

# Algorithmique et programmation

## Structures

R.Gossweiler

29 janvier 2021

- 1 Allocation dynamique
- 2 Libération de mémoire
- 3 Applications de l'allocation dynamique
- 4 Structures et unions
- 5 Structures et pointeurs

# Allocation dynamique

# Problématique

## Question

Comment adapter en mémoire la taille d'un pointeur ?

Comment modifier la taille d'un tableau ?

## Réponse

En agissant sur la mémoire qui leur est liée.

# La fonction malloc

## Rôle

Allouer l'espace mémoire nécessaire

## Prototype

```
1 void *malloc(size_t size);
```

## Valeur de retour

Renvoie NULL si l'allocation a échoué

# La fonction malloc

## Exemple

```
1  int * value = (int*)malloc(sizeof(int));  
2  
3  int * tab = (int*)malloc(15*sizeof(int));  
4  
5  char * s = (char*)malloc(sizeof(char)*100);
```

# Le casting

## Principe

Forcer la reconnaissance d'un élément comme étant d'un autre type

## Syntaxe

```
1      int a;  
2      float b = (float)a/6;  
3  
4      int * tab = (int*)malloc(sizeof(int));
```

# Le casting

## Principe

Le cast ne change pas le type d'un élément !

## Syntaxe

```
1      int a = 5;  
2      float b = (float)a/6;  
3  
4      int r= a/6; // Renvoie 0
```



# La fonction calloc

## Rôle

Allouer l'espace mémoire nécessaire et initialiser la zone mémoire à 0.

## Prototype

```
1 void *calloc(size_t nmemb, size_t size);
```

## Exemple

```
1 int * i = (int*)calloc(15, sizeof(int));
```

# La fonction realloc

## Rôle

Modifier l'espace mémoire associé à un pointeur

## Prototype

```
1 void *realloc(void * ptr, size_t size);
```

## Exemple

```
1 int * i = (int*)malloc(sizeof(int));  
2  
3 i = (int*)realloc(i, 2*sizeof(int));
```

# Libération de mémoire

# Les fuites mémoires

## Définition

Une fuite de mémoire est une perte de mémoire causée par un programme qui consomme plus de mémoire qu'il n'en libère.

## Principe

Tout programme consomme de la mémoire

Toute la mémoire consommée doit être rendue

# La fonction free

## Rôle

Libérer l'espace mémoire d'un pointeur

## Prototype

```
1 void free(void * ptr);
```

## Exemple

```
1 int * i = (int*)calloc(sizeof(int));  
2 free(i);
```

# Applications de l'allocation dynamique

# Fonctions de construction

## Principe

Dédier la création de pointeurs à une fonction spécifique

Approche objet

## Exemple

```
1  int * createInt() {  
2      int * new = malloc(sizeof(int));  
3      return new;  
4  }
```

# Tableaux et allocation dynamique

## Tableaux dynamiques

Un tableau dynamique est un tableau dont la taille est calculée à l'exécution

### Exemple

```
1  int tab[15];  
2  
3  realloc(tab, 20*sizeof(int));
```



# Structures et unions

# Les structures

## Définition

Une structure en C est un type complexe.

## Principe

- Définir une variable comme un ensemble de types
- Alternance entre variables et pointeurs
- Mot-clé struct

# Les structures

```
1 struct cat{  
2     int age;  
3     float poids;  
4 };  
5  
6 struct cat myCat;  
7 myCat.age = 15;
```

```
1 void printAge(struct cat chat) {  
2     printf("\%d",chat.age);  
3 }
```

# Les structures

## Attention

Une structure n'est pas un tableau !

```
1 struct cat{  
2     int age;  
3     float poids;  
4 };  
5  
6 struct cat myCat;  
7 myCat.age = 15;  
8 &(myCat.age+1) // Ne renvoie pas sur poids
```

# Types

## Définition

Les types (ou alias) sont des formes simplifiées de structures.

## Principe

Manipuler le nom d'un type sans le mot-clé **struct**, on utilise la définition d'un type

Augmenter la lisibilité d'un code

Mot-clé **typedef**

## Détails

Séparer la structure du type

Fichiers .h et .c

# Types

```
1 typedef struct cat_{...} cat;  
2  
3 //Ou  
4  
5 struct cat_{};  
6 typedef struct cat_ cat;
```

# Unions

## Définition

Les unions sont des types spéciaux permettant de stocker plusieurs types de données dans la même zone mémoire.

## Principe

Les unions ne sont pas des structures !

Elles permettent d'associer plusieurs types à une même donnée

# Unions

```
1  union character {  
2      int  asciiCode;  
3      char charValue;  
4  };  
5  
6  union character u;  
7  u.asciiCode = 65;  
8  u.charValue = "A";  
9  
10 printf("%d",u.asciiCode); // Erreur
```



# Structures et pointeurs

# Structures et pointeurs

## Principe

Il est possible de définir des pointeurs dans des structures

```
1 struct cat {  
2     char * name;  
3     int * age;  
4 };
```

# Pointeurs sur structures

## Principe

Tout type peut être défini comme un pointeur

Tout élément de structure peut être défini comme un pointeur

```
1 struct cat_ {...};  
2 typedef struct cat_ * cat;  
3  
4 struct cat_ * catA;  
5 cat catA;
```

# Pointeurs sur structures

Création

Allocation dédiée

Fonctions de création

```
1  cat * createCat(int ageP) {  
2      cat * c = (cat*)malloc(sizeof(struct cat_));  
3      c->age = ageP;  
4  
5      return c;  
6  }  
7  
8  cat * newCat = createCat(10);
```

# Opérateur ->

## Problème

Comment accéder à la valeur d'un pointeur dans une structure ?

```
1 struct cat {  
2     char * name;  
3     int * age;  
4 };  
5  
6 struct cat * myCat;  
7 (*myCat).age = 15;  
8 myCat->age = 10;
```

# Opérateur ->

```
1 struct cat_ {  
2     char * name;  
3     int * age;  
4 };  
5 typedef struct cat_ * cat;
```

```
1 cat myCat = (cat)malloc(sizeof(cat));  
2 myCat->name = "Alvina"; // pointeur vers la structure  
3  
4 struct cat_ myCatB;  
5 myCatB.name = "Palico"; // instance de la structure  
6  
7 printf("\%s\n",myCat->name); // Affiche "Alvina"  
8 printf("\%s",myCatB.name); // Affiche "Palico"
```

# Conclusion

- Allocation dynamique
- Libération de mémoire
- Structures
- Pointeurs et structures