Algorithmique et programmation

Chapitre 7:Pointeurs

1. Pointeur
2. Allocation de la mémoire
3. Pointeurs et fonctions

Pointeur

- **Définition**: un pointeur est une variable destinée à contenir une adresse (adresse d'une autre variable)

- **Syntaxe**:

```
type  *nom_pointeur;
```

  - Type sur lequel il pointera
  - Caractère * pour préciser que c'est un pointeur
  - Nom du pointeur

- **Exemple**:

```
int  *p; // p est un pointeur sur un int
char  *c; // c est un pointeur sur un char
```

Pointeur (Suite)

- **Initialisation**:

Nom_pointeur = & variable_pointée;

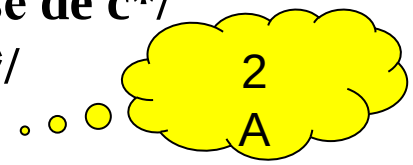
- **Utilisation**:

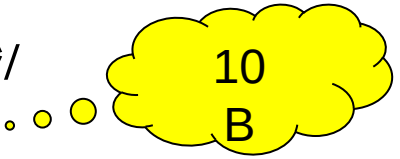
```
# include<stdio.h>
main()
{
 int  *R,S=3; /*R est un pointeur sur un int et S est une variable*/
R = &S;       /* initialiser le pointeur R par l'adresse de S*/
printf ("%d",R);     /*Afficher l'adresse de S*/
printf ("%d",*R);   /*Afficher le contenu de S*/
}
```

Pointeur (Suite)

- **Exemple**:

```
# include<stdio.h>
main()
{
 int  *p1,a=2; /*p1 est un pointeur sur un int et a est une variable*/
 char  *p2,c='A'; /*p2 est un pointeur sur un char et c est une variable*/
p1 = &a;       /* initialiser le pointeur p1 par l'adresse de a*/
p2 = &c;       /* initialiser le pointeur p2 par l'adresse de c*/
printf ("%d \n",*p1);     /*Afficher le contenu de a*/
printf ("%c \n",*p2);   /*Afficher le contenu de c*/
*p1=10; /* affecter 10 à la variable a */
*p2='B'; /* affecter B à la variable c */
printf ("%d \n",*p1);     /*Afficher le contenu de a*/
printf ("%c \n",*p2);   /*Afficher le contenu de c*/
```





Pointeur (Suite)

- **Opérations sur les pointeurs**:

  (*p)++ : incrémentation du contenu

  *(p++) : incrémentation de l'adresse

Allocation de mémoire

- **Principe de l'allocation dynamique**: demander la réservation d'une zone mémoire au système. Cette demande est effectuée à l'aide des fonctions spécialisées dont les arguments permettent de déterminer la taille souhaitée en octets. Ces fonctions retournent un pointeur sur la zone allouée ou un pointeur NULL en cas d'erreur.
- **Règles**:
  - Tester la **valeur de retour** de la fonction d'allocation
  - Mettre un **cast** pour indiquer le type (facultatif)
  - **Libérer** la mémoire allouée à la fin du traitement

Allocation de mémoire (suite)

- **Bibliothèques**: « stdlib.h » et « alloc.h »

- **Taille en octets**: la fonction **sizeof** permet de calculer et retourner le nombre d'octets réservés pour un type.

```
sizeof(type)
```

(Exemple: sizeof(int))

- **Fonctions**:
  - malloc()
  - calloc()
  - realloc()

Allocation de mémoire (suite)

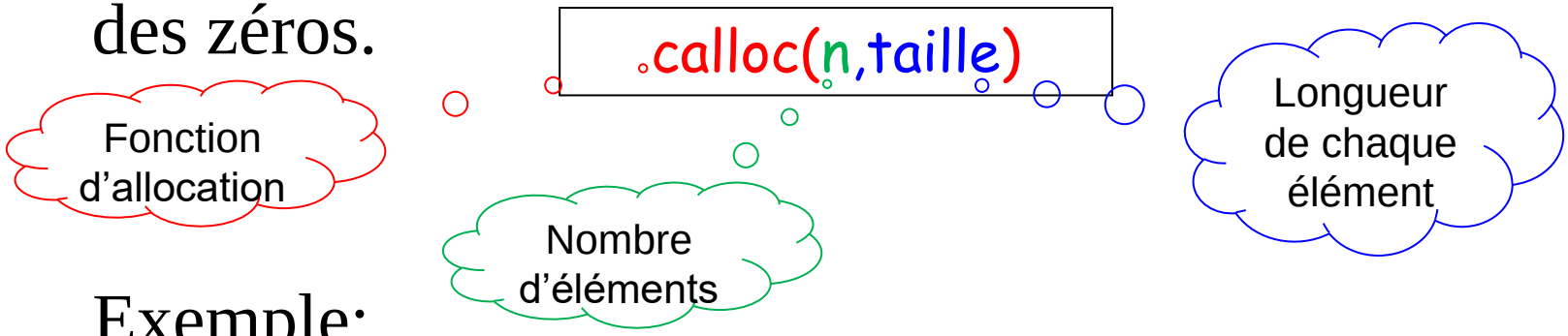
- **malloc**: allocation d'un nombre d'octets (taille: argument de malloc) de la mémoire.

Exemple:

```
int  *p,n=10;
p =(int*) malloc(n*sizeof(int));
if(p!=NULL)
          {
             ……. /* traitement à faire*/
          }
else
          {  printf("Echec de réservation \n");   }
```

Allocation de mémoire (suite)

- **calloc**: allocation de mémoire pour *n* éléments de longueur *taille* et remplissage de la zone réservée par des zéros.

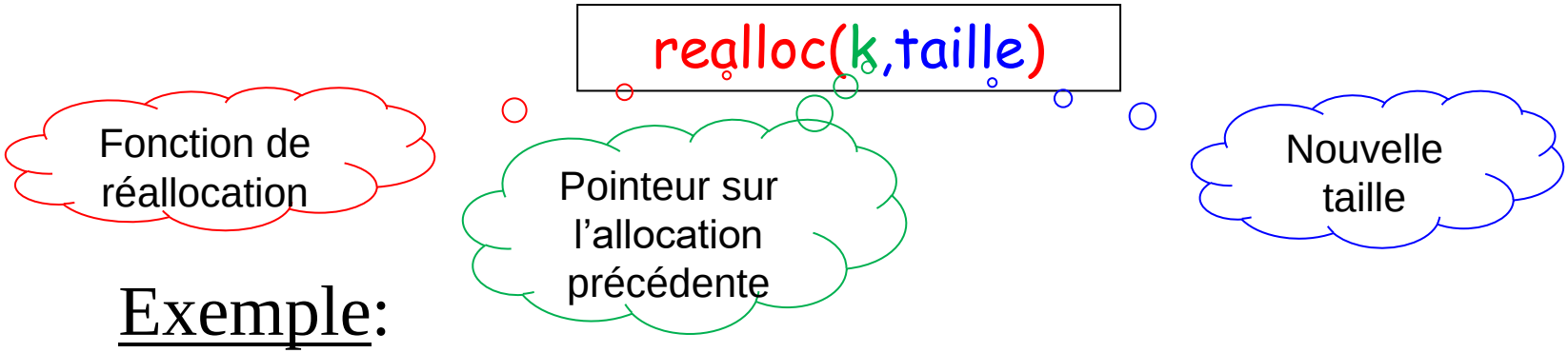


Exemple:

```
char  *c; int n=5;
c =(char*) calloc(n,sizeof(char));
if(c!=NULL)
          {
             ……. /* traitement à faire*/
          }
else
          {   printf("Echec de réservation \n"); }
```

Allocation de mémoire (suite)

- **realloc**: modification de la taille d'une zone préalablement allouée.

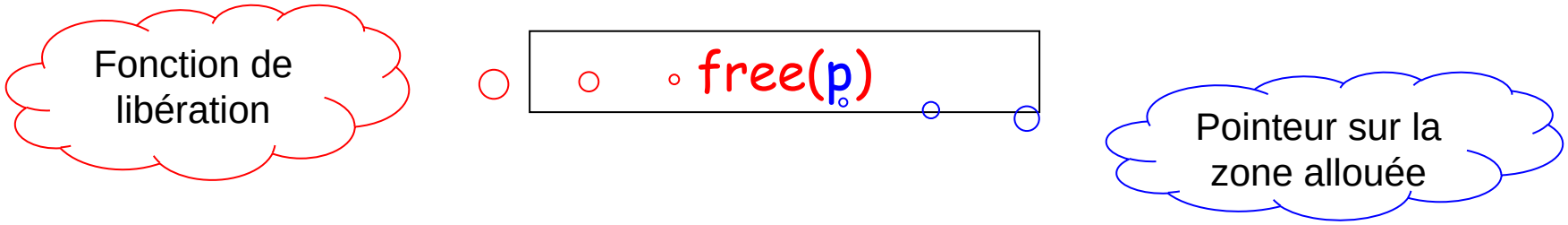


Exemple:

```
int  *p,n=10;
p =(int*) malloc(n*sizeof(int));
…
n=20;
p=(int*)realloc(p, n*sizeof(int));
…
```

Allocation de mémoire (suite)

- **free**: libération de la mémoire réservée



Exemple:

```
int  *h,n=10;
h =(int*) malloc(n*sizeof(int));
...
free(h);
```

Pointeurs et fonctions

- **Passage de paramètres**: il y a deux méthodes pour passer les paramètres à une fonction (passage par valeur et passage par adresse).

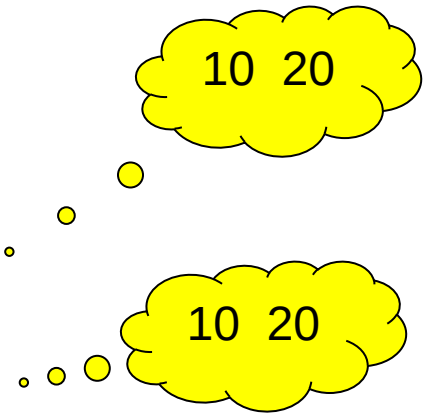
- **Passage par valeur**: la valeur de la variable passée en paramètre est copiée dans une variable locale. La modification de la variable locale dans la fonction appelée ne modifie pas la variable passée en paramètre.

- **Passage par adresse**: l'adresse de la variable passée en paramètre en utilisant un pointeur. Toute modification dans la fonction appelée entraine la modification du contenu de la variable passée en paramètre.

Pointeurs et fonctions (suite)

- **Exemple**: passage par valeur

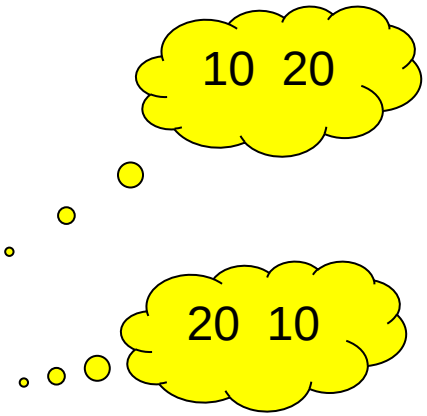
```
#include <stdio.h>
void echange (int  ad1, int  ad2)
{ int x ;
x =  ad1 ;
 ad1 =  ad2 ;
ad2 = x ;
}
main()
{
int a=10, b=20 ;
printf ("avant appel %d %d\n", a, b) ;
echange (a, b) ;
printf ("après appel %d %d", a, b) ;
}
```



Pointeurs et fonctions (suite)

- **Exemple**: passage par adresse

```
#include <stdio.h>
void echange (int  *ad1, int  *ad2)
{ int x ;
x = *ad1 ;
*ad1 = *ad2 ;
*ad2 = x ;
}
main()
{
int a=10, b=20 ;
printf ("avant appel %d %d\n", a, b) ;
echange (&a, &b) ;
printf ("après appel %d %d", a, b) ;
}
```



Exercice 1

Équation de seconde degré

$$\left(a.x^2 + b.x + c = 0\right)$$

On considère ***a***, ***b***, ***c*** et ***det*** quatre variables locales de la fonction ***main()***.

- Saisi de trois réels pour ***a***, ***b*** et ***c***:

void lecture(float*,float*,float*)

- Calcul du discriminant ***det***:

float delta(float*,float*,float*)

- Calcul et affichage de la solution :

void Solution(float*,float*,float*)