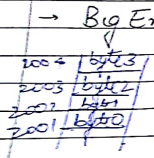
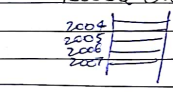
```
            s = zeroptr;
            zeroptr = zeroptr -> right;
            delete (s, & diff);
            count = count - 1;
        }
    }
    // insert count & sign
    // into the header.
    if (p -> info > 0)
        r -> info = count;
    else
```

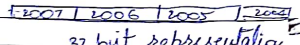
Union can actually be used to find the Byte order of comp

→ Big Endian



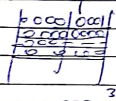
→ Little Endian.





32 bit representation

eg



number=0000 0000 ---- 01 (32 bits)

* Solving postfix expressions using stacks.

stack is data structure that works on last in first out principle

```
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 5
#define TRUE 1
#define FALSE 0
struct stack
{
    int top;
    int items[MAXSIZE];
};
int empty(struct stack *ps);
int full(struct stack *ps);
void push(struct stack *ps, int val)
```

```
int pop(struct stack *ps);
int size(struct stack *ps);
int stacktop(struct stack *ps, int *val);  //value of top.
void display(struct stack *ps);
// before push, check whether stack is full or not
// befor pop, check whether stack is empty, or not.
// if we overflow stacks in computer (stack memory) it will lost overdritting.
* // we usually don't pass structure by value coz it consumes lot of memory.
int main()
{
    int opt, data, i;
    struct stack st;
    st.top = -1;
```

```
int empty (struct stack *ps)
{
        return ((ps->top) == -1);
}
int full (struct stack *ps)
{
    if (ps->top == MAXSIZE-1)
        return TRUE;
    else
        return FALSE;
}
```

```
int void push (struct stack *ps, int val);
{
    if (full (ps))
        printf ("Stack overflow\n");
        exit (1);
    }
    else
    {   ps->top = ps->top +1;
        ps->items[top] = val;
        return;
    }
}
int pop (struct stack *ps)
{
    int x = ps->items[ps->top];
    if (empty (ps))
        printf (" Stack Underflow\n");
        exit (1);
    }
    else
    {   int x = ps->items[ps->top];
        ps->top--;
        return (x);
    }
}
int size (struct stack *ps)
{
        return (ps->top +1);
}
```

```
int stacktop(struct stack *ps, int *val)
{
    if(empty(ps))
        return(-1);
    else
        *val = ps->items[ps->top];
    return(0);
}
void display(struct stack *ps)
{

}
```

Using Dynamic Memory

```
struct node
{
    int data;
    struct node *next;
};

int empty(struct node *ps)    // pointer is head.
{
    if(ps == NULL)
        return(1);
    return(0);
}
```


```
void push(struct node **ps, int val)
{
    struct node *newnode;
    newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = val;
    newnode->next = *ps;
    *ps = newnode;
    free(newnode);
}
int pop(struct node **ps, )
{
    int x = (*ps)->data; if(empty(*ps))
    printf("\n stack empty \n");
else
{   int x = (*ps)->data;
    struct node *temp = *ps;
    (*ps) = (*ps)->next;
    free(temp);
}
}
```

free is imp