

# Calculatrice de style HP

## Principe:

Une calculatrice de style HP s'utilise en entrant d'abord l'ensemble, ou une partie, des valeurs avec lesquelles on va travailler, et ensuite les opérateurs.

Exemples:

Pour effectuer  $3 + 5 \cdot 2$

on tape 3 enter 5 enter 2 enter \* +

Le programme va effectuer 5.2, puis ajouter 3.

Pour effectuer  $(3.2) + (5.3)$

on tape 3 enter 2 enter \* 5 enter 3 enter \* +

## De façon général:

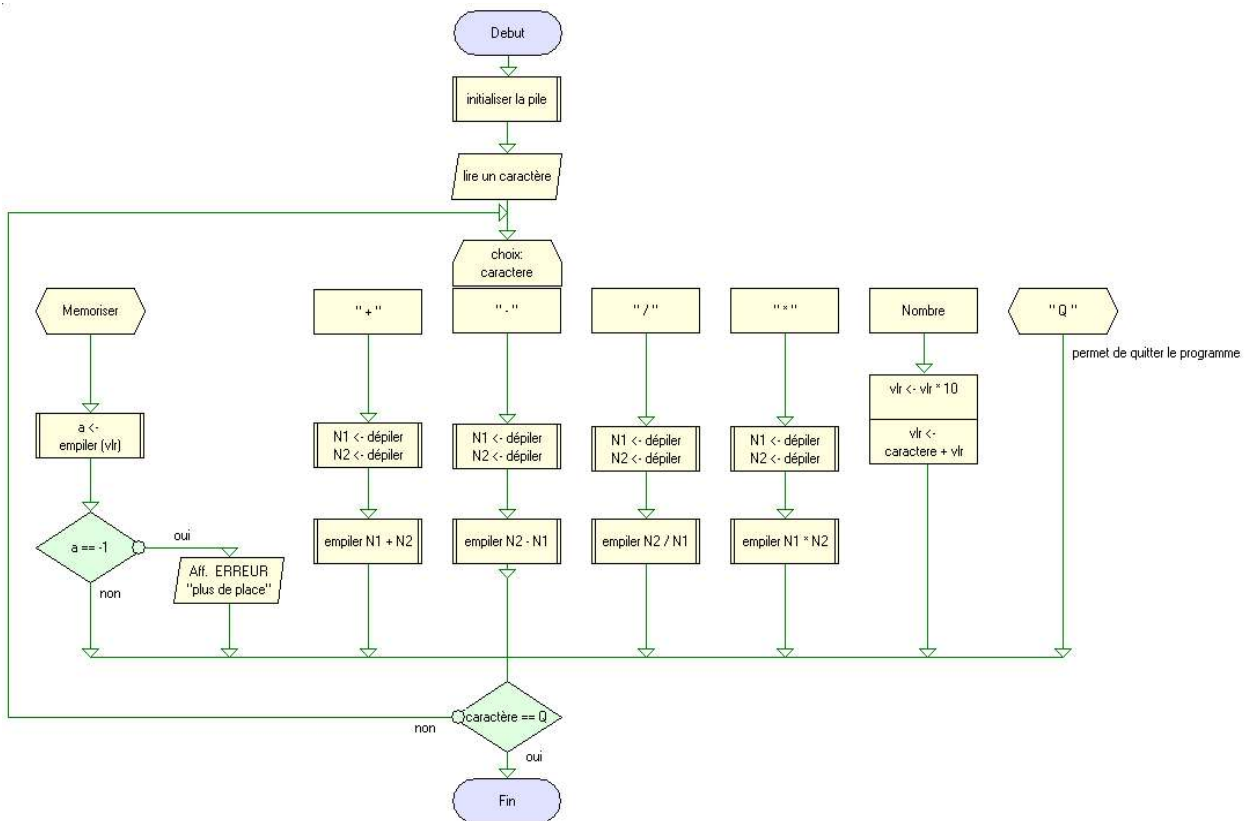
En simplifiant, le programme doit être capable de:

Mémoriser les valeurs qui doivent être traitées.

L'utilisation d'une pile sera ici nécessaire, il faudra donc l'initialiser<sup>1</sup> et empiler.

Récupérer ces valeurs pour les traiter et mémoriser les solutions. (empiler et dépiler<sup>1</sup>)

Effectuer les calculs en fonction des différents opérateurs.



<sup>1</sup> Le Document *Gestion d'une pile* de M. Gosseye traite en détail de ses fonctions

<i>Genre</i>	<i>Nom</i>	<i>Type</i>	<i>Descr.</i>
Define	TLLPILE	Valeur = 30	Taille de la pile
Type structuré	ts_pile v[TLLPILE] indice	Defini la pile int [ ] int	Pile proprement dite indice de la 1 <sup>ere</sup> place vide
Fonction	emp	Int	Mémorise dans la pile
Fonction	dep	Int	Récupère dans la pile
Fonction	inip	Void	Initialise la pile
Variable	pp	ts_pile	Pile réelle
Variable	vlr	Int	Contient le nombre à traiter
Variable	caractere	Char	Caractère tapé au clavier
Variable	i	Int	Sert à être incrémenter
Variable	val1	Int	1 <sup>er</sup> nombre récupéré par <i>dep</i>
Variable	val2	Int	2 <sup>e</sup> nombre récupéré par <i>dep</i>

### [...includes]

```
#define TLLPILE 30 /*taille du vecteur dans ts_pile*/
```

```
typedef struct { int v[TLLPILE]; /*definition du type pile*/
                unsigned int indice;
                }ts_pile;
```

### [...Declaration des fonctions]

```
void main ()
{ ts_pile pp;
  int vlr=0;
  char caractere;
  int i=0;
  int val1,val2;
  /**/inip(&pp); /*initialisation de la pile*/
  /**/clrscr();
  do
  { caractere=getch();

  switch (caractere){
  case '9': i++;
  case '8': i++;
  case '7': i++;
  case '6': i++;
  case '5': i++;
  case '4': i++;
  case '3': i++;
  case '2': i++;
  case '1': i++;
  case '0': printf("%d",i);
              vlr*=10;
              vlr+=i;
              i=0;
              break;
```

Cette structure permet de transformer facilement un caractère numérique en int. Le programme continu de s'exécuter jusqu'au premier *break* rencontré.

ex: si l'utilisateur tape 32  
2  
3 \* 10 = 30  
30 + 2 = 32

on affiche le nombre entré à l'écran  
on décale la valeur à gauche  
on ajoute le nombre entré  
i revient a 0 pour le passage suivant.

```

case 'n' : printf("\t\tValide\n"); /*enter*/
emp (&pp,vlr);
break;
case '/':
val1 = dep(&pp);
val2 = dep(&pp);
val2/=val1;
emp (&pp,val2);
printf ("\n\t%d\n",val2);
break;
[...]
```

```

case 'Q' : clrscr(); break;
default : fprintf (stderr,"tOUPS PAS COMPRIS\n");
vlr=0;

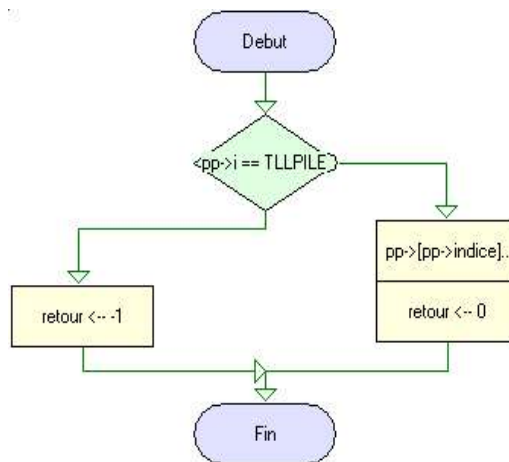
}/*fin du switch*/
}while (caractere!='Q');
}
[...]
```

### fonctions utilisées

#### Empiler (emp) et initialiser (inip) la pile

On utilise les fonctions identiques à celles vues au Labo 11 de M. Gosseye.

Empiler:



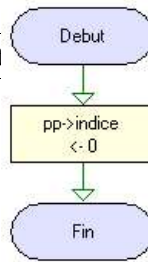
<i>Genre</i>	<i>Nom</i>	<i>Type</i>	<i>Descr.</i>
Paramètre	p	ts_pile * (pointeur)	Adresse de la pile qui reçoit la valeur
Paramètre	vlr	Int	Valeur à mémoriser

```

int emp (ts_pile * p,int vlr)
{ if (p->indice==TLLPILE) return -1;
  else
  { p->v[p->indice]=vlr;
    p->indice++;
    return 0;
  }
}
```

**Initialiser:**

<i>Genre</i>	<i>Nom</i>	<i>Type</i>	<i>Descr.</i>
Paramètre	_pp	ts_pile *	Adresse de la pile à initialiser

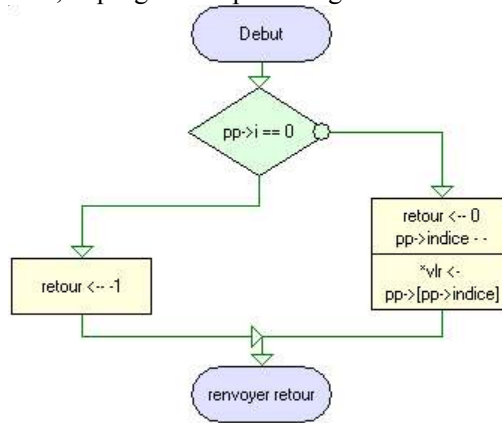


```

void inip (ts_pile * _pp)
{
  _pp->indice=0;
}
  
```

**Dépiler (dep)**

La fonction dépiler reçoit ici l'adresse de la variable qui doit recevoir la valeur dont on a besoin. En effet, si on renvoie cette valeur avec un *return*, le programme pourrait générer un erreur si -1 est contenu dans la pile.



<i>Genre</i>	<i>Nom</i>	<i>Type</i>	<i>Descr.</i>
Paramètre	p	ts_pile * (pointeur)	Adresse de la pile à dépiler
Paramètre	vlr	Int * (pointeur)	Adresse de la variable qui contiendra la valeur dépilée.

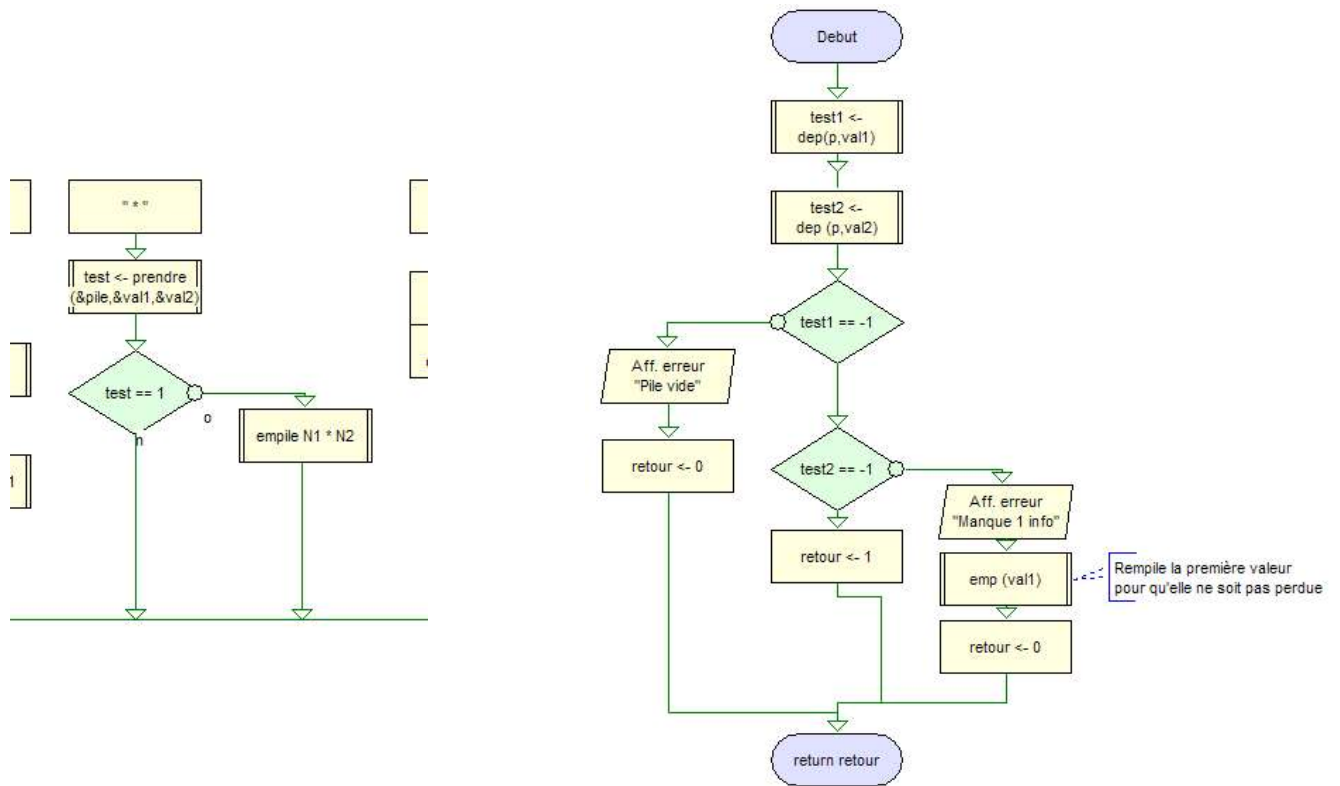
```

int dep (ts_pile * p,int * vlr)
{
  if (p->indice!=0)
  {
    p->indice--;
    *vlr = p->v[p->indice];
    return 0;
  }
  return -1;
}
  
```

## Compléter un peu ...

### Erreur au dépileage

Si la pile est vide, il sera impossible de dépiler une valeur correcte. (logique !)  
Il faut donc créer une nouvelle fonction (p.ex. *prendre*) pour gérer les éventuelles erreurs au dépileage.



Genre	Nom	Type	Descr.
Paramètre	p	ts_pile * (pointeur)	Adresse de la pile dans laquelle chercher les 2 valeurs à calculer.
Paramètre	val1 et val2	int * (pointeur)	Adresses des variables qui contiendront les valeurs dépilées.
Variable	test1 et test2	int	Contiennent les valeurs renvoyées par <i>dep</i>

```
[main...]
case '+': if (prendre(&pp, &val1, &val2) != 1)
    {
        val1 += val2;
        emp (&pp, val1);
        printf ("\n\t%d\n", val1);
    }
    break;

[... ]
int prendre (ts_pile * p, int * val1, int * val2)
{
    int test1, test2;
    test1 = dep(p, val1);
    if (test1 == -1)
    {
        fprintf (stderr, "OUPS C'EST VIDE");
        return 0;
    }
    test2 = dep(p, val2);
```

```

if (test2== -1)
{ fprintf (stderr,"OUPS MANQUE DE DONNEE");
  emp(p,*val1);
  return 0;
}
return 1; }

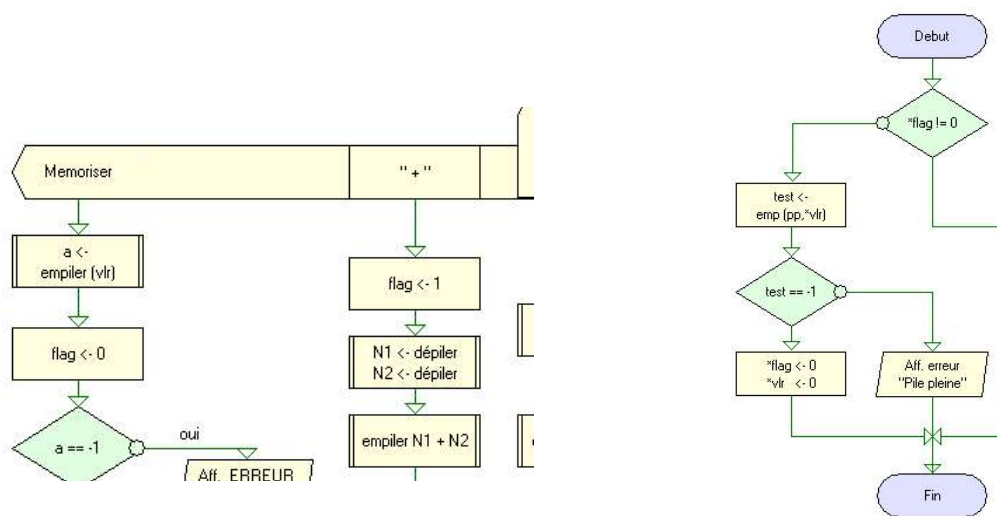
```

### Validation automatique

On peut simplifier l'utilisation du programme en permettant à l'utilisateur d'introduire un opérateur sans avoir à valider le dernier nombre entré.

Il suffirait d'empiler la valeur avant d'effectuer l'opération. Naturellement, si aucun nouveau nombre n'a été tapé au clavier, le programme ne peut pas empiler. Il faudra alors utiliser une variable (ici *flag*) qui sera initialisé à 0 après chaque empilement, et mis à 1 après l'introduction d'un chiffre.

L'idéal étant de confier à une nouvelle fonction, le soin de gérer cette condition.



<i>Ajout dans le main:</i>			
Variable	flag	Int	1 si un chiffre nouveau est entré 0 si le nombre est validé

```

void main ()
{ [...]
  int val1,val2,flag=0;
  /**/inip(&pp);
  /**/clrscr();
  do
  { caractere=getch();
    switch (caractere){
    [...]
    case '0': printf("%d",i);
              vlr*=10;
              vlr+=i;
              i=0;
              flag=1; /*--> on informe qu'une nouvelle valeur est entree*/

```

```

break;
[...]

void intro(ts_pile * pp, int * vlr, int * flag)
{
  int test;
  if (*flag)
  {
    test=emp(pp,*vlr);
    if (test==-1) fprintf(stderr,"OUPS C'EST PLEIN");
    *flag=0; /*on informe que la valeur à été sauvegardée*/
    *vlr=0;
  }
}

```

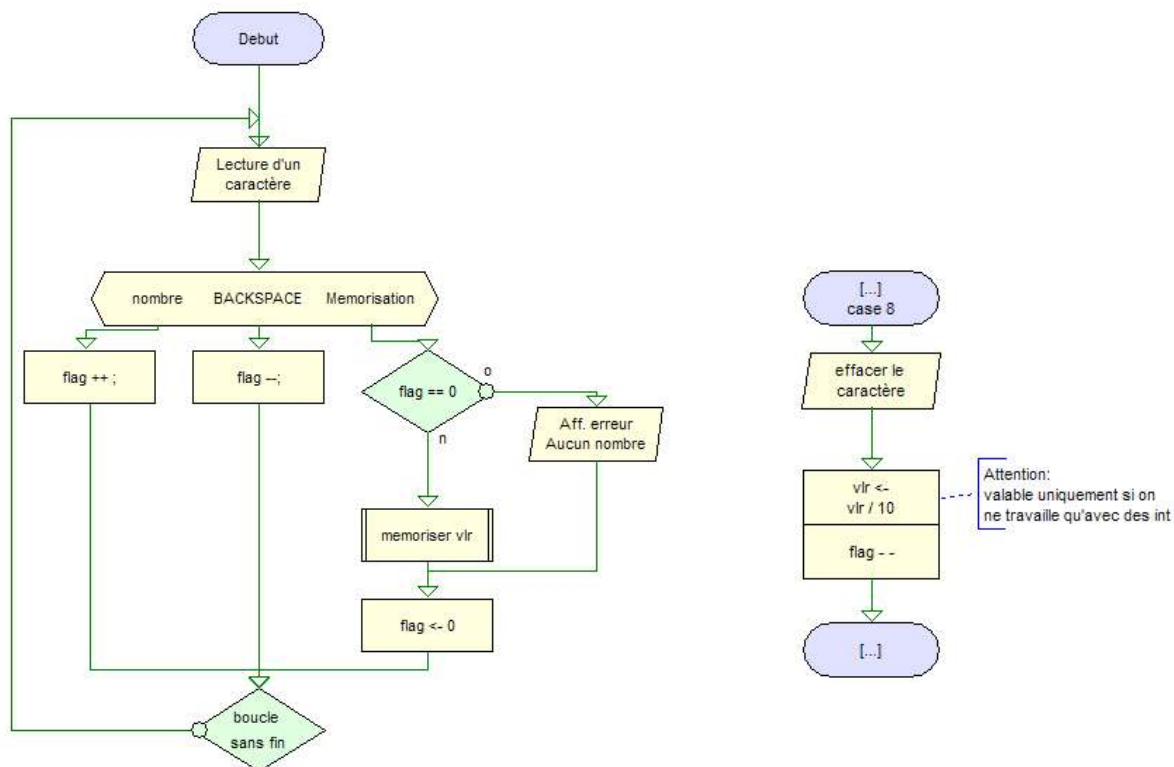
### Utilisation du BACKSPACE

Le *getch* permet une utilisation plus facile du programme, mais ne permet pas à l'utilisateur de corriger une erreur d'introduction. S'il entre 3215 au lieu de 3235, il est obligé de taper n'importe quelle touche, puis de rentrer à nouveau son nombre. On peut facilement lui éviter cette peine en prevoyant l'utilisation de la touche BACKSPACE.

Normalement, un simple *case* supplémentaire permettrait de supprimer la dernière valeur entrée. Dans le cas présent, on risque cependant d'entrer en conflit avec la validation automatique.

Si l'utilisateur entre 3 suivi de BackSpace, *flag* sera de nouveau positif, et s'il entre un opérateur, le programme va empiler le faux contenu de *vlr* (a savoir 0).

Il faudra donc compter le nombre de chiffres entrés en incrémentant *flag*, et décrétement *flag* à chaque BackSpace tapé au clavier.



```

[...] case '2': i++;
      case '1': i++;
      case '0': printf("%d",i);
                vlr*=10;
                vlr+=i;
                i=0;

```

```

        flag++;
        break;
case 8 : if (flag)
        { printf("%c %c", (char)8, (char)8);          /* ou printf("%c", (char)8); */
          vlr/=10;
          flag--;
        }
        break;

```

[...]

## **Le code complet**

```

#include <stdio.h>
#include <conio.h>
#define TLLPILE 30 /*taille du vecteur dans ts_pile*/

typedef struct { int v[TLLPILE];
                unsigned int indice;
                }ts_pile;

int emp (ts_pile * p ,int vlr);
int dep (ts_pile * p );
void inip (ts_pile * _pp );
void intro (ts_pile * pp ,int * vlr , int * flag,int * flagN);
int prendre (ts_pile * p ,int * val1, int * val2);

void main ()
{ ts_pile pp;
  int vlr=0;
  char qqch;
  int i=0;
  int val1, val2, flag=0, flagN=0;

  /**/inip(&pp);
  /**/clrscr();
  printf ("del -> initialise la pile \nesc -> quitte le programme\n\n -> oppose le nombre\n");
  do
  { qqch=getch();

    switch (qqch){
    case '9': i++;
    case '8': i++;
    case '7': i++;
    case '6': i++;
    case '5': i++;
    case '4': i++;
    case '3': i++;
    case '2': i++;
    case '1': i++;
    case '0': printf("%d", i);
              vlr*=10;
              vlr+=i;
              i=0;
              flag++;
              break;
    case 8 : if (flag) /*BackSpace*/
              { printf("%c %c", (char)8, (char)8);
                vlr/=10;

```

```

        flag--;
    }
    break;
case 13 : printf("\t\tValide\n"); /*enter*/
    intro(&pp,&vlr,&flag,&flagN);
    break;
case 'n': if (flagN == 0)
    { flagN=1;
      printf("%c-%d", (char)13, vlr);
    }else
    { flagN=0;
      printf("%c %d", (char)13, vlr);
    }
case '*': intro(&pp,&vlr,&flag,&flagN);
    if (prendre(&pp,&val1,&val2)){
    val1*=val2;
    emp (&pp, val1);
    printf ("\n\t%d\n", val1); }
    break;
case '/': intro(&pp,&vlr,&flag,&flagN);
    if (prendre(&pp,&val1,&val2)){
    val2/=val1;
    emp (&pp, val2);
    printf ("\n\t%d\n", val2); }
    break;
case '-': intro(&pp,&vlr,&flag,&flagN);
    if (prendre(&pp,&val1,&val2)){
    val2-=val1;
    emp (&pp, val2);
    printf ("\n\t%d\n", val2); }
    break;
case '+': intro(&pp,&vlr,&flag,&flagN);
    if (prendre(&pp,&val1,&val2)){
    val1+=val2;
    emp (&pp, val1);
    printf ("\n\t%d\n", val1); }
    break;
case 27 : clrscr(); break; /*27=escape*/
case 127: printf("\nERASE\n\n"); /*127= delete*/
    inip (&pp);
    break;
default : fprintf (stderr, "\tOUPS PAS COMPRIS\n");
    vlr=0; flag=0;

}/*switch*/
}while (qqch!=27);
}

int emp (ts_pile * p,int vlr)
{ if (p->indice==TLLPILE) return -1;
  else
  { p->v[p->indice]=vlr;
    p->indice++;
    return 0;
  }
}

int dep (ts_pile * p,int * vlr)
{ if (p->indice!=0)
  { p->indice--;

```

```

    *vlr = p->v[p->indice];
    return 0;
}
return -1;

void inip (ts_pile * _pp)
{
    _pp->indice=0;
}

void intro(ts_pile * pp, int * vlr,int * flag,int * flagN)
{
    int test;
    if (*flag)
    {
        if (*flagN == 1)
        {
            *vlr *=-1;
            *flagN = 0;
        }
        test=emp (pp,*vlr);
        if (test==-1) fprintf (stderr,"OUPS C'EST PLEIN");
        *flag=0;
        *vlr=0;
    }
}

int prendre (ts_pile * p, int * val1, int * val2)
{
    test1=dep(p,val1);
    if (*val1==-1)
    {
        fprintf (stderr,"OUPS C'EST VIDE");
        return 0;
    }
    test2=dep(p,val2);
    if (*val2==-1)
    {
        fprintf (stderr,"OUPS MANQUE DE DONNEE");
        emp(p,*val1);
        return 0;
    }
    return 1;
}
}

```

*N'hésitez pas à me faire part de vos observations..... [nimalemail-helho@yahoo.fr](mailto:nimalemail-helho@yahoo.fr)*