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
- Crées 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 = malloc(n*sizeof(int)); //pas besoin de cast

//Différents façon d'y accèder
*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  struct date {
5      int j,m,a;
6  };
7
8  int main(){
9      struct date * pDate = malloc(sizeof *pDate);
10
11     printf("sizeof Date:%lu | sizeof *pDate:%lu\n",
12                         sizeof(struct date), sizeof *pDate);
13
14     /*example d'utilisation :*/
15     scanf("%d%d%d",&(pDate->j),
16                    &(pDate->m),
17                    &(pDate->a));
18
19     printf("Date %d/%d/%d\n", pDate->j,
20                               pDate->m,
21                               pDate->a);
22     free(pDate);
23     pDate = NULL;
24 }
```

Allocation dynamique — Structures

Tableau de structures

```
17   int n;
18   struct date *pt; // tableau !
19   printf("Taille : ");
20   scanf("%d", &n);
21   pt = malloc(n * sizeof(struct date));
22   //pt = malloc(n * sizeof *pt); // Alternative!
23
24   for (int i = 0; i < n; i++) {
25     scanf("%d%d%d", &pt[i].j,
26                     &pt[i].m,
27                     &pt[i].a // ou &((*(pt+0)).a)
28     );
29   }
30   //printf("Date %d/%d/%d\n", pt[0].j, pt[0].m, pt[0].a),
31   free(pt);
32   pt = NULL;
```


Allocation dynamique — Liste contiguë

```
struct lcontigue {
  struct date ** espace; //vecteur de struct date *
                         //alloué dynamiquement
  int dernier;
};

int n;
struct lcontigue l;
l.dernier = -1;

printf("Saisi le nb de dates : "); scanf("%d", &n);

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


Allocation dynamique — Liste contiguë

```
printf("\nAllocate:\n");
for(int i=0 ; i<n ; i++){
  l.dernier+=1;
  l.espace[l.dernier] = malloc(sizeof(struct date));
                        //ou sizeof **l.espace
  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);
}
for(int i=0 ; i<=l.dernier ; i++){
  free(l.espace[i]); //libère date[i]
}
free(l.espace); //libère tableau de struct date *
```


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

```
struct cellule * l;
l = NULL;
```

```
/* accès aux champs */
struct cellule * p  = NULL; //N'oubliez pas de l'initial
p = malloc(sizeof(struct cellule));
(*p).valeur  = 17;   /* ou */ p->valeur  = 17 ;
(*p).suivant = NULL; /* ou */ p->suivant = NULL;
```


Listes chaînées — Impression d'une liste

```
//Rappel:  une liste ==> struct cellule *
void imprimer_liste(struct cellule * l) {
    struct cellule * p;
    p = l;
    while (p != NULL) {
        printf ("%d -> ", p->valeur);
        p = p->suivant;
    }
    printf("\n");
    return;
}
```

Listes chaînées — Recherche d'un élément v1

```c
//Rappel:  une liste ==> struct cellule *

//Recherche avec un drapeau booléen
int recherche_bool(struct cellule * l, int x) {
    int existe ;
    struct cellule * p;
    p = l;

    while ( (p != NULL) && (p->valeur != x) ) {
        p = p->suivant;
    }

    existe = (p!=NULL);
    return (existe);
}
```


Listes chaînées — Recherche d'un élément v2

```c
//Recherche avec un return (meilleur IMHO)
int recherche_return(struct cellule * l, int x) {

    struct cellule * p;
    p = l;

    while (p != NULL){
        if (p->valeur != x)
            return 1; //trouvé
        p = p->suivant;
    }

    return 0; //fini le parcours, pas trouvé
}
```


Listes chaînées — Exemple: ajout en tête

```c
//Rappel:  pointeur vers une liste ==> struct cellu
void ajout_tete (struct cellule **pL, int x){    //
    struct cellule * tmp;
    tmp = malloc(sizeof(struct cellule));
    tmp->valeur = x;
    tmp->suivant = *pL;
    *pL = tmp;
}
```


Listes chaînées — Exemple: main

```c
int main(){
    struct cellule * l=NULL;

    int x=1 ;
    while (x > 0) { //lisez un 0 pour sortir
        printf("Insert : ");
        scanf("%d", &x);
        ajout_tete(&l, x);
    }

    imprimer_liste(l);
    printf("Rech b : %d", recherche_bool(l,6));
    printf("Rech r : %d", recherche_return(l,6));
    free_liste(&l); //VOUS SAURIEZ FAIRE ?
```


Algorithmes à implémenter

Fonction `sup_tête`

- L'inverse d'`ajout_tête`

Fonction `liberer_liste`

- Astuce : Peut se servir de `sup_tete`

Fonction `insertion_ordonnée`

- Doit chercher le bon emplacement
- Astuce : Peut se servir d'`ajout_tête`

Fonctions d'allocation dynamique

```c
#include <stdlib.h>

void *malloc(size_t size);
void free(void *ptr);

void *calloc(size_t nmemb, size_t size);

void *realloc(void *ptr, size_t size);
void *reallocarray(void *ptr,
                   size_t nmemb,
                   size_t size);
```