5. القوائم المترابطة

5.1. تمهيد

في البرمجة، للتعامل مع بيانات من نفس النوع (مثلا معلومات طلبة) نحتاج الى الجداول والتي تمثل مفهوما مهما في اي لغة برمجة، حيث تتميز الجداول بسرعة الوصول الى عناصرها، الا ان لها عيبين هما : 1) يجب ان تكون عناصر الجدول متجاورة في الذاكرة. 2) لا يمكن إدراج او إزالة عناصر في الجدول دون إعادة انشاء الجدول من جديد، لذلك نحتاج الى بنية أخرى تعرف بالقائمة المتصلة.

5.2. تعريف

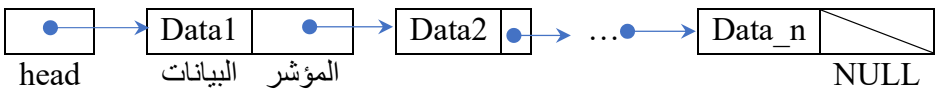
القائمة المترابطة او المتصلة (Linked Lists) هي عبارة عن بنية معطيات تراجعية تتكون من مجموعة من سجلات من نفس النوع مترابطة مع بعضها البعض بواسطة المؤشرات، حيث يمكن لهاته السجلات ان تكون في مواقع غير متجاورة في الذاكرة. وتتكون القوائم المترابطة من عناصر (سجلات، عقد او خلايا)، ويحتوي كل عنصر منها على حقل او عدة حقول لتخزين البيانات ومؤشر (رابطة) إلى العنصر التالي في القائمة.

وتسمح هذه البنية بتعديل بعدها وذلك بالإدراج او إزالة العناصر من أي موضع في القائمة، وللوصول إلى أي عنصر من عناصر القائمة يجب الانطلاق من راسها وتصفح جميع العناصر التي قبله والذي يشكل عبئا من ناحية الوقت مقارنة بالجداول. (لذلك نقول عنها انها بنية خطية مقارنة بالجدول الذي يسمى ببنية ذات وصول عشوائي).

5.3. التمثيل

يتم تمثيل العقدة في لغة C باستخدام البنى او السجلات (structures) اما الراس فيمثل بواسطة مؤشر. لتسهيل عملية الشرح نقوم بتعويض جميع حقول البيانات (مثلا معلومات طالب الاسم اللقب تاريخ ...) بحقل واحد data من نوع عدد صحيح.

يوضح الشكل التالي بنية القوائم المترابطة:



5.4. التصريح بالعناصر

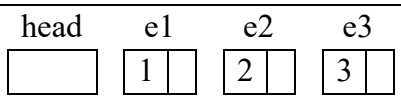
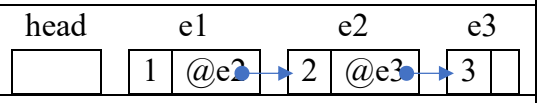
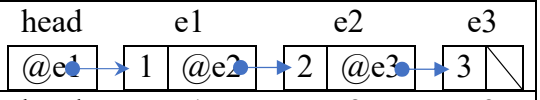
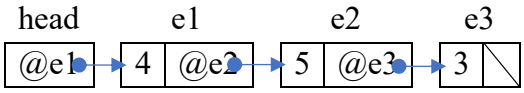
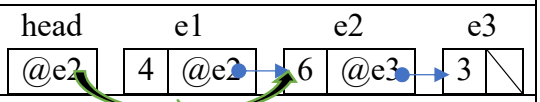
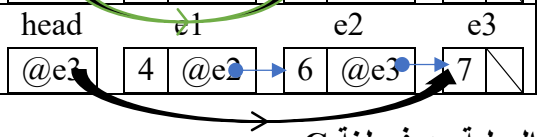
C	Algorithme	الشرح

`typedef struct Cell {`
`  int data;`
`  struct Cell* next;`
`} Cell;`
	Cell structure
	data:entier
	next :^Cell
	fin_structure
		data تمثل البيانات المخزنة في القائمة (حيث يمكن تعويضها باي متغيرات أخرى مثل الاسم اللقب ...)
		next عبارة عن مؤشر يحمل عنوان العنصر التالي او NULL في حالة لا يشير الى أي عنصر.

التصريح بنوع الرأس

typedef Cell* List;	type List: ^Cell	هاته تعني ان List هي نفسها *Cell.

مثال

`List head;`	var head: List	متغير بسيط من نوع مؤشر يشير الى اول عنصر head
`Cell e1, e2,e3 ;`	e1, e2, e3:Cell	3 متغيرات مركبة من نوع بنية Cell
`e1.data=1;e2.data=2;` `e3.data=3;`	e1.data←1 e2.data←2 e3.data←3	
`e1.next=&e2;` `e2.next=&e3;`	e1.next←@e2 e2.next←@e3	
`e3.next= NULL;` `head=&e1;`	e3.next← NULL head←@e1	
`head->data=4;` `head->next->data=5;`	head^.data←4 (head^. next)^.data←5	
`head= head->next;` `head->data=6;`	head← head^.next; head^.data←6;	
`head= head->next;` `head->data=7;`	head← head^.next; head^.data←7;	

العملية <- في لغة C

بما ان head يشير الى e1 فان (*head) و e1 هما نفس المتغير لذلك يمكن استعمال (*head).next بدل e1.next

```
e1.next ⇔ (*head).next ⇔ head->next
e1.data=5; ⇔ (*head).data=5; ⇔ head->data=5;
```

في لغة C نستعمل <- بدل **.(*head)** للوصول الى حقول البنية التي يشير اليها head

ملاحظة الكتابة head.data خاطئة لان head عبارة عن مؤشر وليس بنية.

head و e1.next و e2.next و e3.next في 4 عبارة عن مؤشرات من نفس النوع لذلك يمكن القيام بعملية الاسناد فيما بينها.

العنصر الأخير

لا يشير العنصر الأخير في القائمة الى أي عنصر اخر لذلك يتم اسناد NULL لnext وتستعمل اثناء التنقل في القائمة لمعرفة هل تم الوصول الى العنصر الأخير ام لا.

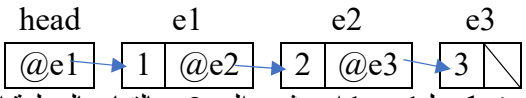
```
e3->next= NULL;
```

head= NULL; عبارة عن قائمة فارغة

التنقل عبر عقد القوائم المترابطة

يوضح المثال التالي طريقة التنقل عبر العناصر في القائمة

لنفرض انه لدينا القائمة التالية



بما ان e1.next يحمل عنوان e2 فيمكن لـhead ان يشير الى e2 بالقيام بالعملية التالية head= e1.next وبما ان head يشير الى e1 فان e1.next⇔head->next وبالتالي

```
head= &e2⇔ head= e1.next⇔ head= head->next
```

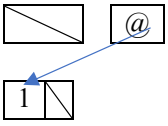
اذن للتنقل من عقدة الى التي تليها نستعمل head= head->next

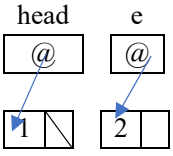
while (head!= NULL){
//do something
head = head -> next;
}
	TQ (head≠NULL)
	faire
	//faire qqs chose
	head←head^.next
	FTQ
		للوصول الى جميع عناصر القائمة نكرر العملة الى ان يأخذ head قيمة next لآخر عقدة وهي NULL.
		ملاحظة while (head) ⇔ while (head!= NULL)

5.5. الانشاء

لإنشاء القائمة نقوم بحجز الذاكرة بطريقة ديناميكية انطلاقا من متغير بسيط من نوع مؤشر. لنفرض انه لدينا قائمة فارغة ;head= NULL ولإنشاء عنصر جديد نستعمل allouer (malloc).

List e, head= NULL;	var e, head: List head← NULL	head e	تم انشاء قائمتين (مؤشر*Cell)

List e = malloc(
sizeof(Cell));
e->data=1;
e->next= NULL;
	allouer(e,1)
	e^.data←1
	e^.next←NULL
			تم انشاء عنصر جديد e وتهيئة حقوله.

head = e;	head ←e		هنا e وhead يشيران الى نفس العنصر

e = malloc(
sizeof(Cell));
e->data=2;
	allouer(e,1)
	e^.data←2
			تم انشاء عنصر جديد e ويمكن إضافته في اول القائمة
			e->next=head; head=e;
			او في اخرها head->next=e

ملاحظة في C++

e = malloc(sizeof(Cell));⇔ e = new Cell;

6. العمليات على القوائم المترابطة

سنقوم في هذا الجزء بإنشاء مجموعة من البرامج الجزئية للتعامل مع القوائم مثل إضافة او حذف عنصر، اظهار جميع عناصر القائمة، البحث في القائمة، الخ. حيث ينصح بإنشاء مكتبة تحتوي على جميع الدوال الخاصة بالتعامل مع القوائم.

ملاحظة

وهناك عدة طرق لإنشاء الدوال الخاصة بإضافة او حذف عنصر من القائمة

- باستعمال الدوال (fonction) التي تأخذ قائمة كمعامل وترجع قائمة وفي هاته الحالة يمكن تمرير القائمة بالقيمة
- باستعمال الإجراءات (procedure) وعنصر مساعد (sentinel) لتجنب التمرير بالعنوان وفي هاته الحالة يمكن تمرير القائمة بالقيمة.
- باستعمال الإجراءات(procedure) وفي هاته الحالة يجب تمرير القائمة بالعنوان.
- باستعمال الدوال (fonction) التي تأخذ قائمة كمعامل وترجع قيمة منطقية (bool) لتخبرنا هل تمت العملية بنجاح (true) ام لا (false) وفي هاته الحالة يجب تمرير القائمة بالعنوان.

وهاته الأخيرة التي سنقوم باستعمالها.

اظهار القائمة

يتم التنقل عبر جميع عناصر القائمة وإظهار البيانات وننوه هنا انه تم تمرير القائمة بالقيمة وبالتالي لن يتغير راس القائمة الأصلي إذا غيرنا قيمة head لذلك نستعمله للتنقل.	procedure display_list(List head)
	debut
	TQ (head ≠ NULL) faire
	ecrire(head->data,"->")
	head ← head^.next
	FTQ
	printf("fin")
	fin
		void display_list(List head) {
		while (head != NULL) {
		printf("%d->", head->data);
		head = head->next;
		}
		printf("fin\n");
		}

		void display_list(List head) {
		if (head)
		printf("fin\n");
		else{
		printf("%d->", head->data);
		display_list(head->next);
		}
		}

حجم القائمة

يتم التنقل عبر القائمة وإضافة 1 الى ان نصل الى NULL	fonction size_list(List head) : entier
	var n:entier
	debut
	n←0
	TQ (head ≠ NULL) faire
	n←n+1
	head ← head^.next
	FTQ
	size_list←n
	fin
		int size_list(List head) {
		int n=0;
		while (head != NULL) {
		head= head->next;
		n++;
		}
		return n;
		}

		int size_list(List head) {
		if (!head) return 0;
		return 1+ size_list(head->next);
		}

إضافة عنصر الى القائمة

تتمّ عملية إضافة عنصر الى القائمة المترابطة عبر 3 مراحل:

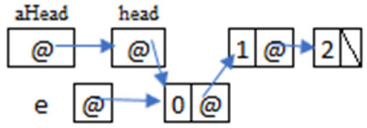
1. انشاء العنصر وتهيئته
2. تحديد مكان العقدة.
3. إضافة العقدة الى القائمة.

إضافة عنصر في البداية (الراس)

bool add_head(List* aHead, int d) {	function add_head(aHead:^List, d:entier):bool var e:List debut	نقوم بإضافة عنصر في الراس لذلك يجب تمرير القائمة بالعنوان

List e = malloc(sizeof(Cell));
if (e == NULL) {
return false;
}
	allouer(e,1)
	si (e = NULL) alors
	return faux
	fsi
		انشاء عنصر جديد وفي حالة الفشل نرجع false الى المنادي

e-> data = d;	e^.data←d	تهيئة العنصر
e-> next = *aHead;	e^.next←aHead^	تغيير next لـe ليشير الى اول عنصر في القائمة

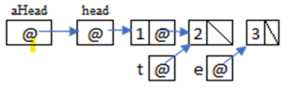
*aHead=e;
return true;
}
	aHead^←e
	return vrai
	fin
		تغيير راس القائمة ليشير الى العنصر الجديد
		 
		aHead وe متغيران محليان يتم حذفهما فور الانتهاء من تنفيذ الاجراء

إضافة عنصر في الأخير

bool append_end(List*aHead, int d) {
List t;
List e = malloc(sizeof(Cell));
if (e == NULL) {
return false;
}
	function append_end(aHead:^List, d:entier):
	bool
	var e,t:List
	debut
	allouer(e,1)
	si (e = NULL) alors
	return faux
	fsi
		هاك احتمال ان نقوم بإضافة عنصر في الراس لذلك يجب تمرير بالعنوان
		انشاء عنصر جديد

e-> data = d; e-> next = NULL;	e^.data←d e^.next← NULL	تهيئة العنصر واسناد NULL لـnext لأنه سيكون العنصر الاخير
if (*aHead == NULL) *aHead = e;	si (aHead^=NULL) alors aHead^←e	في حالة القائمة فارغة تتم الإضافة في الراس

else {
t= *aHead;
while (t-> next != NULL)
t= t-> next;
	sinon
	t←aHead^
	TQ (t^.next≠NULL) faire
	t←t^.next
	FTQ
		في حالة القائمة تحتوي على الأقل على عنصر يتم البحث على العنصر الاخير

t-> next=e;
}
return true;
}
	t^.next←e
	fsi
	return vrai
	fin
		ادراج العنصر في الأخير
		 

حذف عنصر من القائمة

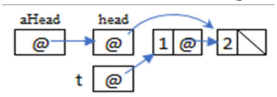
تتمّ عملية حذف عقدة من القائمة المترابطة عبر 4 خطوات:

1. تحديد العقدة السابقة للعقدة المراد حذفها.
2. تخزين عنوان العقدة المراد حذفها
3. ربط العقدة السابقة بالعقدة التالية للعقدة المراد حذفها.
4. إفراغ الذاكرة المحجوزة من قبل العقدة المراد حذفها.

لذلك هناك 3 حالات اما القائمة فارغة او تحتوي على عنصر واحد او تحتوي على أكثر من عنصر.

حذف عنصر من البداية (الراس)

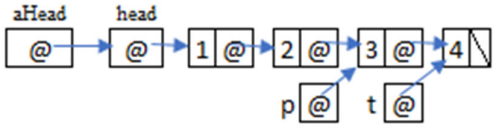
bool delete_head(List*aHead) { List t;	function delete_head(aHead:^List): bool var t:List debut	هاك احتمال ان نقوم بحذف عنصر من الراس لذلك يجب تمرير بالعنوان
if (aHead== NULL) return false;	si (aHead^ ==NULL) alors return faux fsi	في حالة القائمة فارغة لا يوجد أي عنصر للحذف لذلك نرجع false
t = *aHead;	t←aHead^	تخزين عنوان العنصر الاول المراد حذفه
*aHead =t-> next;	aHead← t^.next	الربط مع العنصر الثاني

free(t);
return true;
}
	desallouer(t)
	return vrai
	fin
		إفراغ الذاكرة المحجوزة من قبل العنصر الاول
		 

حذف عنصر من الأخير

bool delete_end (List*aHead) { List t, p;	function delete_end(aHead:^List): bool var t, p:List debut	هاك احتمال ان نقوم بحذف عنصر من الراس لذلك يجب تمرير بالعنوان t يحمل العنصر الأخير وp العنصر ما قبل الأخير
if (aHead== NULL) return false;	si (aHead^ ==NULL) alors return faux fsi	في حالة القائمة فارغة لا يوجد أي عنصر للحذف لذلك نرجع false

if ((*aHead)->next
==NULL) {
free(*aHead);
*aHead = NULL; }
	si (aHead^.next =NULL) alors
	desallouer(*aHead)
	*aHead←NULL
		في حالة القائمة تحتوي على عنصر واحد نحذفه مباشرة من الراس

else {
t = *aHead;
while (t->next != NULL) {
p=t ;
t= t->next;
}
	sinon
	t←aHead^
	TQ (t^.next ≠ NULL) faire
	p←t
	t←t^.next
	FTQ
		في حالة القائمة تحتوي على أكثر من عنصر يتم البحث على العنصر الأخير t وما قبل الأخير p
		 

p-> next=NULL;
free(t);
}
return true;
	p^.next←NULL
	desallouer(t)
	fsi
	return vrai
		نسند NULL لما قبل الأخيرp لأنه أصبح هو الاخير
		نحذف الأخير tمن الذاكرة

}	fin	

حذف القائمة

void delete_list(List*aHead) {
List t;
while(*aHead!= NULL) {
t = *aHead;
*aHead =t-> next;
free(t);
}
}
	procedure delete_ list(aHead:^List)
	var t:List
	debut
	TQ (aHead^ ≠NULL) faire
	t←aHead^
	aHead^←t^.next
	desallouer(t)
	FTQ
	fin
		نقوم بالحذف من الراس الى ان تصبح القائمة فارغة

void delete_list(List*aHead) {
while (delete_head(aHead));
}
	procedure delete_ list(aHead:^List)
	debut
	TQ (aHead^ ≠NULL) faire
	delete_head(aHead)
	FTQ
	fin
		او باستخدام الدالة delete_head الى ان ترجع false

البرنامج الرئيسي (الاستعمال)

int main(int argc, char *argv[]) {
List head = initialization();
add_head(&head, 3);
add_head(&head, 2);
append_end(&head, 4);
add_head(&head, 1);
append_end(&head, 5);
printf("size=%d\n", size_list(head));
display_list(head);
delete_head(&head);
delete_end(&head);
printf("size=%d\n", size_list(head));
display_list(head);
delete_list(&head);
printf("size=%d\n", size_list(head));
display_list(head);
return 0;
}
	debut
	add_head(@head, 3)
	add_head(@head, 2)
	append_end(@head, 4)
	add_head(@head, 1)
	append_end(@head, 5)
	printf("size=", size_list(head))
	display_list(head)
	delete_head(@head)
	delete_end(@head)
	printf("size=", size_list(head))
	display_list(head)
	delete_list(@head)
	printf("size=", size_list(head))
	display_list(head)
	fin
		البرنامج سيظهر
		size=5
		1->2->3->4->5->fin
		ثم يظهر
		size=3
		2->3->4->fin
		وفي الأخير يظهر
		size=0
		fin