Chapitre 1 : Les pointeurs

Hassen NAKBI ::: hassen.nakbi@gmail.com

1^{ere} année Licence

23 janvier 2024

Introduction

Notion de Variable

En programmation, une variable est un objet qui permet de mémoriser une seule information simple. Elle est définie par son nom, son type et sa valeur, occupant ainsi un espace mémoire précis localisé par une adresse mémoire.

Introduction

Problématique

Le problème que nous n'avons pas encore clairement résolu concerne le chapitre de la modularité, en particulier la communication entre le programme principal et le sous-programme pour échanger des informations via l'adresse d'une variable.

Introduction

Exemple d'échange d'informations

```
Algorithme principal échange        DÉBUT
VAR x,y : entier                    x ⟵ 11
Procédure XCHG(a,b :entier)         y ⟵ 7
VAR c : entier                      XCHG(x,y)
Début                               Écrire("x :",x," y :",y)
c ⟵ a                               FIN
a ⟵ b
b ⟵ c
Fin
```

Introduction

Représentation en mémoire

Procédure échange

Variables	a	b	c
État initial	0	0	0
Appel	x	y	0
Exécution	7	11	11

Algorithme principal

Variables	x	y
État initial	0	0
Affectation	11	7
Après appel	11	7

Nous constatons ici qu'il y a un problème d'échange d'informations entre l'algorithme principal et la procédure échange.

2 Notion de pointeur

- 2.1 Définition
- 2.2 Propriétés
- 2.3 Définition algorithmique et programmation
- 2.4 Tableau et pointeur
- 2.5 Passage par adresse

Définition

Un pointeur est une variable particulière qui sert à mémoriser l'adresse d'une autre variable du même type, permettant un accès plus rapide à son contenu.

2 Notion de pointeur

- 2.1 Définition
- 2.2 Propriétés
- 2.3 Définition algorithmique et programmation
- 2.4 Tableau et pointeur
- 2.5 Passage par adresse

Propriétés

- Un pointeur doit être déclaré avec le type de données auquel il se pointe.
- Un pointeur peut être assigné à la valeur spéciale "NIL" pour indiquer qu'il ne pointe sur aucune adresse mémoire.
- Les opérations arithmétiques telles que l'addition et la soustraction peuvent être effectuées sur les pointeurs.
- Un pointeur simple contient l'adresse mémoire d'une autre variable.
- Un pointeur double contient l'adresse mémoire d'un autre pointeur.

2 Notion de pointeur

- 2.1 Définition
- 2.2 Propriétés
- 2.3 Définition algorithmique et programmation
- 2.4 Tableau et pointeur
- 2.5 Passage par adresse

Définition algorithmique et programmation

- Algorithmique
  VAR ps : ^type_variable
  VAR pd : ^^type_pointeur
- Langage C
  type_variable *ps ;
  type_pointeur **pd ;

Opérateurs des pointeurs

Les opérateurs des pointeurs sont des éléments clés pour manipuler et accéder aux données à travers les adresses mémoire.

Opérateur	Algo	C	Rôle
Adresse	@	&	Récupère l'adresse mémoire d'une variable.
Indirection	$\wedge$	*	Accède à la valeur stockée à une adresse mémoire pointée par un pointeur.

Exemple d'un algorithme

Algorithme avec les pointeurs

Algorithme Pointeurs
VAR x : entier, psx :^entier,
pdx :^^entier
y : réel, psy :^réel,
pdy :^^réel
z : caractère, pz :^caractère,
pdz :^^caractère
DÉBUT
x ⟵ 13
y ⟵ 3.25
z ⟵ 'F'
psx ⟵ @x
pdx ⟵ @psx
psy ⟵ @y
pdy ⟵ @psy
psz ⟵ @z
pdz ⟵ @psz
psx^ ⟵ psx^ + 5
pdx^^ ⟵ psx^ - 7
psy^ ⟵ psy^ - 1.2
pdy^^ ⟵ psy^ - 0.75
psz^ ⟵ psz^ - 2
pdz^ ⟵ psz^ + 3
FIN

Exemple d'un programme C

Programme avec les pointeurs

```
#include <stdio.h>
short x = 13, *psx = &x,
**pdx = &psx ;
float y = 3.25, *psy = &y,
**pdy = &psy ;
char z = 'F', *psz = &z,
**pdz = &psz ;
int main( ) {
*psx = *psx + 5 ;
**pdx = *psx - 7 ;
*psy = *psy - 1.2 ;
**pdy = *psy - 0.75 ;
*psz = *psz - 2 ;
**pdz = *psz + 3 ;
return 0 ;}
```

2 Notion de pointeur

- 2.1 Définition
- 2.2 Propriétés
- 2.3 Définition algorithmique et programmation
- 2.4 Tableau et pointeur
- 2.5 Passage par adresse

Tableau et pointeur

Adresse de premier élément

Le nom d'un tableau, lorsqu'il est utilisé dans une expression, est converti implicitement en un pointeur constant pointant vers l'adresse du son premier élément.

Exemple :

```
VAR T :TAB
T ≡ @T[1]
VAR A :MAT
A ≡ @A[1][1]
```

```
short T[N];
T ≡ &T[0];
short A[M][N];
A ≡ &A[0][0];
```

Tableau et pointeur

Accès aux éléments du tableau

Plutôt que d'utiliser l'opérateur crochets avec des indices, l'accès aux éléments d'un tableau à l'aide d'un pointeur se fait en utilisant l'opérateur d'indirection ^ ou bien *

```
Procédure affiche(T :TAB)
VAR PT :^entier
Début
Pour PT de T à T+N-1
pas←—1 faire
Écrire(PT^)
Fin Pour
Fin
```

```
void affiche(short T[ ]) {
short *PT ;
for(PT=T ;PT <
T+N ;PT++)
printf("%hi ",*PT) ;}
```

Tableau et pointeur

Chaîne est un pointeur

Le nom d'une variable chaîne fait référence à un pointeur constant sur l'adresse de premier caractère. Cette adresse nous facilite la manipulation d'une chaîne.

Algorithme chaîne
VAR C1,C2 :^caractère
DÉBUT
C1 ⟵ "chaîne statique"
C2 ⟵ C1
Tant que (C2^ ≠ '\0') faire
Écrire(C2^)
C2 ⟵ C2+1
Fin Tant que
FIN

```
int main( ) {
char *C2 = NULL,
*C1 = "chaîne statique" ;
C2 = C1 ;
while (*C2 != '\0') {
printf("%c ",*C2) ;
C2++ ;}
return 0 ;}
```

2 Notion de pointeur

- 2.1 Définition
- 2.2 Propriétés
- 2.3 Définition algorithmique et programmation
- 2.4 Tableau et pointeur
- 2.5 Passage par adresse

Passage par adresse

Passage par adresse

```
Algorithme principal échange        DÉBUT
VAR x,y : entier                    x ⟵ 11
Procédure                           y ⟵ 7
XCHG(a,b :^entier)                  XCHG(@x,@y)
VAR c : entier                      Écrire("x :",x," y :",y)
Début                               FIN
c ⟵ a^
a^ ⟵ b^
b^ ⟵ c
Fin
```

Passage par adresse

Passage par adresse

```
#include <stdio.h>
short x,y ;
void XCHG(short *a,
short *b){
short c = *a ;
*a = *b ;
*b = c ;}
```

```
int main( ){
x = 11 ;
y = 7 ;
XCHG(&x,&y) ;
printf("x : %hi et y : %hi
",x,y) ;
return 0 ;}
```

3 Allocation dynamique

- 3.1 Définition
- 3.2 Avantages
- 3.3 Fonctions de gestion mémoire
- 3.4 Définition algorithmique et programmation
- 3.5 Tableau dynamique

Définition

L'allocation dynamique permet de réserver et de libérer de l'espace mémoire pendant l'exécution d'un programme.
Plutôt que de définir la taille statique d'une structure de données tel que un tableau, une chaîne et un enregistrement à la compilation, l'allocation dynamique permet d'ajuster dynamiquement la taille des structures de données à l'exécution du programme.

3 Allocation dynamique

- 3.1 Définition
- 3.2 Avantages
- 3.3 Fonctions de gestion mémoire
- 3.4 Définition algorithmique et programmation
- 3.5 Tableau dynamique

Avantages

Les avantages de l'allocation dynamique sont :

- **Gestion de la mémoire** : Offre un contrôle plus précis sur la gestion de la mémoire, permettant une allocation et une libération selon les besoins.
- **Optimisation des ressources** : Permet d'optimiser l'utilisation de la mémoire en ne réservant de l'espace que lorsque cela est nécessaire.
- **Économie d'espace** : Évite la réservation d'une grande quantité fixe de mémoire à la compilation, ce qui peut économiser de l'espace lorsque la taille réelle des données est inconnue à l'avance.

Fonctions de gestion mémoire

Les fonctions de gestion de la mémoire sont essentielles pour l'allocation et la libération dynamiques de mémoire.

Fonction	Rôle
Allouer(taille*type)	Réserver un ensemble des cases mémoire et retourne l'adresse de début de cet espace.
Réallouer(pt,taille*type)	Modifie la taille d'un bloc de mémoire déjà alloué.
Libérer(pt)	Libère un bloc de mémoire préalablement alloué.

Fonctions de gestion mémoire

En langage C, les fonctions de gestion mémoire sont définies dans le fichier bibliothèque <stdlib.h>.

Fonction	Rôle
sizeof(type)	Renvoi la taille mémoire en octets, c'est le nombre des cases mémoires.
malloc(nb*sizeof(type))	Réserver un ensemble des cases mémoire et retourne l'adresse de début de cet espace.
realloc(pt,nb*sizeof(type))	Modifie la taille d'un bloc de mémoire déjà alloué.
free(pt)	Libère un bloc de mémoire préalablement alloué.

Définition algorithmique et programmation

Algorithme allocationdynamique
VAR pte : ^entier, ptr : ^réel, ptc : ^caractère
DÉBUT
pte ⟵ allouer(entier)
ptr ⟵ allouer(2*réel)
ptc ⟵ allouer(caractère)
pte^ ⟵ 13
ptr^ ⟵ 0.75
ptc^ ⟵ 'E'
libérer(pte)
libérer(ptr)
libérer(ptc)
FIN

Définition algorithmique et programmation

```
#include <stdlib.h>
int main( ) {
short *pte = malloc(sizeof(short)) ;
float *ptr = malloc(sizeof(float) ;
char *ptc = malloc(sizeof(char)) ;
*pte = 13 ;
*ptr = 0.75 ;
*ptc = 'E' ;
free(pte) ;
free(ptr) ;
free(ptc) ;
return 0 ; }
```

Tableau dynamique

```
Procédure taille(N :^entier)
VAR adr : ^entier
Début
Répéter
Écrire("Donner une taille du tableau")
Lire(N^)
jusqu'à(N^ > 2)
Fin
Fonction reserver(N :entier) : ^entier
VAR adr : ^entier
Début
adr ⟵ allouer(N*entier)
Renvoyer adr
Fin
```

Tableau dynamique

```
Procédure remplir(tab :^entier,N :entier)
VAR i : entier
Début
Pour i de 0 à tab+N-1 pas←—1 faire
Écrire("Donner une valeur")
Lire((tab+i)^)
FinPour
Fin
Procédure affichage(tab :^entier,N :entier)
VAR i :entier
Début
Pour i de 0 à tab+N-1 pas←—1 faire
Écrire((tab+i)^)
FinPour
Fin
```

Tableau dynamique

```
Algorithme principal tabdynamique
VAR T :^entier
N :entier
DÉBUT
taille(@N)
T⟵reserver(N) ;
remplir(T,N)
affichage(T,N)
Libérer(T)
FIN
```