Lesson 15 Lab

Task 1: to define a program to allocate a memory block with the size of 10 integers, and then resize it to as big as 20 integers, deallocate it before exiting the program

|  |
| --- |
| #include <stdio.h>  int main()  {      int \*ptr = malloc(10\* 4);      ptr = realloc(ptr, 20 \* 4);      free(ptr);      ptr = NULL;      return 0;  } |

Task 2: to define a function (named find\_sum) using what we just learnt, to traverse a linked list to figure out and return the sum of data in all nodes

Assume: we already have the setup as below:

Typedef struct node node;

struct node {

int data;

node \*next;

}

int find\_sum (node\* head);

/\* return the sum of all nodes’ value, return 0 if the list is empty or head is null \*/

|  |
| --- |
| int find\_sum (node\* head) {      node \*p;      int sum = 0;      for (p = head; p != 0; p = p->next)      {          sum += p->data;      }      return sum;  } |

Task 3: to define a function (named insert\_asc) using what we just learnt, to traverse a linked list with double pointer to insert a value into the linked list in ascending order

Assume: we already have the setup as below:

Typedef struct node node;

struct node {

int data;

node \*next;

}

int insert\_asc(node\*\* phead, int value) ;

/\* return 1 if inserting is succeeded; return 0 otherwise \*/

|  |
| --- |
| int insert\_asc(node\*\* phead, int value) {  node \*\*tracer;  node \*newNode = malloc(sizeof(node));  if (newNode == 0)  return 0;  newNode->data = value;  for (tracer = phead; \*tracer != 0; tracer = &(\*tracer)->next)  {  if ((\*tracer)->data >= value)  {  break;  }  }  newNode->next = \*tracer;  \*tracer = newNode;  return 1; } |

Task 4: to define a function (named insert\_end) to insert a given value into the linked list at the end

[Hint: 1) loop through the list to find the current last node, 2) may need a double pointer]

Assume: we already have the setup as below:

typedef struct node node;

struct node {

int data;

node \*next;

}

/\* return 1 if inserting is succeeded; return 0 otherwise \*/

|  |
| --- |
| int insert\_end (node\*\* phead, int value){  node \*newNode = malloc(sizeof(node));  if (newNode == 0)  return 0;  newNode->data = value;  newNode->next = 0;  if (\*phead == 0)  {  \*phead = newNode;  return 1;  }  node \*p;  for (p = \*phead; p->next != 0; p = p->next)  ;  p->next = newNode;  return 1;  } |

Task 5: To define a function named remove to remove all nodes with same value as the input

Assume: we already have the setup as below:

Typedef struct node node;

struct node {

int data;

node \*next;

}

Eg, before remove(), the list as 1 -> 2 -> 3 -> 2 -> 1

after calling remove(&head, 2); the list as 1 -> 3 -> 1

/\* return 1 if removing is succeeded; return 0 otherwise \*/

|  |
| --- |
| int remove (node\*\* phead, int value) {  int removed = 0;  for (node \*\*p = phead; \*p != 0;) {  if ((\*p)->data == value) {  node \*nodeToBeRemoved = \*p;  \*p = nodeToBeRemoved->next;  free(nodeToBeRemoved);  removed = 1;  } else {  p = &(\*p)->next;  }  }  return removed;  } |