과제3\_1번

텍스트, 영수증이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

**과제3\_2번.c**

**1. Code explanation**

1) typedef struct ListNode

: structure type for a node in the list

2) typedef struct ListHeader

-length: number of nodes

-head: head pointer

-tail: pointer of the last node

3) void error(const char\* message)

: print a error message

-message: error message

4) void init(ListHeader\* list)

: initialize a list

-list: pointer variable of the list we will initialize

5) void insert\_node\_last(ListHeader\* list,int data )

: insert a node at the last of the list

-list: pointer variable of the list

-data: int type data to that we will insert in the list

-temp: temporary ListNode\* type variable to store the data and link of the node we will insert

6) void merge(ListHeader\* list1, ListHeader\* list2, ListHeader\* list3)

: merge list1 and list2 in ascending order and store it into list3

-list1, list2, list3: pointer variables of three lists.

list a, b are sended to parameter list1, 2, and list c is sended to parameter list3.

-a, b: head pointer of list 1, 2

7) void print\_list(ListHeader\* list)

: print a list

-list: pointer variable of the list that we will print

8. int main()

-m, n: number of nodes in list a and b

-int a\_data[MAX]: array to store the data of list a

-int b\_data[MAX]: array to store the data of list b

**2. Theoretical Explanation**

In main function

1) initialize list a, b, and c

2) get number of nodes in list a b as input. Store them in integer type variable m and n.

3) get list a and b as input. Store them in integer type arrau a\_data[] and b\_data[]

4) generate list a and b

5) merge list a and b and store the result in list c

->if a->data== b->data then insert them in list 3, move a and b to next node

->if a->data > b->data then insert b->data in list 3, move b to next node

->if a->data < b->data then insert a->data in list 3, move a to next node

->insert the rest of the node in list3

6) print list c

**3. Result**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**4. Time complexity analysis**

Let f(n): number of operations in function ‘insert\_node\_last’ when the list size is n

T(n): number of operations in function ‘merge’ when the sum of list 1 and list 2 size is n

f(n) = 1+2+2+1=5 (Worst case-> when temp!=NULL, and list->tail != NULL)

T(n) = 2 + ( f(n)+1 )\*n (Worst case-> when a->data and b->data are all different)

= 2+6n

6n <= 2+6n , 2+6n <= 7n (when n>=2),

So the time complexity is T(n)=O(n)=Ɵ(n)

**과제 3\_3번.c**

**1. Code explanation & Theroritical explanation**

1) typedef struct ListNode

: structure type for a node in the list

2) typedef struct ListHeader

-length: number of nodes

-head: head pointer

-tail: pointer of the last node

3) void error(const char\* message)

: print a error message

-message: error message

4) void init(ListHeader\* list)

: initialize a list

-list: pointer variable of the list

5) int is\_empty(ListType \*list)

: check if a list is empty

-list: pointer variable of the list

6) int get\_length(ListType \*list)

: return the length of the list

-list: pointer variable of the list

7) void insert\_node(ListNode\* head, ListNode\* p, ListNode\* new\_node)

: insert 'new\_node' right after node 'p' in the list whose pointer of head pointer is 'head'

-head: head pointer

-p: a node which will be right before the new node

-new node: a node we will insert

8) void remove\_node(ListNode\* head, ListNode\* p, ListNode\* removed)

: remove a node

-head: head pointer

-p: a node which is be right before the node we will remove

-removed: a node we will remove

9) ListNode\* get\_node\_at(ListType \*list, int pos)

: return node pointer whose location is 'pos' in the list

-list: pointer variable of the list

-pos: position of the node

10) void add(ListType \*list, int position, element data)

: insert a new node at 'position' th location

-list: pointer variable of the list

-pos: position where we will add the node

-data: data of the node

11) void delete(ListType\* list, int pos)

: delete a node whose location is 'pos'

-list: pointer variable of the list

-pos: position of the node

12) element get\_entry(ListType\* list, int pos)

: return the data whose location is 'pos'

-list: pointer variable of the list

-pos: position of the node

13) void display(ListType\* list)

: display the data of the whole list in the buffer

-list: pointer variable of the list

14) void add\_first(ListType \*list, element data)

: add a node at the beginning of the list

-list: pointer variable of the list

-data: data of the node

->if the list is blank then let head pointer and tail pointer point ‘node’ and node->link point NULL

->if not, then let the link of ‘node’ point head node and let the head pointer point ‘node’

->add 1 to the length

15) void add\_last(ListType\* list, element data)

: add a node at the end of the list

-list: pointer variable of the list

-data: data of the node

-temp: temporary variable the store the node

->if the list is empty, let head pointer and tail pointer point ‘temp’

->if not, let the last node point ‘temp’ and tail pointer point temp

->add 1 to the length

16) void delete\_first(ListType\* list)

: delete the first node in the list

-list: pointer variable of the list

-> if the list is blank, print “the list is blank”

->if not, let the head pointer point the second node and deallocate the memory of first node

17) void delete\_last(ListType\* list)

: delete the last node in the list

-list: pointer of the list

-> if the list is blank, print “the list is blank”

->if not, remove the last node using function ‘remove\_node’

18) int is\_in\_list(ListType\* list, element item)

: check if a node whose data is 'item' exists

-list: pointer of the list

-item: data of a node

-> return TRUE(1) if node exists, and return FALSE(0) if not

**2. Result**

**텍스트이(가) 표시된 사진

자동 생성된 설명**