TP d’algorithmique

n°2

Table des matirères

Exercice 1 2

Rappel de l’énoncé 2

Analyse du travail à faire 2

Schémas 2

Exercice 2 3

Rappel de l’énoncé 3

Programme 3

Erreurs de compilation 4

Erreurs d’exécution 4

Programme corrigé 4

Exercice 3 4

Rappel de l’énoncé 5

Analyse du travail à faire 5

Algorithme 5

Programme 5

Jeu d’essai 6

# Exercice 1

## Rappel de l’énoncé

Tester le programme de l’exercice 2 du TD n°2 afin de vérifier les affichages et expliquer avec des schémas le fonctionnement de ce programme.

## Analyse du travail à faire

Recopier l’algorithme et faire des schémas, suivant chaque instruction pas à pas.

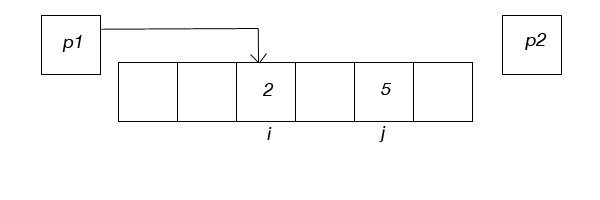
## Schémas

int i, j ;

int \*p1, \*p2 ;

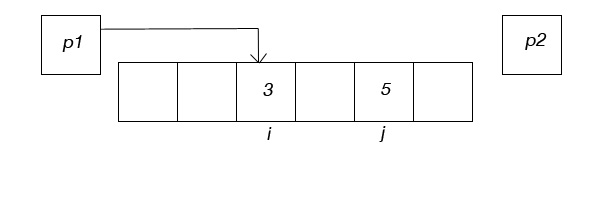
i=2 ; j=5 ;

p1=&i ;



contenu de p1 vaut : 2

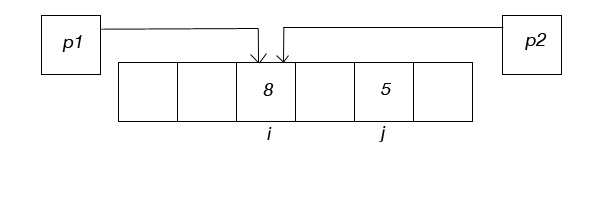
i=i+1 ;



contenu de p1 vaut : 3

p2=p1 ;

\*p2=\*p1+j ;

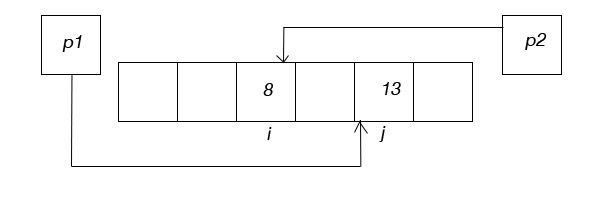


p1 vaut : 8 p2 vaut : 8

i vaut : 8 j vaut : 5

p1=&j ;

\*p1=i+j ;

p1 vaut : 13 p2 vaut : 8

i vaut : 8 j vaut : 13

# Exercice 2

## Rappel de l’énoncé

Recopier le programme de l’exercice 4 du TD. Noter les erreurs de compilation, puis les erreur d’exécution et corriger ses erreurs afin que le programme se compile et se lance correctement, peu importe ce qu’il renvoie.

## Programme

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #include <iostream>  #include <stdio.h>  using namespace std ;  int main()  {  int tab[5 ] ;  int \*p1, \*p2 ;  char str[]20 ;  char \*c ;  tab[0] = 5 ;  cout << « tab[2] vaut :  » << tab[2] << endl ;  p1 = tab+1 ;  \*p1 = 10 ;  \*p2 = 20 ;  p2 p1 ;  cout << « p1 vaut : » << \*p1 << « p2 vaut : » << \*p2 << endl ;  tab = p1+1 ;  c = tab+3 ;  gets(str) ;  c = str+2 ;  c = c+1 ;  cout << « caractère : » << \*(c+1) << endl ;  gets(c) ;  cout << « chaine : » << str ;  return 0 ;  } |

## Erreurs de compilation

Ligne 18 -- On ne peut pas convertir un pointeur en tableau : on ne modifie pas tab, l’adresse du tableau.

Ligne 19 -- Erreur de type, le pointeur tab, pointe sur un entier alors que \*c pointe vers un caractère.

## Erreurs d’exécution

Ligne 13 -- tab[2] n’est pas initialisé, la valeur affichée dedans sera alors fausse.

Ligne 16 -- p2 est utilisé sans être initialisé, on ne sait pas vers quelle adresse il pointe.

## Programme corrigé

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #include <iostream>  #include <stdio.h>  using namespace std ;  int main()  {  int tab[5 ] ;  int \*p1, \*p2 ;  char str[]20 ;  char \*c ;  tab[0] = 5 ;  cout << « tab[2] vaut :  » << tab[2] << endl ;  p1 = tab+1 ;  \*p1 = 10 ;  \*p2 = 20 ;  // p2 p1 ;  cout << « p1 vaut : » << \*p1 << « p2 vaut : » << \*p2 << endl ;  // tab = p1+1 ;  // c = tab+3 ;  gets(str) ;  c = str+2 ;  c = c+1 ;  cout << « caractère : » << \*(c+1) << endl ;  gets(c) ;  cout << « chaine : » << str ;  return 0 ;  } |

# Exercice 3

## Rappel de l’énoncé

Programmer l’algorithme de tri rapide récursif sur un tableau d’entiers.

## Analyse du travail à faire

Nous plaçons des indices aux extrémités du tableau ainsi qu’un pivot au milieu. Dans une boucle, nous testons si la valeur contenue à l’indice gauche est plus petite ou grande que le pivot, si elle l’est, nous nous arrêtons. Nous testons de la même manière l’indice droit. Si les deux sont mal placés alors nous les échangeons.

Une fois que les indices se sont croisés, nous scindons le tableau en deux sous-tableaux auxquels nous appliquons cet algorithme.

## Algorithme

Procédure quicksort(tab : Entier[], debut : Entier, fin : Entier ; tab : Entier[])

Début

Avec temp : Entier

a : Entier

b : Entier

pivot : Entier

a <- deb

b <- fin

Pivot <- tab[(fin+deb)/2]

Si (fin-deb)+1<=1 Alors

Retourne

Fin Si

Tant Que a<=b Faire

Tant que tab[a]<pivot Faire

a <- a+1

Fin Faire

Tant Que tab[b]>pivot Faire

b <- b+1

Fin Faire

Si a<=b Alors

temp <- tab[a]

tab[a] <- tab[b]

tab[b] <- temp

a <- a+1

b <- b-1

Fin Si

Fin Faire

quicksort(tab, deb, b)

quicksort(tab, a, fin)

Fin quicksort

## 

## Programme

#include <iostream>

#include <stdlib.h>

using namespace std;

void quicksort(int tab[],const int debut, const int fin)

{

int a;

int b;

int pivot;

int temp;

if ((fin-debut)+1 <= 1)

return;

a = debut;

b = fin;

pivot = tab[(fin+debut)/2];

while (a<=b)

{

while (tab[a]<pivot)

a++;

while (tab[b]>pivot)

b--;

if (a<=b)

{

temp=tab[a];

tab[a]=tab[b];

tab[b]=temp;

a++;

b--;

}

}

quicksort(tab, debut, b);

quicksort(tab, a, fin);

}

void afficherTab(int tab[], int taille)

{

int compt;

for (compt=0; compt<taille; compt++)

cout << tab[compt] << " ";

}

int main()

{

int tableau[]={9,2,5,6,2,8,4,3,1};

afficherTab(tableau, 9);

quicksort(tableau, 0, 8);

cout << endl;

afficherTab(tableau, 9);

system("pause");

return (0);

}

## Jeux d’essai

|  |  |  |
| --- | --- | --- |
| **Cas testé** | **Entrée** | **Résultat** |
| Cas général | {4,6,2,3,9,8,1} | {1,2,3,4,6,8,9} |
| Nombres égaux | {8,3,4,2,2,9,1,5} | {1,2,2,3,4,5,8,9} |
| Déjà trié | {1,2,3,4,5,6,7} | {1,2,3,4,5,6,7} |
| Tableau vide | {} | {} |

Oncle D:Users:Junior:tp_dutinfo:TP02:Ex3_scr1.png

Oncle D:Users:Junior:tp_dutinfo:TP02:Ex3_scr2.png

Oncle D:Users:Junior:tp_dutinfo:TP02:Ex3_scr3.png

Oncle D:Users:Junior:tp_dutinfo:TP02:Ex3_scr4.png