**Section (I): Tracing Problems (Total: 3 marks)**

*In the following tracing question use the following definition for the nodes in the list:*

**class CListNode**

**{**

**public:**

**int info;**

**CListNode \*pNext;**

**};**

**A**

**4**

**3**

**1**

**2**

**Write one statement to do the following:-**

* Display the info of node (**4**) .

Cout<<A->pNext->pNext->pNext->info;

**True or False :**

* A 🡪 pNext 🡪 pNext == A 🡪 pNext 🡪 pNext 🡪 pNext 🡪 pNext

TRUE

* Make node (**2**) points to node (**3**).

A->pNext->pNext->pNext=A;

**Section (II): Algorithms (Total: 7 marks)**

**Algorithm 1: (7 marks)**

*In the following Algorithm question use the following definition for the nodes in the list:*

**class CNode**

**{public: int info;**

**CNode \*pNext;**

**};**

Write  **void Attach\_After\_Max** **( CNode \*pNN )** function that do the following:

* Case 1: The info of (pNN) is **greater** than all values in the List 🡪 then attach it at tail as normal.
* Case 2: The info of (pNN) is **Not** greater than all values in the List 🡪 then attach it after the largest node in the list.
* **Note:** There is a third pointer (pExtra) in your list additional to the pHead & pTail.
* Your function should be in **O(1)**

e.g.

|  |  |  |  |
| --- | --- | --- | --- |
| **Attach… (40)**  pTail  pTail  pTail  pTail  pTail  pTail  pTail  pExtra  pHead  pExtra  pHead  pTail  pHead  pExtra | **Attach… (30)** | | **Attach… (15)** |
| **Attach… (25)**  pExtra  pHead  pExtra  pHead  pExtra  pHead  pExtra  pHead  pExtra  pHead | | **Attach… (70)** | |
| **Attach…** (**50**)  **Attach… (60)**    **Attach… (19)**    #include <iostream>  using namespace std;  class CNode  {  public:  int info;  CNode\* pNext;  };  class CList  {  public:  CNode\* pHead;  CNode\* pTail;  CNode\* pExtra;  CList()  {  pHead = NULL;  pTail = NULL;  pExtra = NULL;  }  void Attach\_After\_Max(CNode\* pnn)  {  if (pHead == NULL)  {  pHead = pnn;  pTail = pnn;  pExtra = pnn;    }  else  { //pExtra point to max number  if (pnn->info < pExtra->info)  {  if (pExtra->pNext == NULL)  {  pExtra->pNext = pnn;  pTail = pnn;  pnn->pNext = NULL;  }  else  {  pnn->pNext = pExtra->pNext;  pExtra->pNext = pnn;  }  }  else  {  //pExtra will point to the new max valued when found  pTail->pNext = pnn;  pExtra = pnn;  pTail = pnn;  pnn->pNext = NULL;  }  }  }  ~CList()  {  CNode\* pTrav = pHead;  while (pHead != NULL)  {  pHead = pTrav->pNext;  pTrav->pNext = NULL;  delete pTrav;  pTrav = pHead;  }  }  };  void main()  {  CNode\* pnn;  CList L;  int N;  cin >> N;  for (int i = 0; i < N; i++)  {  pnn = new CNode;  cin >> pnn->info;  pnn->pNext = NULL;  L.Attach\_After\_Max(pnn);  }  CNode\* pTrav = L.pHead;  while (pTrav != NULL)  {  cout << pTrav->info<<"\n";  pTrav = pTrav->pNext;  }  } | | | |

**Section (III): Problem Solving (Total: 10 marks)**

*In the following question use the following definition for the nodes in the list:*

info

**class CNode**

pNext

**{**

**public:**

**int info;**

**CNode \* pNext;**

pDown1

**CNode \* pDown1;**

pDown2

**CNode \* pDown2;**

**};**

Write a main function to do:

* + Read 20 Lists from the user.

(Assume all lists with the same length) & (assume the length is odd)

* + For 1st List connect the middle node , to the both (2nd node & node before the last ) but in the 2nd List.

Repeat this connection between the 3rd list & 4th list.

And so on.

Head

Head

Head

L

Head

Head

Head

* Cut the connected intervals & past them into a New List **(NL)**.

Head

Head

Head

Head

Head

Head

L

NL

Head

**It runs but did not check with example**

#include <iostream>

using namespace std;

class CNode

{

public:

int info;

CNode\* pNext;

CNode\* pDown1;

CNode\* pDown2;

};

class CList

{

public:

CNode\* pHead;

CNode\* pTail;

CList()

{

pHead = NULL;

pTail = NULL;

}

void Attach(CNode\* pnn)

{

if (pHead == NULL)

{

pHead = pnn;

pTail = pnn;

}

else

{

pTail->pNext = pnn;

pTail = pnn;

}

}

~CList()

{

CNode\* pTrav = pHead;

while (pHead != NULL)

{

pHead = pTrav->pNext;

pTrav->pNext = NULL;

delete pTrav;

pTrav = pHead;

}

}

};

void main()

{

int N,nn,half,ct=0,k=0;

CList L[20];

CList newL;

CNode\* pnn;

cout << "enter n \n";

cin >> N;

for (int i = 0; i < N; i++)

{

cin >> nn;

for (int k = 0; k < nn; k++)

{

pnn = new CNode;

cout << "enter info \n";

cin >> pnn->info;

pnn->pNext = NULL;

pnn->pDown1 = NULL;

pnn->pDown2 = NULL;

L->Attach(pnn);

}

}

for (int i = 0; i < N; i+=2)

{

CNode\* pTrav = L[i].pHead;

CNode\* pB = L[i].pHead;

while (pTrav != NULL)

{

pTrav = pTrav->pNext;

ct++;

}

half = (ct / 2) + 1;

pTrav = L[i].pHead;

while (pTrav != NULL && k!=half)

{

pB = pTrav;

pTrav = pTrav->pNext;

}

pTrav->pDown1 = L[i + 1].pHead->pNext;

CNode\* pTrav2 = L[i + 1].pHead;

while (pTrav2->pNext->pNext != NULL)

{

pTrav2 = pTrav2->pNext;

}

pTrav->pDown2 = pTrav2;

pB = pTrav->pNext;

pTrav->pNext = NULL;

L[i + 1].pHead->pNext = L[i + 1].pTail;

pTrav->pDown2->pNext = NULL;

if (newL.pHead == NULL)

{

newL.pHead = pTrav;

newL.pTail - pTrav;

}

else

{

newL.pTail->pNext = pTrav;

newL.pTail = pTrav;

}

}

}