Week 2 PT 2

**1. Write a statement or set of statements to accomplish each of the following. Assume that all the manipulations occur within the main function (therefore, no addresses of pointer variables are needed).**

**Use the following structure definition to answer questions 2.a.b.c.d and e**

*struct bankEmployee {*

*char name[20];*

*int salary;*

*struct bankEmployee \*next;*

*};*

*typedef struct bankEmployee BANKEmployee;*

*typedef BANKEmployee \*BANKEmployeePtr;*

1. **Create a pointer to the start of the list called startPtr, the list is currently empty.**

*ANS>*

BANKEmployeePtr startPtr = NULL;

1. **Create a new node of type BANKEmployee that's pointed to by pointer newPtr of type BANKEmployeePtr. Assign "Justin" as the name and 1000 as the salary. Make startPtr to point to this node. Provide any necessary declarations and statements.**

*ANS>*

BANKEmployeePtr newPtr;

if (newPtr != NULL)

{

newPtr = malloc (sizeof(BANKEmployee));

strcpy(newPtr->name, "Justin");

newPtr->salary =1000;

newPtr->startPtr = NULL;

}

Use diagram to show the startPtr and the new node.

|  |  |
| --- | --- |
| Justin | 1000 |

|  |  |
| --- | --- |
| data | next |

\*startPtr

1. **Assume that the list pointed to by startPtr currently consists of 2 nodes one containing "Justin" and one containing "Sam”. Assume Sam’s salary as 999 and the nodes are in alphabetical order.**

**Use diagrams to show the insertion of the following nodes with these data for name and salary:**

**"Antony" 200**

**"Tony" 300**

**"Peter" 400**

**Provide C programming statements to insert the above nodes**

**Use pointers previousPtr; currentPtr and newPtr to perform the insertions; State what previousPtr and currentPtr point to before each insertion. Assume that newPtr always points to the new node, and that the new node has already been assigned the data.**

*ANS>*

To insert "Antony":

previousPtr is NULL, and currentPtr points to the first node in the list.

newPtr->nextPtr = currentPtr;

startPtr = newPtr;

To insert "Tony"

previousPtr points to the last node in the list (containing "Sam") and currentPtr is NULL

newPtr->nextPtr = currentPtr; /\*or newPtr->nextPtr = NULL\*/

previousPtr->nextPtr = newPtr;

To insert "Peter"

previousPtr points to the node containing "Justin" and currentPtr points to the node containing "Sam"

newPtr->nextPtr = currentPtr;

previousPtr->nextPtr = newPtr;

|  |
| --- |
| head |

|  |  |
| --- | --- |
| data | next |

|  |  |
| --- | --- |
| data | next |

|  |  |
| --- | --- |
| data | next |

|  |  |
| --- | --- |
| Tony | 300 |

|  |  |
| --- | --- |
| Antony | 200 |

|  |  |
| --- | --- |
| Peter | 400 |

1. **Write a while loop that prints the data in each node of the list. Use pointercurrentPtr to move along the list.**

*ANS>*

currentPtr = startPtr;

currentPtr = startPtr;

while (currentPtr != NULL)

{

printf ("Last name = %s\nGrade = %6.2f\n", currentPtr->lastName,

currentPtr->grade);

currentPtr = currentPtr->nextPtr;

}

**e. Write a while loop that deletes all the nodes in the list and frees the memory associated with each node. Use pointer currentPtr and pointer tempPtr to walk along the list and free memory, respectively.**

*ANS>*

currentPtr = startPtr;

while (currentPtr != NULL)

{

tempPtr = currentPtr;

currentPtr = currentPtr->nextPtr;

free (tempPtr);

}

startPtr = NULL;

**2. Create a linked list using the following structure**

*struct studentname {*

*char letter;*

*struct studentname \*next;*

*};*

*typedef struct studentname STUDENTName;*

*typedef STUDENTName \*STUDENTNamePtr;*

**Create a linked list manually (without using any loops or recursive functions or any functions) that contains five nodes where the data part (structure element letter) of the nodes should be the first five letters of yourlast name. One letter will go to one node and the node insertion should happen one after another in alphabetical order. When you insert a new node, it should be in the right place to get alphabetical order as shown in previous task. If your last name doesn’t have five letters fill the remaining nodes with letters from your first name starting from the first letter (Eg: if your name is Devin Ly, then nodes will contain L, y, D, e and v).**

**Example:**

**Assume your name is Ricky Ponting; take the first five letters of the last name, which is Ponti so the insertion order is**

*newptr = new STUDENTName;*

*newptr -> letter = 'P ';*

**.**

**.**

**.**

**o**

**.**

**.**

**n**

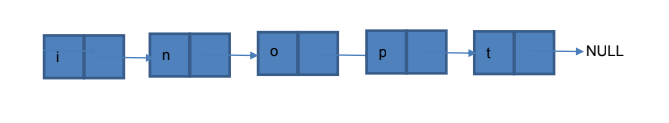
**.**

**t**

**.**

**I**

**and when you print the linked list it should be like as shown below (Note, you are not using any sorting to get this in alphabetical order, you manually insert nodes in the right place to get it in alphabetical order).**



**Code:**

#include <iostream>

using namespace std;

struct studentname {

char letter;

struct studentname \*next;

};

int i;

struct studentname \*head,\*curr, \*c, \*c1;

int main() {

char name[50]="PONTING";

head->letter = name[0];

if(name[i]=='\0')

{

i=0;

}

curr = new struct studentname;

curr->letter = name[i];

if(curr->letter < head->letter) {

// set current as head

curr->next = head;

head = curr;

} else { c = head;

c1 = c->next;

while( c1->letter < name[i])

{

c = c->next;

if(c1->next==NULL)

{

break; // break if end of string

}

else

{

c1=c1->next;

}

}

c->next = curr;

curr->next = c1;

}

}

c = head;

i=0;

while(i<5) {

cout<<c->letter<<" ";

c = c->next;

i++;

}

return(0);

}

**Output:**

