الفصل الخامس : القوائم المترابطة
Linked Lists

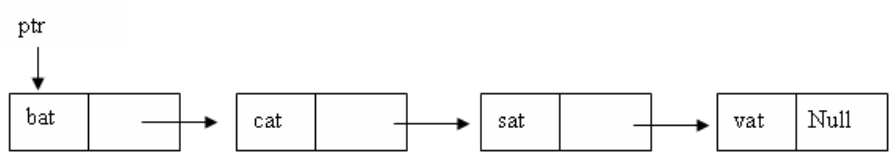
تعتبر المتجهات (arrays) – سواء المرتبة العناصر أو غير المرتبة العناصر– تحتوي على العديد من العيوب و نقاط الضعف كشكل من بنى تخزين المعطيات . ففي المتجهة غير المرتبة العناصر (unordered array) ، تكون عملية البحث بطئيه . و في المتجهة المرتبة العناصر (ordered array) ، تكون عملية الإدخال هي البطيئة . و كذلك العيب المشترك في كلا البنيتين هو بطء عملية الحذف و كذلك محدودية الحجم بمدى معين لا يمكن تغييره بعد إنشاء البنية .

و قد تفادي بنى القوائم المترابطة (Linked list) هذه المشكلات بعد ذلك ؛ الأمر الذي جعله يأخذ المرتبة الثانية بين بنى المعطيات بعد المتجهات من حيث شيوع الاستخدام . و تعتبر بنى القوائم المترابطة بنى متعددة الاستخدامات تصلح للعديد من التطبيقات الخاصة بتخزين البيانات . بل و من الممكن أيضا" أن تحل محل المتجهات في استخدامها كقاعدة لتوضيح أساليب بنى المعطيات الأخرى مثل : المكدسات (stacks) و الأرتال (Queues) . و تكون بذلك قد أجريت بعض التعديلات التي من شأنها رفع مستوى الكفاءة في عمليتي الحذف و الإدخال .
و لكن ، ليس معنى ذلك أن القوائم المترابطة تقدم حلولاً لجميع المشكلات المتعلقة بتخزين البيانات . و لكن كل ما في الأمر أنها تقدم نموذجا" متعدد الاستخدامات ينطوي على مفهوم أبسط من مفاهيم بنى المعطيات الأخرى مثل الأشجار (trees) .

٥–١ القوائم الأحادية الترابط Singly Linked Lists

القوائم المترابطة ترسم بشكل متتالية مرتبة من العقد (nodes) ، كل عقدة تتضمن حقلين : حقلاً للبيانات (data) و حقلاً للربط (link) يشير إلى عقدة يتضمن حقل البيانات فيها العنصر التالي من القائمة . كما أن حقل الربط في آخر عقدة يساوي الصفر (null) ،

ويسمى حقل الربط الذي يشير إلى أول عقدة في القائمة باسم القائمة ptr ، و تمثل الروابط بأسهم . الشكل(٥-١) يمثل بنية قائمة مترابطة تتضمن الكلمات : bat, cat, sat, vat

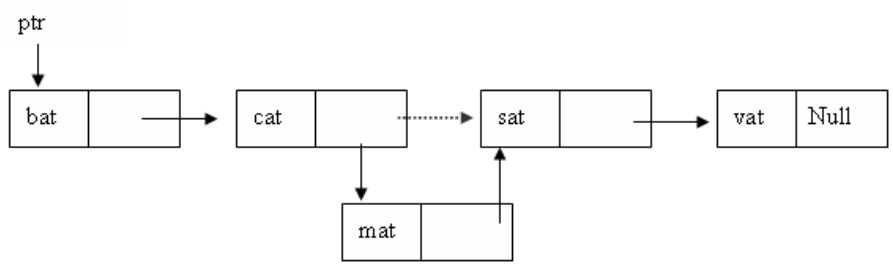


الشكل (٥-١) : قائمة مترابطة

تعتبر القوائم المترابطة من بنى المعطيات المتعددة الاستخدامات التي تتميز بالسرعة في عمليتي الحذف و الإدخال. لحشر الكلمة mat في القائمة المعطاة بالشكل ٥-١ بين الكلمتين cat و sat نحتاج :

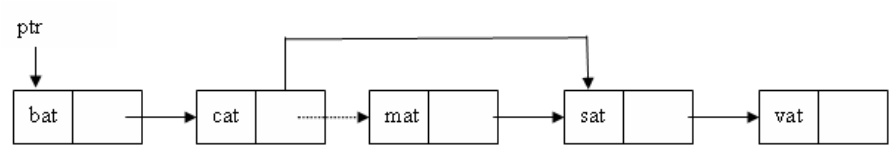
١. إنشاء عقدة جديدة ، و تعنون بـــ paddr .
٢. نجعل حقل البيانات في هذه العقدة لـــ mat.
٣. نجعل حقل الربط للعقدة paddr يشير إلى العقدة التي تتضمن الكلمة sat.
٤. نجعل حقل ربط العقدة cat يشير إلى العقدة paddr.

الشكل (٥-٢) يمثل عملية حشر الكلمة mat بين الكلمتين cat, sat حيث تمثل الأسهم المنقطة روابط قديمة ، كما نلاحظ أن حشر العنصر mat في القائمة لا يستدعي تحركاً لأي عنصر من عناصر القائمة .



الشكل(٥-٢)يمثل عملية حشر الكلمة mat بعد cat

لنفرض الآن أننا نرغب بحذف العنصر mat من القائمة . في هذه الحالة نحتاج لمعرفة العنصر الذي يسبق العنصر mat مباشرة و الذي هو العنصر cat . نجعل حقل رابط العقدة cat يشير إلى حقل ربط العقدة mat كما هو موضح بالشكل(٥-٣) .



الشكل(٥-٣) يمثل عملية حذف الكلمة mat من القائمة

من الدراسة الموجزة للقوائم المترابطة نجد أنه :

١. لتمثيل بنية عقدة في القائمة المترابطة ، نستخدم مفهوم التراكيب ذات المرجعية الذاتية (self-referential structures)
٢. لإنشاء عقد جديدة في القائمة ، نستخدم الدالة malloc() لمعالجة هذه العملية .
٣. لحذف عقدة من القائمة ، نستخدم الدالة free() لمعالجة هذه العملية .

يتم الإعلان عن قائمة مترابطة من الكلمات طول كل كلمة 4 بالشكل :

```
typedef struct list-node  *list-pointer
typedef struct list-node {
      char data[4];
      list-pointer link;};
       list-pointer ptr =NULL;
```

هذه الإعلانات تتضمن مثالاً من التركيب ذات المرجعية الذاتية . حيث عرفنا المؤشر list-pointer إلى التركيب list-node قبل تعريف هذا التركيب ، لأن هذا مسموح في لغة C .

بعد تعريف بنية العقدة ، ننشئ قائمة جديدة فارغة ptr بالعبارة التالية :

```
list-pointer ptr =NULL;
```

لاختبار أن القائمة فارغة ، نستخدم الماكرو IS-EMPTY بالشكل :

```
#define IS-EMPTY(ptr) (!(ptr))
```

لإنشاء عقد جديدة في القائمة المترابطة ، نستخدم الدالة malloc (memory allocation) الموجودة في <alloc.h> بالشكل :

```
Ptr = (list-pointer)malloc(sizeof(list-node));
```

لتحديد استخدام الذاكرة المتاحة ، نستعمل الماكرو IS-FULL مع الدالة ()malloc و الذي يسترجع NULL إذا لم يتوفر ذاكرة متاحة . و يعرف بالصيغة :

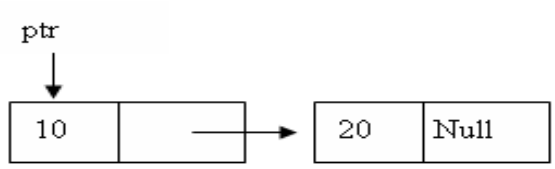
```
#define IS-FULL(ptr) (!(ptr))
```

لوضع الكلمة bat في القائمة ، نستعمل العبارات التالية :

```
Strcpy(ptr->data, "bat");
Ptr->link = NULL;
```

٥–٢ العمليات على القوائم أحادية الترابط

٥–٢–١ إنشاء قائمة مترابطة: يتم إنشاء قائمة مترابطة مكونة من عقدتين –مثلا"– والموضحة بالشكل (٥–٤) ، من خلال الدالة الدالة ()create



الشكل (٥–٤): قائمة مترابطة بعقدتين من الأعداد الصحيحة

أولا" نعرف بنية العقدة بالشكل التالي :

```
typedef struct list-node  *list-pointer
typedef struct list-node {
      int data;
      list-pointer link;};
       list-pointer ptr =NULL;
```

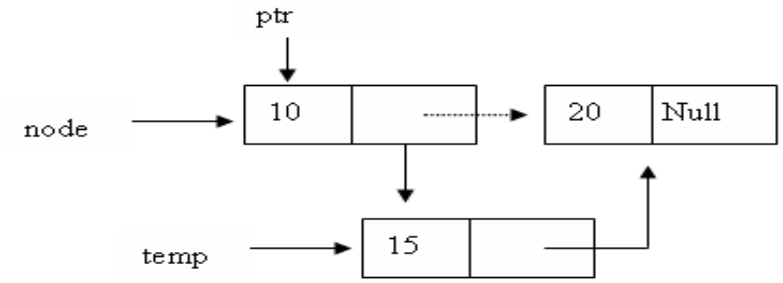
الدالة ()create تنشئ هذه القائمة بالشكل :

```
list-pointer create()
  {
list-pointer first, second;
first = (list-pointer)malloc(sizeof(list-node));
second=(list-pointer)malloc(sizeof(list-node));
second->link=NULL;
second->data=20;
first->data=10;
first->link=second;
return first;
}
```

٥-٢-٢ حشر عقدة في قائمة مترابطة: ليكن ptr المؤشر إلى القائمة المترابطة في الشكل(٥-٤) . و لنحشر عقدة حقل بياناتها 15 بين العقدتين 10 و 20 . الدالة ()insert تنجز هذه المهمة بالصيغة التالية :

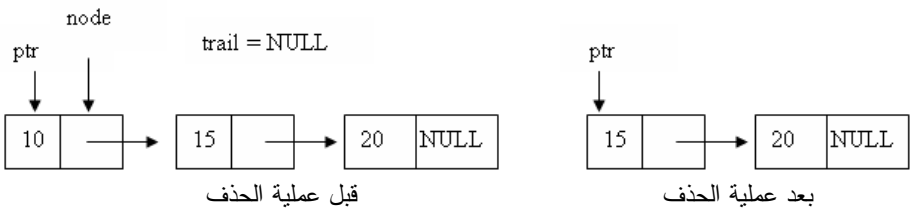
```
void insert(list-pointer *ptr, list-pointer node)
  {
 list-pointer temp;
temp = (list-pointer)malloc(sizeof(list-node));
 if(IS-FULL(temp)){
   fprintf(stderr,"The memory is full\n");
    exit(1);
 }
 temp->data=15;
 if(*ptr){
  temp->link=node->link;
  node->link=temp;
}
else {
temp->link=NULL;
*ptr=temp;
  }
}
```

الشكل(٥-٥) يوضح القائمة من الشكل(٥-٤) بعد حشر temp بين عقدتي first و second .

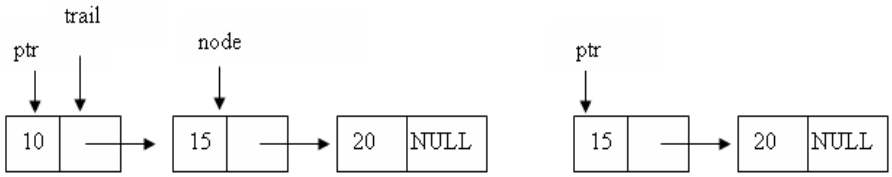


الشكل (٥-٥): قائمة مترابطة بعقدتين بعد استدعاء الدالة insert(&ptr, ptr)

٥-٢-٣ حذف عقدة من قائمة مترابطة : إن حذف عقدة اختيارية من قائمة مترابطة أكثر تعقيدا" من حشر عقدة ، لأن عملية الحذف تعتمد على موضع العقدة . ليكن لدينا ثلاثة مؤشرات : ptr يشير إلى أول القائمة ، و node يشير إلى العقدة المراد حذفها ، و trail يشير إلى العقدة التي تسبق العقدة node مباشرة . الشكل(٥-٦(a)) والشكل(٥-٦(b)) يظهران عملية حذف عقدة من قائمة مترابط .



الشكل(٥-٦(a)): قائمة مترابطة بعد استدعاء الدالة delete(&ptr, NULL, ptr)



بعد عملية الحذف **قبل عملية الحذف**

الشكل(٥-٦(b)): قائمة مترابطة بعد استدعاء الدالة delete(&ptr, NULL, ptr->link)

تنجز الدالة ()delete عملية حذف عقدة كيفية من قائمة مترابطة بالشكل التالي:

```
void delete(list-pointer *ptr, list-pointer trail,
                              list-pointer node)
  {
   if(trail)
     trail->link=node->link;
   else
     *ptr=(*ptr)->link;
   free(node);
  }
```

٥-٢-٤ طباعة قائمة مترابطة: الدالة الإجرائية ()print-list تقوم بطباعة محتوى حقول البيانات في القائمة ، و ذلك بطباعة محتوى حقل البيانات لأول عقدة في القائمة ، و بعد ذلك نبدل المؤشر ptr بعنوان حقل ربطه . نستمر بطباعة حقل البيانات و نتحرك إلى العقدة التالية حتى نصل إلى نهاية القائمة .

```
void print-list(list-pointer ptr)
   {
    printf("The list contains: ");
    for( ; ptr !=NULL; ptr=ptr->link);
      pintf("%4d",ptr->data);
    pintf("\n");
   }
```

٥-٢-٥ دمج قائمتين مترابطتين: بفرض: لدينا قائمتان مترابطتان ptr1 و ptr2 ، لندمج القائمتين في قائمة جديدة ptr1 تتضمن عقد القائمة الأولى و عقد القائمة الثانية من خلال الدالة الإجرائية concatenate () :

```
list-pointer concatenate(list-pointer ptr1,
                         list-pointer ptr2)
  {
   list-pointer temp;
   if(IS-EMPTY(ptr1) return ptr2;
   else{
     if(!IS-EMPTY(ptr2)){
      for(temp=ptr1;temp->link != NULL; temp=temp->link)
          ;
        temp->link=ptr2;
         }
        return ptr1;
       }
     }
```

5-٢-٥ معكوس قائمة مترابطة: بفرض لدينا قائمة مترابطة lead . نحصل على معكوس القائمة المعطاة من خلال الدالة الإجرائية invert() :

```
list-pointer invert(list-pointer lead)

  {
   list-pointer middle, trail;
   middle = NULL;
   while(lead != NULL) {
     trail=middle;
     middle = lead ;
     lead =lead->link;
     middle->link=trail;
      }
    return middle;
     }
```

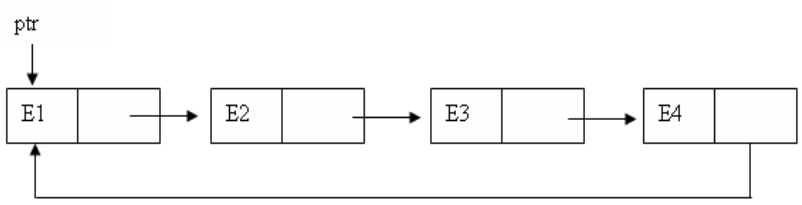
٥-٢-٦ حذف قائمة مترابطة: بفرض لدينا قائمة مترابطة ptr . يتم حذف القائمة من خلال حذف جميع عقدها و ذلك بواسطة الدالة الإجرائية erase() التالية :

```
void erase(list-pointer *ptr)

  {
   list-pointer temp;
   while(*ptr) {
     temp=*ptr;
     *ptr =(*ptr)->link;
     free(temp)
      }
       }
```

٥-٣ القوائم المترابطة الدائرية Circularly Linked Lists

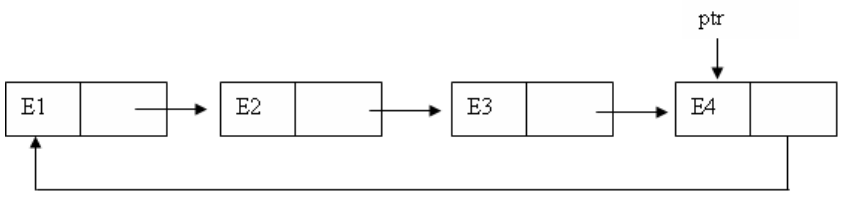
بدلاً من أن يأخذ حقل الربط في آخر عقدة في قائمة أحادية الترابط القيمة NULL نجعل حقل الربط يؤشر إلى أول عقدة في القائمة كما في الشكل (٥-٧) .



الشكل (٥-٧): قائمة مترابطة دائرية

٥-٤ العمليات على القوائم الدائرية

٥-٤-١ حشر عقدة في قائمة دائرية : بفرض لدينا قائمة دائرية كما في الشكل (٥-٧) . لإضافة عقدة في مقدمة القائمة . يجب تغيير حقل ربط العقدة E4 ، هذا يعني إنه يجب أن نتحرك بطول القائمة للوصول إلى آخر عقدة فيها لتغيير قيمة حقل ربطها ، للتغلب على هذه المشكلة نجعل اسم القائمة الدائرية يشير إلى آخر عقدة فيها بدلا" من أول عقدة فيها كما في الشكل(٥-٨) .



الشكل(٥-٨): قائمة مترابطة دائرية

الدالة الإجرائية insert-front() تقوم بإضافة عقدة node في بداية القائمة .

```
void insert-front(list-pointer *ptr, list-pointer node)
   {
    if(*ptr == NULL){
      *ptr=node;
      node->link=node;
         }
    else{
       node->link=(*ptr)->link;
       *ptr)->link=node;
       }
    }
```

ملاحظة: لإضافة العقدة node في نهاية القائمة ، نضيف العبارة node=(*ptr)*إلى عبارة else في الدالة insert-front()

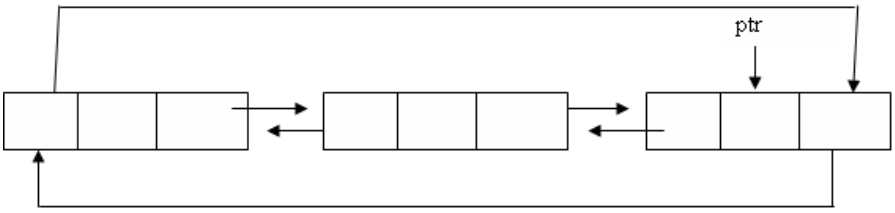
٥-٤-٢ طول قائمة دائرية : بفرض لدينا قائمة دائرية ptr ، يحسب طول هذه القائمة بتعريف عداد count ، نزيده واحدا" كلما مسحنا عقدة إضافية حتى نصل إلى نهاية القائمة .

```
int length(list-pointer ptr)
   {
    list-pointer temp;
    int count=0;
    if(ptr !=NULL){
      temp=ptr;
      do{
        count++;
        temp=temp->link;
```

```
  }while(temp!=ptr);
 }
return count;
}
```

٥-5 القوائم المضاعفة الارتباط Doubly Liked Lists

يمكّن التمثيل المقدم للقوائم المترابطة من مسح عقد القائمة في اتجاه واحد . في الكثير من التطبيقات نحتاج إلى مسح عقد القائمة ابتداء" من آخر عنصر ، و انتهاء" بأول عقدة فيها ، لذلك نضيف من أجل كل عقدة مؤشراً إضافياً يدلنا على العقدة السابقة في القائمة ، فنحصل بذلك على قائمة مضاعفة الارتباط ، يوضح الشكل(٥-٩) قائمة مضاعفة الارتباط .



الشكل (٥-٩) : قائمة مضاعفة الارتباط

بنية العقدة في القائمة مضاعفة الترابط تملك على الأقل ثلاثة حقول : حقلاً للربط اليساري (llink)، و حقلاً للبيانات (data) ، و حقلاً للربط اليميني (rlink). يتم الإعلان عن بنية العقدة بالصيغة الآتية:

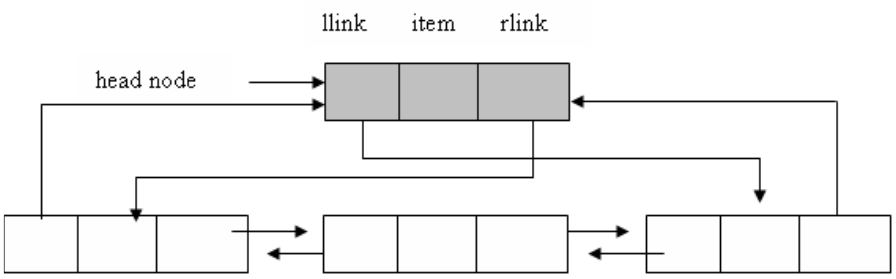
```
typedef struct node  *node-pointer
typedef struct node {
      node-pointer llink
      element data;
      node-pointer rlink;
       };
```

ملاحظة: القائمة مضاعفة الترابط يمكن أن تكون دائرية كما في الشكل (5-٩) و أحيانا تكون غير دائرية . إذا كان ptr مؤشراً إلى أي عقدة في قائمة مضاعفة الترابط عندئذ يكون :

Ptr=ptr->llink->rlink= ptr->rlink->llink

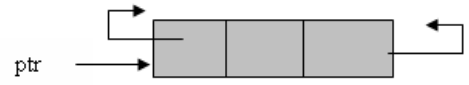
5-6 العمليات على قوائم مضاعفة الترابط

لتسهيل تنفيذ العمليات على القوائم المضاعفة نضيف إلى القائمة عقدة رأسية head node ، لا يتضمن حقل بياناتها أي معلومات . كما هو موضح في الشكل(5-١٠) .



الشكل(٥-١٠) : قائمة دائرية مضاعفة الترابط بعقدة رأسية .

كما أن القائمة الفارغة تتضمن فقط العقدة الرأسية و الموضحة بالشكل (٥-١١).

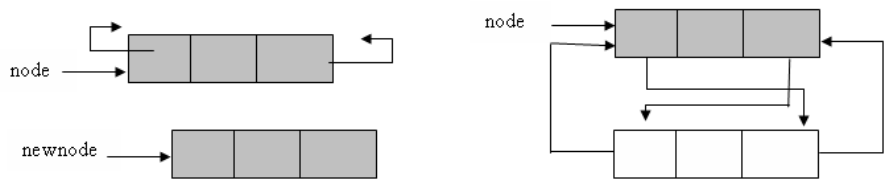


الشكل(٥-١١): قائمة دائرية مضاعفة الترابط فارغة بعقدة رأسية

٥-٦-١ حشر عقدة في قائمة مترابطة التضاعف: بفرض لدينا قائمة مضاعفة الترابط مكونة من عقدتين node, newnode ، العقدة node يمكن أن تكون عقدة رأسية أو عقدة داخلية في القائمة . الدالة dinsert() تنجز عملية الحشر بزمن تنفيذ ثابت و الموضحة بالشكل التالي :

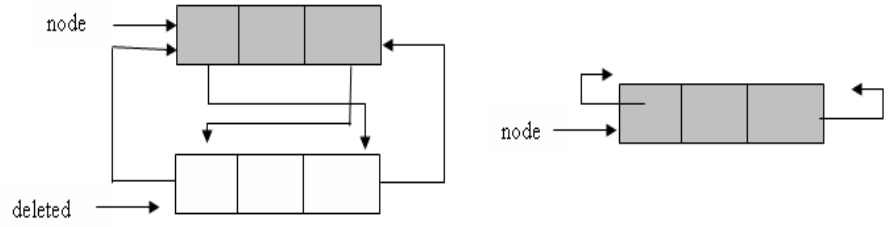
```
void dinsert(node-pointer node, node-pointer newnode)
   {
newnode->llink=node;
newnode->rlink=node->rlink;
node->rlink->llink=newnode;
node->rlink=newnode;
   }
```

يوضح الشكل(٥-١٢) عملية الحشر عندما تمثل العقدة node عقدة رأسية في القائمة الفارغة .



الشكل(٥-١٢): الحشر في قائمة دائرية فارغة و مضاعفة الترابط

٥-٦-٢ حذف عقدة في قائمة مضاعفة الترابط: دالة الحذف ddelete() تحذف العقدة deleted من القائمة node ، و ذلك عن طريق تغيير حقول الربط للقوائم التي تسبق العقدة المراد حذفها (deleted->llink->rlink) ومن ثم (deleted->rlink->llink).
الشكل(٥-١٣) يبين عملية الحذف من قائمة دائرية مضاعفة الترابط مكونة من عقدة واحدة .



الشكل (٥-١٣): الحذف من قائمة دائرية مضاعفة الترابط

```
void ddelete(node-pointer node, node-pointer deleted)
   {
if(node==deleted)
  printf("Deletion of head node not permitted");
else{
  deleted->llink->rlink=deleted->rlink;
  deleted->rlink->llink=deleted->llink;
  free(deleted);
    }
 }
```

٥–٧ تمارين الفصل الخامس

تمرين١: بفرض لديك قائمة مترابطة من الأعداد الصحيحة .

١. اكتب دالة إجرائية search() تقوم بالبحث عن عدد صحيح num . إذا كان num موجوداً في القائمة ، فالدالة search ترجع مؤشرا" إلى العقدة التي تتضمن num ، و غير ذلك ترجع NULL .

٢. اكتب دالة إجرائية delete() تقوم بحذف العقدة التي تتضمن العدد num من القائمة.

٣. اكتب دالة إجرائية length() تقوم بحساب عدد العقد في القائمة .

تمرين٢: لتكن القائمتان المترابطتان $x=(x_1,x_2,\ldots,x_n)$ و $y=(y_1,y_2,\ldots,y_m)$ و المرتبتان تصاعدياً وفقاً لقيم حقول بياناتهما . قدم خوارزمية تقوم بدمج القائمتين في قائمة مترابطة z و مرتبة تصاعدياً ، ثم احسب التعقيد الزمني للخوارزمية المقدمة .

تمرين٣: لتكن القائمتان المترابطتان $list1=(x_1,x_2,\ldots,x_n)$ و $list2=(y_1,y_2,\ldots,y_m)$. اكتب دالة إجرائية تدمج القائمتين للحصول على القائمة المترابطة $list3=(x_1,y_1,x_2,y_2,\ldots,x_m,y_m,x_{m+1},\ldots,x_n)$ إذا كان $m \le n$ و $list3=(x_1,y_1,x_2,y_2,\ldots,x_n,y_n,y_{n+1},\ldots,y_m)$ إذا كان $m > n$.

تمرين٤: بفرض لديك كثير الحدود من الشكل:

$A(x) = a_m x^{e_m} + a_{m-1}x^{e_{m-1}} + \ldots\ldots + a_1 x^{e1}$ حيث a_i تمثل أمثال كثير الحدود غير الصفرية و e_i تمثل قوى صحيحة غير سالبة و تحقق :

$e_m > e_{m-1} > \ldots\ldots > e_2 > e_1 \ge 0$ و المطلوب :

١. أوجد بنى المعطيات اللازمة لتعريف بنية عقدة في قائمة مترابطة تمثل كثير الحدود المعطى

٢. قدم خوارزمية تقوم بجمع كثيري حدود ممثلين بقائمتين مترابطتين . و احسب زمن تنفيذها .

٣. قدم خوارزمية تقوم بضرب كثيري حدود ممثلين بقائمتين مترابطتين . و احسب زمن تنفيذها .

٤. بفرض أن a مؤشر إلى قائمة مترابطة تمثل كثير حدود . اكتب دالة إجرائية لتقييم كثير الحدود a عند قيمة حقيقية معطاة للمتغير x .

تمرين٥: المصفوفة المثقبة (sparse matrix) : هي مصفوفة أغلب عناصرها أصفار ، على سبيل المثال: $a=\begin{bmatrix} 0 & 0 & 11 & 0 \\ 12 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 \\ 0 & 0 & 0 & -15 \end{bmatrix}$ و المطلوب :

١. أوجد بنى المعطيات اللازمة لتعريف بنية عقدة في قائمة مترابطة تمثل المصفوفة المثقبة a .

٢. قدم خوارزمية تقوم بجمع مصفوفتين مثقبتين ممثلتين بقائمتين مترابطتين . و احسب زمن تنفيذها .

٣. قدم خوارزمية تقوم بضرب مصفوفتين مثقبتين ممثلتين بقائمتين مترابطتين . و احسب زمن تنفيذها .

٤. قدم خوارزمية تقوم بحساب منقول مصفوفة مثقبة.