->left->color))
                        {
                                w->color = RED;
                                x = x->parent;
                        }
                        else
                        {
                                if((!w->right->color) && (!w->left->color))
                                {
                                        w->right->color = BLACK;
                                        w->color = RED;
                                        rb_left_rotate(&tree, w);
                                        w = x->parent->left;
                                                                316,1
```

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

alswnqodrl@alswnqodrl-900X3K: ~/Downloads

```
}

void rb_tree_del_fixup(rb_tree *tree, rb_node *x)
{
        rb_node *root = tree->root->left;
        rb_node *w;

        while((!x->color) && (root != x))
        {
                if(x->parent->left == x)
                {
                        w = x->parent->right;

                        if(w->color)
                        {
                                w->color = BLACK;
                                x->parent->color = RED;
                                rb_left_rotate(&tree, x->parent);
                                w = x->parent->right;
                        }

                        if((!w->right->color) && (!w->left->color))
                        {
                                w->color = RED;
                                x = x->parent;
                        }
                        else
                        {
                                if(!w->right->color)
                                {
                                        w->left->color = BLACK;
                                        w->color = RED;
                                        rb_right_rotate(&tree, w);
                                        w = x->parent->right;
                                }

                                w->color = x->parent->color;
                                x->parent->color = BLACK;
                                w->right->color = BLACK;
                                rb_right_rotate(&tree, x->parent);
                                x = root;
                        }
                }
                else
                {
                        w = x->parent->left;

                        if(w->color)
                        {
                                w->color = BLACK;
                                x->parent->color = 1;
                                rb_right_rotate(&tree, x->parent);
                                w = x->parent->left;
                        }

                        if((!w->right->color) && (!w->left->color))
                        {
                                w->color = RED;
                                x = x->parent;
                        }
                        else
                        {
                                if((!w->right->color) && (!w->left->color))
                                {
                                        w->right->color = BLACK;
                                        w->color = RED;
                                        rb_left_rotate(&tree, w);
                                        w = x->parent->left;
                                                                316,1
```

최적화 프로세스를 기술하도록 한
다.

Unity 를 계속 사용하는 방법
Synaptic 패키지 관리자로 가서 unity-2d 를 설치합니다. (ompiz를 제거
)
로그인 화면에서 아래쪽에 Unity 2D 가 선택되어있는지 확인

전통적인 Gnome Desktop으로 돌아가는 방법
로그인 화면 아래의 Ubuntu를 Ubuntu Classic으로 변경
그리고 gnome에서도 compiz가 작동할 우려가 있으니 compiz를 제거

Queue 에서 데이터로서 숫자 값이 아닌 문자열을 받아보도록 하자.

15

AVL 트리에 데이터로서 숫자가 아닌 문자열을 입력하도록 프로그램
하시오.

Binary Tree 에 문자열을 입력한
다.

17

```
#include
int func(int n){        return n*n;}
void apply(int arr[2][3], int (*p)(int n)){        int i, j;        for(i=0; i<3; i++)        {                arr[1][i]=func(arr[0][i]);                printf("%d\t", arr[0][i]);        }
        printf("\n");        for(i<0; i<3; i++)        printf("%d\t", arr[1][i]);}
int main(void){        int arr[2][3]={{1,2,3},{1,2,3}};        apply(arr, func);}~
```