Mémoire dynamique

Allocation

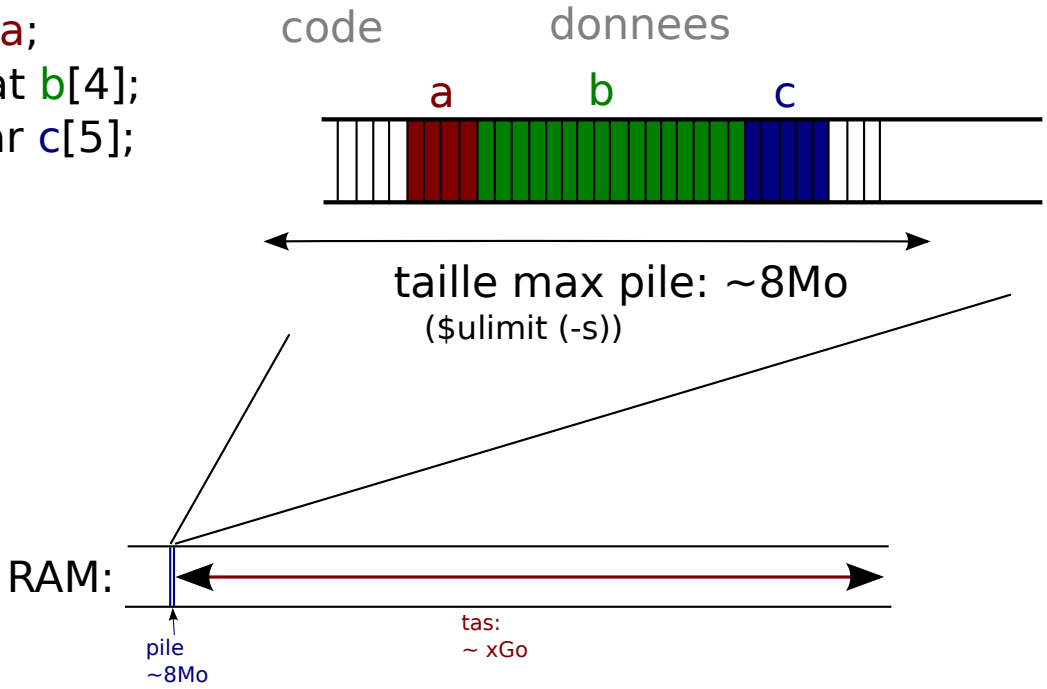
Structure de données:

chainage dynamique et graphes

Mémoire dynamique

Toutes les variables déclarées explicitement sont dans la pile (stack)

```
int a;
float b[4];
char c[5];
```

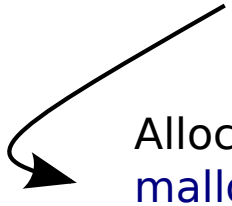


Mémoire dynamique

Il est possible d'allouer des emplacements mémoire dans le tas (heap)

Avantages: + Peut gérer les grandes quantitées de données
+ Peut allouer des tableaux de tailles variables

Inconvénient: - N'est pas géré automatiquement en C



Allocation/occupation d'espace mémoire par:
malloc(taille)

Désallocation/libération d'espace mémoire par:
free(adresse)

=> A la charge du programmeur de bien gérer l'allocation/libération
=> Attention: >90% des erreurs se font sur la gestion mémoire

Note: Le C n'est pas le langage le + adapté pour la gestion de mémoire simple

Mémoire dynamique

Exemple:

```c
int main()
{
  int *mon_tableau=NULL;
  mon_tableau=malloc(5*sizeof(int));

  if(mon_tableau==NULL)
    {printf("Erreur allocation memoire\n");exit(1);}

  int k=0;
  for(k=0;k<5;++k)
    mon_tableau[k]=2*k;

  for(k=0;k<5;++k)
    printf("%d\n",mon_tableau[k]);

  free(mon_tableau);
  mon_tableau=NULL;

  return 0;
}
```

allocation espace mémoire

vérification

utilisation comme un tableau standard

libération mémoire

[malloc/p1]

Mémoire dynamique

Exemple:

```c
int main()
{
  int *mon_tableau=NULL;
  mon_tableau=malloc(5*sizeof(int));

  if(mon_tableau==NULL)
    {printf("Erreur allocation memoire\n");exit(1);}

  int k=0;
  for(k=0;k<5;++k)
    mon_tableau[k]=2*k;

  for(k=0;k<5;++k)
    printf("%d\n",mon_tableau[k]);

  free(mon_tableau);
  mon_tableau=NULL;

  return 0;
}
```

erreur classique: malloc(5)
oublie taille de l'entier=4 octets

erreur classique: oublie verification

erreur classique: dépassement tableau

erreur classique: oublie libération
(perte d'espace RAM!)

[malloc/p1]

Mémoire dynamique

Exemple: tableau de taille non connue à la compilation

```c
int main()
{
  printf("Donnez un nombre de cases a allouer positif: ");

  int n=0;
  scanf("%d",&n);

  if(n<=0 || n>5000000)
    {
      printf("Nombre %d invalide\n",n);
      exit(1);
    }

  int *tableau=NULL;
  tableau=malloc(n*sizeof(int));

  printf("Je viens d'allouer dynamiquement un tableau de %d entier\n",n);

  int k=0;
  for(k=0;k<n;++k)
    tableau[k]=k;

  free(tableau);
  tableau=NULL;
}
```

Mémoire dynamique

Exemple: redimensionnement d'un tableau

```c
int main()
{
  int taille_1=500;
  tableau=malloc(taille_1*sizeof(float));

  if(tableau==NULL)
    {printf("Erreur allocation tableau\n");exit(1);}

  int k=0;
  for(k=0;k<taille_1;++k)//remplissage de 500 cases
    tableau[k]=cos((float)k/taille_1*2*M_PI);

  int taille_2=1000;
  copie_et_aggrandissement_tableau(taille_1,taille_2);

  //remplissage des 500 cases suivantes
  for(k=taille_1;k<taille_2;++k)
    tableau[k]=k*k;

  free(tableau);//liberation de l'espace memoire
  tableau=NULL;

  return 0;
}
```

```c
float* tableau=NULL;

void copie_et_aggrandissement_tableau(int ancienne_taille,
                                      int nouvelle_taille)
{
  float *tableau_temporaire=tableau;//copie du pointeur

  //allocation du tableau avec nouvelle taille
  tableau=NULL;
  tableau=malloc(nouvelle_taille*sizeof(float));

  if(tableau==NULL)
    {printf("Erreur allocation tableau\n");abort();}

  //copie des valeurs du tableau precedent
  int k=0;
  for(k=0;k<ancienne_taille;++k)
    tableau[k]=tableau_temporaire[k];

  //liberation du tableau precedent
  free(tableau_temporaire);
  tableau_temporaire=NULL;

}
```

[malloc/p3]

Mémoire dynamique

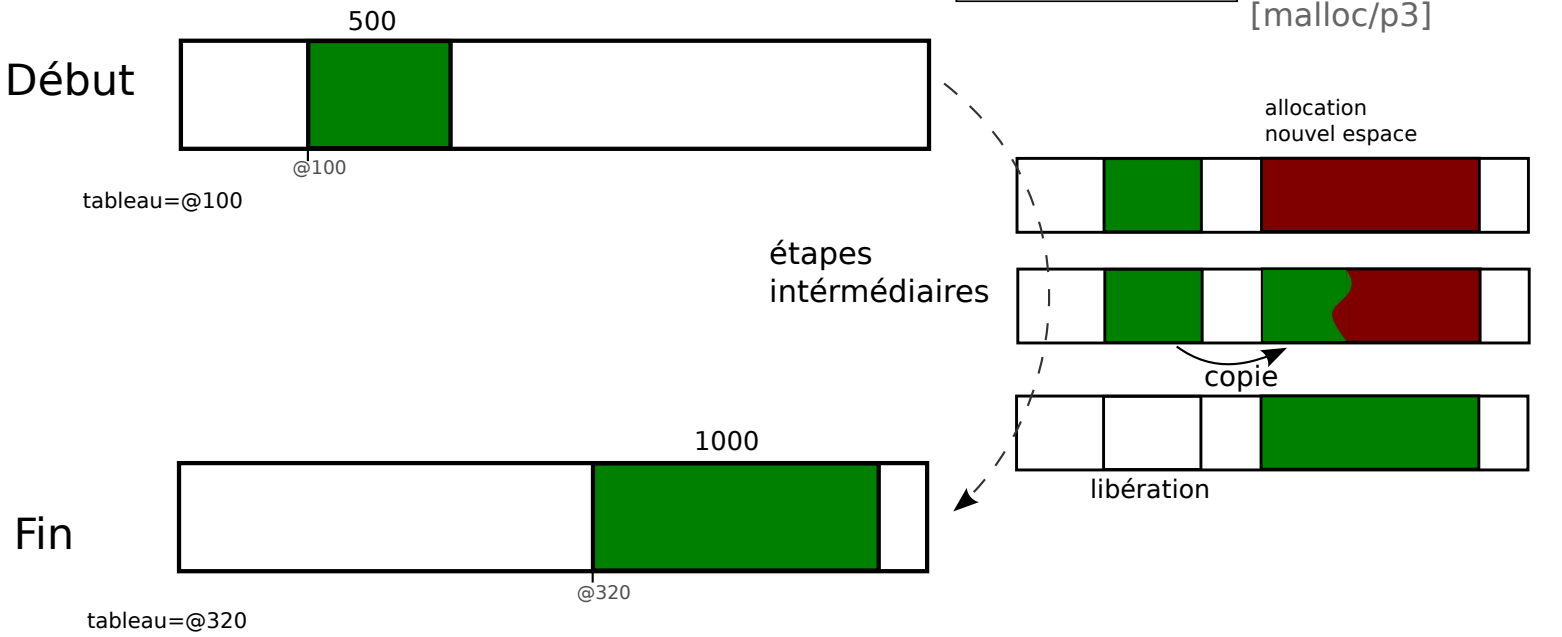
Exemple: redimensionnement d'un tableau

```c
int main()
{
  int taille_1=500;
  tableau=malloc(taille_1*sizeof(float));

  if(tableau==NULL)
    {printf("Erreur allocation tableau\n");exit(1);}

  int k=0;
  for(k=0;k<taille_1;++k)//remplissage de 500 cases
    tableau[k]=cos((float)k/taille_1*2*M_PI);

  int taille_2=1000;
  copie_et_aggrandissement_tableau(taille_2);

  //remplissage des 500 cases suivantes
  for(k=taille_1;k<taille_2;++k)
    tableau[k]=k*k;

  free(tableau);//liberation de l'espace memoire
  tableau=NULL;

  return 0;
}
```

```c
float *tableau=NULL;

void copie_et_aggrandissement_tableau(int nouvelle_taille)
{
  float *tableau_temporaire=tableau;//copie du pointeur

  //allocation du tableau avec nouvelle taille
  tableau=NULL;
  tableau=malloc(nouvelle_taille);

  if(tableau==NULL)
    {printf("Erreur allocation tableau\n");exit(1);}

  //copie des valeurs du tableau precedent
  int k=0;
  for(k=0;k<nouvelle_taille;++k)
    tableau[k]=tableau_temporaire[k];

  //liberation du tableau precedent
  free(tableau_temporaire);
  tableau_temporaire=NULL;
}
```

[malloc/p3]

Principe:



Attention: Aggrandissement de tableau=
nouvelle allocation+copie = long !

Mémoire dynamique

Exemple: mémoire dynamique sur une struct

```c
struct type_tronc
{
  float epaisseur_ecorce;
  int nombre_anneaux;
};
struct type_feuilles
{
  float largeur;
  float longeur;
};
struct type_arbre
{
  struct type_tronc tronc;
  struct type_feuilles feuilles;
};

int main()
{
  struct type_arbre *foret=NULL;
  foret=malloc(3*sizeof(struct type_arbre));

  foret[0].tronc.epaisseur_ecorce=1.5;
  foret[1].feuilles.largeur=5.0;

  free(foret);
  foret=NULL;

  return 0;
}
```

[malloc/p4]

Mémoire dynamique

Allocation

→ **Structure de données:**

chainage dynamique et graphes

Mémoire dynamique

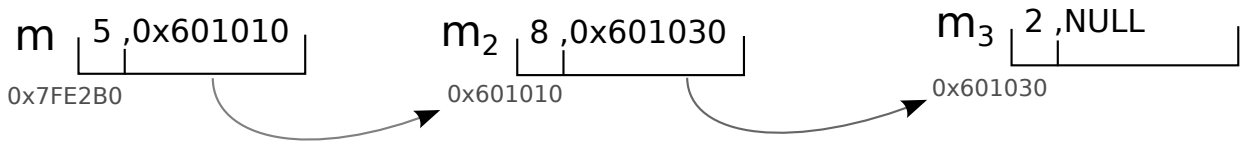
Exemple: structure de données avancées: liste chaînées

```c
struct maillon
{
  int valeur;
  struct maillon *suivant;
};

int main()
{
  struct maillon m;
  m.valeur=5;
  m.suivant=malloc(sizeof(struct maillon));

  struct maillon* m2=NULL;
  m2=m.suivant;
  m2->valeur=8;

  m2->suivant=malloc(sizeof(struct maillon));

  struct maillon* m3=NULL;
  m3=m2->suivant;
  m3->valeur=2;

  printf("%d %d %d\n",m.valeur,
     m.suivant->valeur,
     m.suivant->suivant->valeur);

  printf("%d %d %d\n",m.valeur,m2->valeur,m3->valeur);

  return 0;
}
```

```
m                                   
    valeur                  5
  ↪ suivant                 0x601010
    *0x601010
        valeur              8
      ↪ suivant             0x601030
        *0x601030
            valeur          2
          ↪ suivant         0x0
            *0x0            Cannot access memory at address 0x0
↪ m2                        0x601010
  *0x601010
      valeur                8
    ↪ suivant               0x601030
↪ m3                        0x601030
  *0x601030
      valeur                2
    ↪ suivant               0x0
```

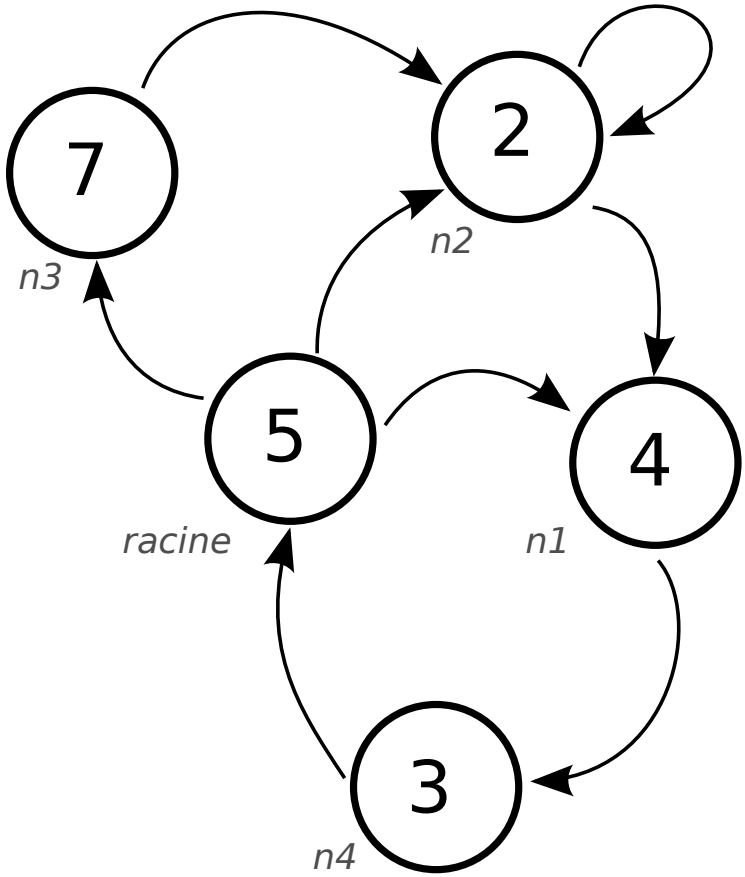
[malloc/p5]

("tableau" de taille quelconque
qui peut s'aggrandir en O(1))



Mémoire dynamique

Exemple: structure de données avancées: graphe



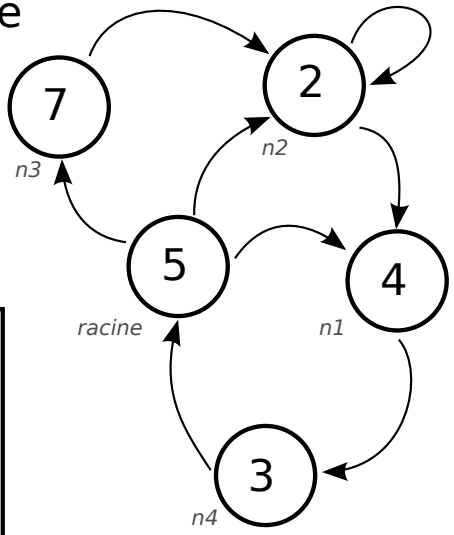

```
struct noeud
{
  int valeur;

  int nombre_fils;
  struct noeud **fils;
};
```

[malloc/p6]

Mémoire dynamique

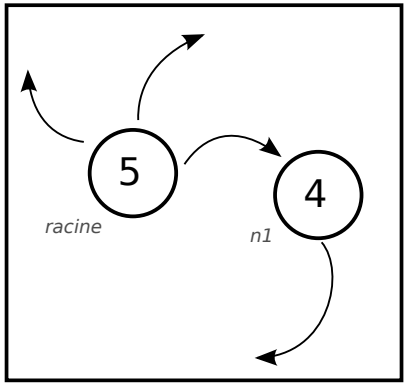
Exemple: structure de données avancées: graphe

```
struct noeud
{
  int valeur;

  int nombre_fils;
  struct noeud **fils;
};
```



[malloc/p6]

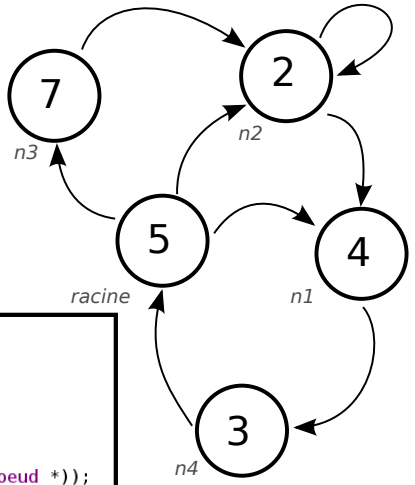
```
int main()
{
    //allocation
    struct noeud *racine=NULL;
    racine=malloc(sizeof(struct noeud));
    assert(racine!=NULL);

    racine->valeur=5;
    racine->nombre_fils=3;
    racine->fils=NULL;
    racine->fils=malloc(racine->nombre_fils*sizeof(struct noeud *));
    assert(racine->fils!=NULL);

    struct noeud* n1=NULL;
    n1=malloc(sizeof(struct noeud));
    assert(n1!=NULL);
    n1->valeur=4;
    racine->fils[0]=n1;
    n1->nombre_fils=1;
    n1->fils=NULL;
    n1->fils=malloc(n1->nombre_fils*sizeof(struct noeud *));
    assert(n1->fils!=NULL);
```



Mémoire dynamique

Exemple: structure de données avancées: graphe

```c
struct noeud
{
  int valeur;

  int nombre_fils;
  struct noeud **fils;
};
```




```c
int main()
{
    //allocation
    struct noeud *racine=NULL;
    racine=malloc(sizeof(struct noeud));
    assert(racine!=NULL);

    racine->valeur=5;
    racine->nombre_fils=3;
    racine->fils=NULL;
    racine->fils=malloc(racine->nombre_fils*sizeof(struct noeud *));
    assert(racine->fils!=NULL);

    struct noeud* n1=NULL;
    n1=malloc(sizeof(struct noeud));
    assert(n1!=NULL);
    n1->valeur=4;
    racine->fils[0]=n1;
    n1->nombre_fils=1;
    n1->fils=NULL;
    n1->fils=malloc(n1->nombre_fils*sizeof(struct noeud *));
    assert(n1->fils!=NULL);
```

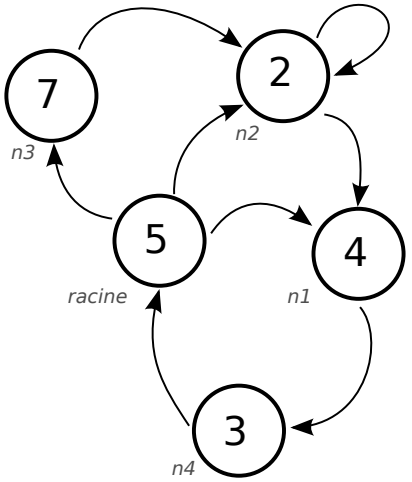
[malloc/p6]

```c
struct noeud* n2=NULL;
n2=malloc(sizeof(struct noeud));
assert(n2!=NULL);
n2->valeur=2;
n2->nombre_fils=2;
n2->fils=NULL;

n2->fils=malloc(n2->nombre_fils*sizeof(struct noeud *));
n2->fils[0]=n1;
n2->fils[1]=n2;

racine->fils[1]=n2;

struct noeud* n3=NULL;
n3=malloc(sizeof(struct noeud));
assert(n3!=NULL);
n3->valeur=7;
n3->nombre_fils=1;
n3->fils=NULL;
n3->fils=malloc(sizeof(struct noeud*));
assert(n3!=NULL);
n3->fils[0]=n2;

racine->fils[2]=n3;

struct noeud* n4=NULL;
n4=malloc(sizeof(struct noeud));
assert(n4!=NULL);
n4->valeur=3;
n4->nombre_fils=1;
n4->fils=NULL;
n4->fils=malloc(n4->nombre_fils*sizeof(struct noeud*));
assert(n4->fils!=NULL);
n4->fils[0]=racine;


n1->fils[0]=n4;
```

Mémoire dynamique

Exemple: structure de données avancées: graphe

```c
struct noeud
{
  int valeur;

  int nombre_fils;
  struct noeud **fils;
};
```

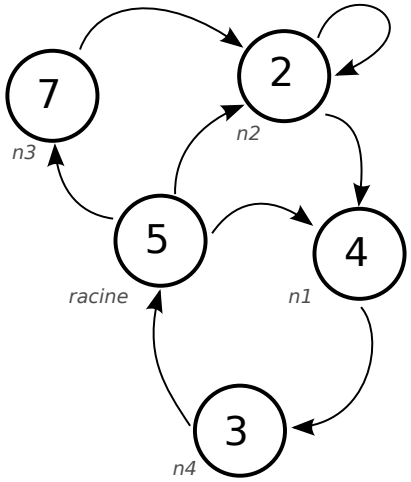



```c
//utilisation
int valeur_racine=racine->valeur;
int valeur_n4=racine->fils[1]->fils[0]->fils[0]->valeur;

printf("%d %d\n",valeur_racine,valeur_n4);
```

[malloc/p6]

Mémoire dynamique

Exemple: structure de données avancées: graphe




```c
struct noeud
{
  int valeur;

  int nombre_fils;
  struct noeud **fils;
};
```

[malloc/p6]

```c
    //desallocation
    free(racine->fils);
    free(n1->fils);
    free(n4->fils);
    free(n3->fils);
    free(n2->fils);

    free(racine);
    free(n1);
    free(n2);
    free(n3);
    free(n4);

    return 0;
}
```

Ne pas oublier de libérer la mémoire
=> 10 malloc, donc 10 free !

Vérification par valgrind (absence fuite mémoire)
Obligatoire!

```
HEAP SUMMARY:
    in use at exit: 0 bytes in 0 blocks
  total heap usage: 10 allocs, 10 frees, 144 bytes allocated

All heap blocks were freed -- no leaks are possible
```

Exercice difficile!