

# Programmation avancée

# Allocation Dynamique

Walter Rudametkin

Walter.Rudametkin@polytech-lille.fr  
<https://rudametw.github.io/teaching/>

Bureau F011  
Polytech Lille

CM3

# Allocation de mémoire

## Variables automatiques

- ▶ Variables de bloc, paramètres de fonctions
- ▶ Créées automatiquement à l'exécution
- ▶ Allocation dynamique sur la pile (stack)

## Variables dynamiques

- ▶ Créées et détruites dynamiquement et explicitement
- ▶ Fonctions `malloc` et `free`
- ▶ Allocation sur le tas (heap)

# Erreur d'allocation

*/\* À ne pas faire \*/*

```
int * allouer_entier() {  
    int var_static ; // alloué sur la pile  
    printf("var_static address is : %p\n",  
           &var_static);  
    return &var_static ;  
    /* var_static est libéré lors  
    de la fin de la fonction */  
}
```

# Allocation dynamique — malloc

## Fonction malloc

- ▶ `void * malloc (size_t taille);`
  - ▶ Alloue dynamiquement dans le tas un espace de `taille` octets
  - ▶ Résultat : *pointeur non typé* vers la zone allouée
  - ▶ Pointeur peut être converti automatiquement vers le type désiré (conversion implicite)
  - ▶ Besoin de `#include<stdlib.h>`

# Allocation dynamique — Exemples

## Allocation dynamique d'un entier

```
int *pt;  
//pt = (int *) malloc(sizeof(int));  
pt = malloc(sizeof(int));  
*pt = 42; //utilisation
```

## Allocation dynamique d'un tableau d'entiers

```
int n; int *pt;  
scanf("%d", &n);  
pt = (int *) malloc(n*sizeof(int));  
//Accès  
*pt = 11; //premier élément  
*(pt+1) = 22 ; //deuxième  
pt[2] = 33 ; //troisième  
*(pt+n-1) = 9876 ; //dernier
```

# Allocation dynamique — Structures

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct {
5      int j,m,a;
6  } Date;
7
8  int main(){
9      /*Date *pDate = (Date *) malloc(sizeof(Date));*/
10     Date *pDate = malloc(sizeof *pDate);
11
12     printf("sizeof Date:%lu | sizeof *pDate:%lu\n",
13           sizeof(Date), sizeof *pDate);
14
15     /*ex. utilisation :*/
16     scanf("%d%d%d",&(pDate->j),
17           &(pDate->m),
18           &(pDate->a));
19
20     printf("Date %d/%d/%d\n", pDate->j,
21           pDate->m,
22           pDate->a);
23     free(pDate); pDate = NULL;
24 }
```

# Allocation dynamique — Structures

## Tableau de structures

```
int n;  
Date *pt;  
scanf("%d", &n);  
/*pt = (Date *) malloc( n * sizeof(Date));*/  
pt = malloc(n * sizeof *pt);  
  
/*utilisation: notation equivalent*/  
scanf("%d%d%d", &(pt[0].j),  
        &((*pt+0).m),  
        &pt[0].a);  
printf("Date %d/%d/%d\n", pt[0].j, pt[0].m, pt[0].a);  
free(pt); pt = NULL;
```

# Allocation dynamique — Liste contiguë

```
typedef Date * PtDate;
```

```
typedef struct {  
    PtDate * espace; //vecteur de PtDate alloué dynamiquement  
    int dernier;  
} Liste;
```

```
int n; Liste l;  
l.dernier = -1;
```

```
scanf("%d", &n); //nb de pointeurs à Date
```

```
l.espace = malloc (n * sizeof *l.espace);  
/* Alternative  
l.espace = malloc (n * sizeof (PtDate)); */
```



# Allocation dynamique — Liste contiguë

```
printf("\nAllocate:\n");
for(int i=0 ; i<n ; i++){
    l.dernier+=1;
    l.espace[l.dernier] = malloc(sizeof **l.espace);
    /*l.espace[l.dernier] = malloc(sizeof(Date));*/
    l.espace[l.dernier]->j=i;
    l.espace[l.dernier]->m=i;
    l.espace[l.dernier]->a=i;
}
printf("\nIndice du dernier : %d\n", l.dernier);
for(int i=0 ; i<=l.dernier ; i++){
    printf("Date[%d] %d/%d/%d\n", i, l.espace[i]->j,
                                                l.espace[i]->m,
                                                l.espace[i]->a);
}
```

# Fonction free

- ▶ `void free(void *ptr);`
  - ▶ libère l'espace mémoire pointé par `ptr` (précédemment alloué)
- ▶ Exemple d'utilisation:  
*Suppression du dernier élément de la liste*

```
free(l.espace[l.dernier]);  
l.dernier -= 1;
```

# Listes chaînées — Implantation en C

*//Définition*

```
typedef struct cellule {  
    int valeur;  
    struct cellule *suivant;  
} Cellule;
```

```
typedef Cellule *Liste, *Ptcellule; //optionnel
```

---

*/\* liste vide \*/*

```
Liste l; l = NULL;
```

---

*/\* accès aux champs \*/*

```
Ptcellule p ; //N'oubliez pas de l'initialiser
```

*//...*

```
(*p).valeur; /* ou */ p->valeur ;
```

```
(*p).suivant; /* ou */ p->suivant ;
```

# Listes chaînées — Implantation en C

*//Fonction qui alloue une cellule en mémoire*

```
Ptcellule allouer(){  
    return( (Ptcellule) malloc(sizeof(Cellule)) ) ;  
}
```

```
struct cellule * allouer_sans_typedef(){  
    return( malloc(sizeof(struct cellule)) ) ;  
}
```

---

*//Allocation d'une cellule*

*/\*avec typedef\*/* Ptcellule p2 = allouer();

*/\*sans\*/* **struct** cellule \* p1 = allouer\_sans\_typedef();

*//Libération d'une cellule*

free(p1);

free(p2);

# Listes chaînées — Recherche d'un élément

*//Rappel: Liste ==> struct cellule \**

```
int recherche(int x, Liste l) {  
    int existe ; Ptcellule p;  
    p = l;  
    while ( (p != NULL) && (p->valeur != x) ) {  
        p = p->suivant;  
    }  
    existe = (p!=NULL);  
    return (existe);  
}
```

## Listes chaînées — Exemple: ajout en tête

*//Rappel: Liste \* ==> struct cellule \*\**

```
void ajout_tete (int x, Liste *pL){    // pL en D/R
    Ptcellule p;
    p = allouer();
    p->valeur = x;
    p->suivant = *pL;
    *pL = p;
}

int main(){
    int x ; Liste l=NULL;
    scanf("%d", &x);
    while (x > 0) {
        ajout_tete(x, &l);
        scanf("%d", &x);
    }
    imprimer_liste(l);
}
```