

시험 1 차

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com

1. 이것이 없으면 사실상 C 언어를 사용할 수 없다.

C 언어에서 함수를 호출하기 위해서도 이것은 반드시 필요하다.

이와 같은 이유로 운영체제의 부팅 코드에서도

이것을 설정하는 작업이 진행되는데 이것은 무엇일까 ?

독립적 수행가능한 **main()** 함수를 가져야함.

2. 배열에 아래와 같은 정보들이 들어있다.

2400, 2400, 2400, 2400, 2400, 2400, 2400,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5,
2400, 2400, 2400, 2400, 2400, 2400, 1, 2, 3, 4,
5, 1, 2, 3, 4, 5, 2400, 2400, 2400, 2400, 2400,
5000,
1, 2, 3, 4, 5, 5000, 5000, 500, 500, 500, 500,
1, 2, 3, 4, 5, 500, 500, 500, 500, 500, 500,
500,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4,
5,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4,
5,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4,
5,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4,
5,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4,
5,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4,
5,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10,
11, 12, 13, 14, 15, 16, 17, 18, 234, 345, 26023,
346, 345, 234,
457, 3, 1224, 34, 646, 732, 5, 4467, 45, 623, 4,
356, 45, 6, 123,
3245, 6567, 234, 567, 6789, 123, 2334, 345, 4576, 678,
789, 1000,
2400, 2400, 2400, 2400, 2400, 2400, 1, 2, 3, 4, 5, 1,
2, 3, 4, 5,
2400, 2400, 2400, 2400, 978, 456, 234756, 5000, 5000,
5000, 2400, 500, 5000, 2400, 500,
500, 500, 500, 500, 500, 1, 2, 3, 4, 5, 500, 500,
500, 500, 500,
500, 500, 500, 500, 500, 1, 2, 3, 4, 5, 500, 500, 500,
5000, 2400, 5000,

5000, 5000, 5000, 5000, 5000, 5000, 5000, 1, 2, 3, 4,
5, 5000, 5000,
5000, 5000, 2400 5000, 500, 2400, 5000, 5000, 5000, 5000,
5000, 5000, 5000, 5000,
1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 5000, 5000,
5000, 5000, 5000,
1, 2, 3, 4, 5, 5000, 5000, 5000, 5000, 5000, 234,
4564, 3243, 876,
645, 534, 423, 312, 756, 235, 75678, 2400, 5000, 500,
2400, 5000, 500, 2400, 5000,
500, 2400, 5000, 500, 2400, 5000, 500, 2400, 5000, 500, 2400,
5000, 500, 2400, 5000, 500, 2400,
5000, 500, 2400, 5000, 500, 7, 8, 9, 6, 7, 8, 9, 6, 7, 8, 9, 6, 7, 8,
9, 6, 7, 8, 9, 6, 7, 8,
9, 6, 500, 2400, 5000, 500, 2400, 5000, 500, 2400, 5000, 500,
2400, 5000, 500, 2400, 5000,
500, 2400, 5000,

여기서 가장 빈도수가 높은 3 개의 숫자를 찾아 출력하시오!

함수에서 이 작업을 한 번에 찾을 수 있도록 하시오.

(찾는 작업을 여러번 분할하지 말란 뜻임)

3. 12 비트 ADC 를 가지고 있는 장치가 있다.

보드는 12 V 로 동작하고 있고 ADC 는 -5 ~ 5 V 로 동작한다.

ADC 에서 읽은 값이 2077 일 때

이 신호를 디지털 관점에서 재해석하도록 프로그래밍 한다.

4. 전세계 각지의 천재들이 모여서 개발하는 리눅스 운영체제 코드에는

엄청나게 많은 양의 goto 가 사용되고 있다.

goto 를 도대체 왜 사용해야만 할까 ?

if 문과 flag 문을 사용할때 jmp 등 파이프라인에 손실을 가져다준다. 허나 goto 는 if 문
flag 문을 사용하지 않기때문에 가속성이 좋다.

5. 포인터 크기에 대해 알고 있는대로 기술하시오.

포인터는 메모리 위치를 가리키기 위한 주소를 담는 변수다. 만약 32 비트 시스템이라면 CPU 의 메
모리 접근을 위한 주소가 32 비트 만큼 가능하기 때문에 포인터도 32 비트(4 바이트)가 된다. 반면,
64 비트 시스템이라면 64 비트(8 바이트) 만큼 가능하기 때문에 포인터도 64 비트(8 바이트)가 된다.

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

위의 문장에서 **Safety MCU** 가 몇 번째 부터 시작하는지 찾아내도록 프로그래밍 해보자.

(이 정도는 가볍게 해야 파싱 같은 것도 쉽게 할 수 있다)

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

-코드

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

```
int main(void)
{
    int a[3][3]={ {1,2,3},{4,5,6},{7,8,9}};
    int b[3][3]={ {1,1,1},{2,2,2},{3,3,3}};
    int c[3][3];
    int i,j,k;

    for(i=0; i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            c[j][i]=0;
            for(k=0;k<3;k++)
            {
                c[j][i]+=a[j][k]*b[k][i];
            }
        }
    }
    printf("%d %d %d\n%d %d %d\n%d %d %d",c[0][0],c[0][1],c[0][2],c[1][0],
        c[1][1],c[1][2],c[2][0],c[2][1],c[2][2]);
}
```

-결과

| | | |
|----|----|----|
| 14 | 14 | 14 |
| 32 | 32 | 32 |
| 50 | 50 | 50 |

8. void (* signal(int signum, void (* handler)(int)))(int) 의 프로토타입을 기술하시오.

리턴: void(*) (int)

함수명: signal

인자: int signum 과 void(*handler)(int)

signal 은 인자로 (int 형 signum)과 (int 형인자를 갖고 반환값이 없는 함수에 대한 포인터 handler)를 갖고, int 를 인자로 갖고 void 를 리턴하는 함수에 대한 포인터를 리턴하는 함수이다.

9. 함수 포인터를 반환하고 함수 포인터를 인자로 취하는 함수의 주소를 반환하고 인자로 int 2 개를 취하는 함수를 작성하도록 한다.

(프로토타입이 각개 다를 수 있으므로 프로토타입을 주석으로 기술하도록 한다

10. 파이프라인이 깨지는 경우 어떠한 이점이 있는지 기술하시오.

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

12. 값이 1 ~ 4096 까지 무작위로 할당되어 배열에 저장되도록 프로그래밍 하시오.
(배열의 크기는 100 개정도로 잡는다)

-code

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

int main(void)
{
    int i;
    int arr[100];
    for(i=0;i<100;i++)
    {
        arr[i]=rand()%4096+1;
        printf("%d\n",arr[i]);
    }
}
```

결과

```
2350
611
563
1951
1225
1946
2918
3378
13
1975
2609
1112
163
2394
293
2397
261
791
2136
2793
1118
2260
3499
3250
```

13. 12 번 문제에서 각 배열은 물건을 담을 수 있는 공간에 해당한다.

앞서서 100 개의 공간에 물건들을 담았는데 공간의 낭비가 있을 수 있다.

이 공간의 낭비가 얼마나 발생했는지 파악하는 프로그램을 작성하시오

14. 13 번 문제에서 확장하여 공간을 보다 효율적으로 관리하고 싶어서
4096, 8192, 16384 등의 4096 배수로 크기를 확장할 수 있는 시스템을 도입했다.
이제부터 공간의 크기는 4096 의 배수이고
최소 크기는 4096, 최대 크기는 131072 에 해당한다.
발생할 수 있는 난수는 1 ~ 131072 로 설정하고
이를 효율적으로 관리하는 프로그램을 작성하시오.
(사실 리눅스 커널의 Buddy 메모리 관리 알고리즘임)

15. 이진 트리를 재귀 호출을 사용하여 구현하도록 한다.

(일반적인 SW 회사들 면접 당골 문제 - 이게 되면 큐 따위야 문제 없음)

- code

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

typedef struct __tree
{
    int data;
    struct __tree *right;
    struct __tree *left;
}Tree;

Tree *get_node()
{
    Tree *tmp;
    tmp=(Tree *)malloc(sizeof(Tree));
    tmp->right=NULL;
    tmp->left=NULL;
    return tmp;
}

void print_tree(Tree *root)
{
    if(root!=NULL)
    {
        printf("%d\n",root->data);
        print_tree(root->left);
        print_tree(root->right);
    }
}

void Enqueue(Tree **root,int data)
{
    if(*root==NULL)
    {
        *root=get_node();
        (*root)->data=data;
        return;
    }

    if((*root)->data>data)
        Enqueue(&(*root)->left,data);
    else if((*root)->data<data)
        Enqueue(&(*root)->right,data);
    return;
}

Tree *Change(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)
```

-결과

```
79
23
11
18
15
45
33
58
55
90
170
124
93
*****79를 삭제합니다*****
58
23
11
18
15
45
33
55
90
170
124
93
*****170를 삭제합니다*****
58
23
11
18
15
45
33
55
90
124
93
*****93를 삭제합니다*****
58
23
11
18
15
45
33
55
90
124
```

```

{
    if(root->right)
        root->right = find_max(root->right, data);
    else
    {
        *data = root->data;
        root = Change(root);
    }

    return root;
}

```

```
Tree *Dequeue(Tree *root, int data)
```

```

{
    int num;
    Tree *tmp;
    if(root == NULL)
    {
        printf("삭제할 데이터가 없어\n");
        return NULL;
    }
    else if(root->data > data)
        root->left = Dequeue(root->left, data);
    else if(root->data < data)
        root->right = Dequeue(root->right, data);
    else if(root->left && root->right)
    {
        root->left = find_max(root->left, &num);
        root->data = num;
    }
    else
        root = Change(root);
    return root;
}

```

```
int main(void)
```

```

{
    Tree *root=NULL;
    int arr[13]={79,23,45,11,18,15,90,33,58,170,124,93,55};
    int i,j,k;
    for(i=0;i<13;i++)
        Enqueue(&root,arr[i]);
    print_tree(root);
    for(j=0;j<3;j++)
    {
        k=rand()%13+1;
        Dequeue(root,k);
        printf("*****%d 를 삭제합니다*****\n",k);
        print_tree(root);
    }
    return 0;
}

```


16. 이진 트리를 재귀 호출 없이 구현하도록 한다.

결과를 확인하는 `print` 함수(전위, 중위, 후위 순회) 또한 재귀 호출을 수행하면 안됨

17. AVL 트리를 재귀 호출을 사용하여 구현하도록 한다.

18. AVL 트리를 재귀 호출 없이 구현하도록 한다.

19. Red Black 트리와 AVL 트리를 비교해보도록 한다.

20. 난수를 활용하여 Queue 를 구현한다.

(중복되는 숫자를 허용하지 않도록 프로그래밍 하시오)

제일 좋은 방법은 배열을 16 개 놓고 `rand() % 16` 을 해서
숫자가 겹치지 않는지 확인하면 된다.

복되는 숫자를 허용하지 않도록 프로그래밍 하시오.
배열은 16 개 두었고, 1~16 까지 순서없이 저장했습니다.

-코드

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

typedef struct __queue
{
    int data;
    struct __queue *link;
}Queue;

Queue *get_node()
{
    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);
    return;
}

Queue *Dequeue(Queue *head,int data)
{
    Queue *tmp;
    tmp=head;
    if(head->data!=data)
        head->link=Dequeue(head->link,data);
    else
```

결과

```
8
7
10
4
2
16
11
13
14
12
3
5
9
15
6
1
*****3를 삭제합니다*****
8
7
10
4
2
16
11
13
14
12
5
9
15
6
1
```

```

    {
        tmp=head->link;
        free(head);
    }
    return tmp;
}

```

```

void print_queue(Queue *head)
{
    if((head)!=NULL)
    {
        printf("%d\n", (head->data));
        print_queue(head->link);
    }
}

```

```

int* randomm(int arr[])
{
    int i,k;
    for(i=0;i<16;i++)
    {
        arr[i]=rand()%16+1;
        for(k=0;k<i;k++)
        {
            while(arr[i]==arr[k])
            {
                arr[i]=rand()%16+1;
                k=0;
            }
        }
    }
    return arr;
}

```

```

int main(void)
{
    Queue *head=NULL;
    int i;
    int arr[16];
    int* arr_m=randomm(arr);
    for(i=0;i<16;i++)
    {
        Enqueue(&head,arr_m[i]);
    }
}

```

```

    }
    print_queue(head);
    int a=rand()%16+1;
    head=Dequeue(head,arr[a]);
    printf("*****%d 를 삭제합니다***** \n",arr[a]);
    print_queue(head);
    return 0;
}

```

21. 함수 포인터를 활용하여 float 형 3 by 3 행렬의 덧셈과 int 형 3 by 3 행렬의 덧셈을 구현하도록 하시오.

-코드

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

```
void array_floatt(float a[][3],float b[][3])
{
```

```
    float c[3][3];
    int i,j,k;
```

```
    for(i=0; i<3;i++)
    {
```

```
        for(j=0;j<3;j++)
        {
```

```
            c[j][i]=0;
```

```
            c[j][i]=a[j][i]+b[j][i];
```

```
        }
```

```
    }
```

```
    printf("float 형 덧셈행렬 입니다.\n%.1f %.1f %.1f\n%.1f %.1f %.1f\n%.1f %.1f\n%.1f\n",c[0][0],c[0][1],c[0][2],c[1][0],
        c[1][1],c[1][2],c[2][0],c[2][1],c[2][2]);
```

```
}
```

```
void array_intt(int e[][3],int d[][3])
```

```
{
```

```
    int c[3][3];
    int i,j;
```

```
    for(i=0; i<3;i++)
    {
```

```
        for(j=0;j<3;j++)
        {
```

```
            c[j][i]=0;
```

```
            c[j][i]=e[j][i]+d[j][i];
```

- 결과

```

float형 덧셈행렬 입니다.
2.4  3.5  4.6
6.7  7.8  8.9
11.0 12.1 13.2

int형 덧셈행렬 입니다.
2  4  6
5  7  9
8 10 12

```

```

    }
}
printf("\nint 형 덧셈행렬 입니다.\n%d %d %d\n%d %d %d\n%d %d %d\n",c[0][0],c[0]
[1],c[0][2],c[1][0],
    c[1][1],c[1][2],c[2][0],c[2][1],c[2][2]);

}

int main(void)
{
    float a[3][3]={ {1.1,2.2,3.3},{4.4,5.5,6.6},{7.7,8.8,9.9}};
    float b[3][3]={ {1.3,1.3,1.3},{2.3,2.3,2.3},{3.3,3.3,3.3}};

    int e[3][3]={ {1,2,3},{4,5,6},{7,8,9}};
    int d[3][3]={ {1,1,1},{2,2,2},{3,3,3}};

    array_floatt(a,b);
    array_inttt(e,d);
    return 0;
}

```

22. 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.

(프로그래밍 하시오)

-코드

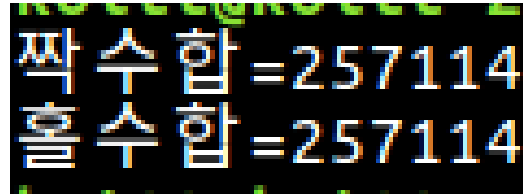
```
#include <stdio.h>

void main()
{
    int i;
    int sum1=0, sum2=0; // sum1 짝수, sum2 홀수
    int arr[27]={0,};
    arr[0]=1;
    arr[1]=1;

    for(i=2;i<27;i++)
    {
        arr[i]=arr[i-1]+arr[i-2];
    }
    for(i=0;i<27;i++)
    {
        if(arr[i]%2==0)
            sum1 = sum1+arr[i];
        else
            sum2 =sum2 +arr[i];
    }

    printf("짝수합=%d \n", sum1);
    printf("홀수합=%d \n", sum2);
}
```

- 결과



```
짝수합=257114
홀수합=257114
```

23. 1, 4, 5, 9, 14, 23, 37, 60, 97, ... 형태로 숫자가 진행된다.

23 번째 숫자는 무엇일까 ?

(프로그래밍 하시오)

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

23번째숫자 =81790

```
void main()
{
    int i;
    int sum1=0, sum2=0; // sum1 짝수, sum2 홀수
    int arr[27]={0,};
    arr[0]=1;
    arr[1]=4;

    for(i=2;i<23;i++)
    {
        arr[i]=arr[i-1]+arr[i-2];
    }

    printf("23 번째숫자 =%d \n", arr[22]);
}
```

24. Intel Architecture 와 ARM Architecture 의 차이점은 ?

arm 은 인텔에 비해 low 전압에서 작동하도록 설계되어있어 소형 기기 에 적합하다.
intel 은 arm 보다 peripheral 기기 들과 호환이 더 쉽게 설계되어 있다.

25. 우리반 학생들은 모두 25 명이다.

반 학생들과 함께 진행할 수 있는 사다리 게임을 만들도록 한다.

참여 인원수를 지정할 수 있어야하며

사다리 게임에서 걸리는 사람의 숫자도 조정할 수 있도록 만든다.

26. 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)

27. char *str = "WTF, Where is my Pointer ? Where is it ?" 라는 문자열이 있다

여기에 소문자가 총 몇 개 사용되었는지 세는 프로그램을 만들어보자

28. int *p[3] 와 int (*p)[3] 는 같은 것일까 ? 다른 것일까 ?

이유를 함께 기술하도록 한다.

같다.

int *p[3] : int 형 데이터를 3 개 저장할 수 있는 배열의 포인터 = (int (*)(3))

int (*p)[3] : int 형 데이터를 3 개 저장할 수 있는 배열의 포인터 = (int (*)(3))

으로 프로토타입이 같다.

29. 임의의 값 x 가 있는데, 이를 134217728 단위로 정렬하고 싶다면 어떻게 해야할까 ?

어떤 숫자를 넣던 134217728 의 배수로 정렬이 된다는 뜻임

(힌트 : 134217728 = 2^27)

-코드

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

```
int main(void)
{
```

```
    int number;
    scanf("%d",&number);
    int a=number&~(134217728-1);
    printf("%d,몇배=%d",a,a/134217728);
}
```

-결과

```
134217728
134217728, 몇배=1
270000000
268435456, 몇배=2
600000000
536870912, 몇배=4
```

30. 단 한 번의 연산으로 대소문자 변환을 할 수 있는 연산에 대해 기술하시오.

(프로그래밍 하시오), 덧셈 혹은 뺄셈 같은 기능이 아님

-코드

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

```
int main(void){
    char ch;
```

```
    printf("입력해: ");
    scanf("%c", &ch);
```

```
    if(ch >= 'A' && ch <= 'Z')
        printf("%c => %c", ch, ch+32);
    else if(ch >= 'a' && ch <= 'z')
        printf("%c => %c", ch, ch-32);
```

```
    return 0;
```

```
}
```

-결과

```
koitt@koitt
입력해: a
a => Akoitt
입력해: 56
koitt@koitt
입력해: d
d => Dkoitt
입력해: D
D => dkoitt
```

31. 변수의 정의를 기술하시오.

변수는 무언가 저장하는 곳, 데이터 배열이나 상수, 논리값(True, False)을 저장하기 위한 저장장소로서의 역할

32. 포인터의 정의를 기술하시오.

주소를 저장할 수 있는 변수, 주소를 저장하는 곳

33. 함수 포인터의 정의를 기술하시오.

함수 포인터는 말그대로 함수의 위치를 가리키는 포인터이다

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

35. 파이프라인은 언제 깨지는가 ?

분기가 발생할 때, 명령어들이 취소되면서 파이프라인이 깨지게된다. 분기 명령어는 파이프라인을 때려부순다.
jmp 나 call 등의 분기명령어가 많을
수록 힘들다.

36. 메모리 계층 구조에 대해 기술하시오.

메인메모리, 레지스터, 캐시, 하드디스크 구조이다. 보조 기억장치는 주기억장치보다 늦고, 주기억장치는 캐시메모리보다 속도가 느다.
주기억장치의 속도를 맞추기위해 메모리를 계층적으로 사용한다. 처리속도는: 레지스터 > 캐시메모리 > 주기억장치 > 보조기억장치

37. C 언어의 기본 메모리 구조에 대해 기술하시오.

메모리공간으로 text 영역, 데이터영역, 힙영역, 스택영역이 있다. text 영역은 코드가 저장되는 메모리 공간이다. 데이터영역은 전역변수와 static 변수가 할당되고, 프로그램 종료때까지 남아있다. 스택영역은 지역변수와 매개변수가 할당된다. 선언된 함수를 빠져나가면 없어진다.
heap 영역은 프로그래머가 원하는 시점에 변수를 할당하고 해제할 필요가있을 때 사용하는 영역이다.

38. 우리가 사용하는 윈도우나 리눅스, 맥(유닉스)에서 보는 변수 주소는 진짜 주소일까 아닐까 ? 알고 있는대로 기술하시오.

모두 가짜 주소이다. 이 주소값은 가상 메모리 주소에 해당하고, 운영체제의 실제 물리메모리의 주소로 변환된다.

윈도우와 맥(유닉스)도 리눅스의 가상메모리 개념을 사용한다. 리눅스의 경우 가상메모리는 32 비트, 64 비트 버전 두가지가 있다.

39. 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)

40. 리눅스에서 디버깅을 수행하기 위한 프로그램 컴파일 방법을 기술하시오.

1. c 언어 프로그램 작성 → 명령어: Vi 파일명.c
2. gcc 로 작성된 c 프로그램 컴파일 → 명령어: gcc -g -O0 -o debug(별칭) 파일명.c
3. 컴파일 완료된 파일 실행 → 명령어: ./debug(별칭)

41. 난수를 활용해서 Stack 을 구성한다.

같은 숫자가 들어가지 않게 하고 20 개를 집어넣는다.

이때 들어가는 숫자는 1 ~ 100 사이의 숫자로 넣는다.
(마찬가지로 중복되지 않게 한다)

-코드

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

#define EMPTY 0

    struct node{
        int data;
        struct node *link;
    };
typedef struct node Stack;

Stack *get_node()
{
    Stack *tmp;
    tmp=(Stack *)malloc(sizeof(Stack));
    tmp->link=EMPTY;
    return tmp;
}

void push(Stack **top, int data)
{
    Stack *tmp;
    tmp = *top;
    *top = get_node();
    (*top)->data = data;
    (*top)->link = tmp;
}

int pop(Stack **top)
{
    Stack *tmp;
    int num;
    tmp = *top;
    if(*top == EMPTY)
    {
        printf("Stack is empty!!!\n");
        return 0;
    }
    num = tmp->data;
    *top = (*top)->link;
    free(tmp);
    return num;
}

int* randomm(int arr[])
{
    int i,k;
    for(i=0;i<20;i++)
    {
        arr[i] = rand()%100+1;
        for(k=0;k<i;k++)
        {
            while(arr[i]==arr[k])
            {
```

-결과

69
12
37
73
41
27
64
60
91
28
63
22
50
93
36
94
16
78
87
84

```

        arr[i]=rand()%100+1;
        k=0;
    }
}
return arr;
}

```

```

int main(void)
{
    Stack *top = EMPTY;
    int i;
    int arr[20];
    int* arr_m=randomm(arr);
    for(i=0;i<20;i++)
        push(&top, arr_m[i]);
    for(i=0;i<20;i++)
        printf("%d\n", pop(&top));
    return 0;
}

```

42. 41 번에서 홀수만 빼내서 AVL 트리를 구성하도록 한다.

43. 41 번에서 짝수만 빼내서 RB 트리를 구성하도록 한다.

44. vi 에서 코드가 정렬이 잘 안되어 있다.

이 상황에서 어떻게 해야 예쁘고 깔끔하게 정렬되는가 ?

표준모드에서 ‘**gg=G**’를 순서대로 입력하거나 ‘**V**’로 드래그한 후 ‘**=**’ 를 누르면 깔끔하게 정렬된 모습을 볼 수 있다.

45. 프로그램을 최적화하는 컴파일 옵션을 적고

반대로 어떠한 최적화도 수행하지 않는 컴파일 옵션을 적도록 한다.

gcc -g -O0 -o debug(별칭) 파일이름.c → 별칭으로 저장됨
gcc 파일이름.C → a.out 으로 저장됨

46. 최적화 프로세스를 기술하도록 한다.

프로세스 클린

47. 기존에는 자료구조에서 숫자값만을 받았다.

이제 Queue 에서 데이터로서 숫자가 아닌 문자열을 받아보자.

48. Binary Tree 에서 위 작업을 수행해보라.

49. AVL Tree 에서 위 작업을 수행해보라.

50. 성적 관리 프로그램을 만들어보자.

1. 통계 기능(총 합산, 평균, 표준 편차 계산)
2. 성적순 정렬 기능
3. 성적 입력 기능
4. 학생 정보 삭제 기능

51. gdb 를 사용하는 이유를 기술하라.

디버깅은 문법적으로는 문제가 없으나 실행해보면 문제가 생기는 경우에 디버깅을 해서 그 오류가 발생 하는 지점을 찾아서 왜 오류가 났는지 확인하기 위함이다. gdb 은 디버거 프로그램인데, 어셈블리어를 분석할 수 있다.

52. 기계어 레벨에서 스택을 조정하는 명령어는 어떤것들이 있는가 ?

PUSH, POP

53. a 좌표(3, 6), b 좌표(4, 4) 가 주어졌다.

원점으로부터 이 두 좌표가 만들어내는 삼각형의 크기를 구하는 프로그램을 작성하라.

54. 가위 바위 보 게임을 만들어보자.

프로그램을 만들고 컴퓨터랑 배틀 한다.

55. 화면 크기가 가로 800, 세로 600 이다.

여기서 표현되는 값은 x 값이 $[-12 \sim +12]$ 에 해당하며 y 값은 $[-8 \sim +8]$ 에 해당한다.

x 와 y 가 출력하게 될 값들을 800, 600 화면에 가득차게 표현할 수 있는 스케일링 값을 산출하는 프로그램을 작성하도록 한다.

56. 등차 수열의 합을 구하는 프로그램을 for 문을 도는 방식으로 구현하고 등차 수열의 합 공식을 활용하여 구현해본다.

함수 포인터로 각각의 실행 결과를 출력하고

이 둘의 결과가 같은지 여부를 파악하는 프로그램을 작성하라.

57. $\sin(x)$ 값을 프로그램으로 구현해보도록 한다.

어떤 **radian** 값을 넣든지 그에 적절한 결과를 산출할 수 있도록 프로그래밍 한다.