Pointeurs : gestion dynamique de la mémoire[1]

Salem BENFERHAT

Centre de Recherche en Informatique de Lens (CRIL-CNRS)
email : benferhat@cril.fr

[1] Version préliminaire du cours. Tout retour sur la forme comme sur le fond est le bienvenu.

Variables, types
espaces mémoire et adresses

Les variables ...

Types

- Les langages de programmation permettent de décrire des données de natures et complexités différentes en utilisant la notion de *type*
- Des exemples de type de base en langage C sont :
  - Char
  - Int
  - unsigned int
  - Short
  - Long
  - unsigned long
  - float
  - double
  - long double

Les variables ...

Les variables

- A chaque fois qu'une variable de type T (de base ou complexe) est déclarée un espace mémoire lui est réservé.
- En langage C, la commande sizeof permet de connaître l'espace mémoire associé à chaque type.

Taille des données

```
1  void taille_des_types ()
2  {
3  printf ("char = %lu byte\n", sizeof(char));
4  printf ("short = %lu bytes\n", sizeof(short));
5  printf ("int = %lu bytes\n", sizeof(int));
6  printf ("unsigned int = %lu bytes\n",sizeof(unsigned int));
7  printf ("long = %lu bytes\n", sizeof(long));
8  printf ("unsigned long = %lu bytes\n",sizeof(unsigned long));
9  printf ("float = %lu bytes \n", sizeof(float));
10 printf ("double = %lu bytes\n", sizeof(double));
11 printf ("long double = %lu bytes\n", sizeof(long double));
12 }
```

Espace mémoire pour les variables

Espace mémoire

L'exécution du programme donne l'espace mémoire qui sera réservé pour chaque type :

char	=	1 byte
short	=	2 bytes
int	=	4 bytes
unsigned int	=	4 bytes
long	=	8 bytes
unsigned long	=	8 bytes
float	=	4 bytes
double	=	8 bytes
long double	=	16 bytes

Adresse

Emplacement des données

- Les programmes, les fonctions, les variables et les données sont stockés en mémoire.
- La mémoire peut-être vue comme un grand tableau où chaque case contient généralement un mot mémoire.
- Les indices de ce tableau sont appelés des adresses.
- On accède aux informations grâce à ces adresses.

L'opérateur &

L'opérateur &, appliqué à une variable ou à une fonction, permet de récupérer l'adresse de la variable ou de la fonction.

Exemple sur l'opérateur &

Adresse des variables

Grâce à l'opérateur &, on peut récupérer les adresses des variables locales et paramètres utilisés dans la fonction Estpremiernaif.

```
int Estpremiernaif( int n)
{
 int i;
 if( n <= 1) return 0;
 printf ("Dans la fonction Estpremiernaif :\n");
 printf ("la variable n se trouve a l'adresse %p :\n", &n);
 printf ("la variable local i se trouve a l'adresse %p :\n", &i);
 for(i = 2; i <= n; ++i)
 {
  if(n % i == 0) return 0;
 }
 return 1;
}
```

Exemple sur l'opérateur &

Adresse

- L'exécution de la fonction donne :

  Dans la fonction Estpremiernaif :

  la variable n se trouve a l'adresse : 0x7fff61f5cb2c
  la variable local i se trouve a l'adresse : 0x7fff61f5cb20

- Les adresses sont données en hexadécimal

Exemple sur l'opérateur & (suite)

Adresse des variables

Grâce à l'opérateur &, on peut également récupérer les adresses des fonctions.

```
2
3  int main ()
4  {
5  int nombre;
6  printf ("Introduire un nombre : ");
7  scanf ("%d", &nombre);
8  printf ("l\'adresse de la variable nombre est : %p \n", &nombre);
9  printf ("l\'adresse de la fonction Estpremiernaif est : %p \n",
0                &Estpremiernaif);
   printf ("l\'adresse de la fonction main est : %p \n", &main);
2  if (Estpremiernaif(nombre)==1)
3   printf ("Le nombre %d est premier : ", nombre);
4   else printf ("Le nombre %d n est pas premier : ", nombre);
5  return(0);
6  }
```

Exemple sur l'opérateur & (suite)

L'adresse

- L'exécution de la fonction donne :

l'adresse de la variable nombre est :	0x7fff6d03db44
l'adresse de la fonction Estpremiernaif est :	0x10d43eba0
l'adresse de la fonction main est :	0x10d43ec50

Exemple sur l'opérateur & (suite)

L'adresse

- L'exécution de la fonction donne :

l'adresse de la variable nombre est :	0x7fff6d03db44
l'adresse de la fonction Estpremiernaif est :	0x10d43eba0
l'adresse de la fonction main est :	0x10d43ec50

Remarque

- La variable *nombre* utilisée dans le programme principal et le paramètre *n* de la fonction *Estpremiernaif* n'ont pas la même adresse.
- De ce fait, toute modification sur *n* n'affectera pas la variable *nombre*.

Gestion dynamique de la mémoire

Pointeur

- Pour manipuler les adresses mémoire nous avons besoin de variables particulières appelées *pointeurs*.

Gestion dynamique de la mémoire

Pointeur

- Pour manipuler les adresses mémoire nous avons besoin de variables particulières appelées *pointeurs*.
- Un pointeur est donc une variable qui contient l'adresse d'une donnée contenue en mémoire.

Gestion dynamique de la mémoire

Pointeur

- Pour manipuler les adresses mémoire nous avons besoin de variables particulières appelées *pointeurs*.
- Un pointeur est donc une variable qui contient l'adresse d'une donnée contenue en mémoire.
- Un pointeur aura une valeur entière (qui représente une adresse mémoire).

Gestion dynamique de la mémoire

Pointeur

- Pour manipuler les adresses mémoire nous avons besoin de variables particulières appelées *pointeurs*.
- Un pointeur est donc une variable qui contient l'adresse d'une donnée contenue en mémoire.
- Un pointeur aura une valeur entière (qui représente une adresse mémoire).
- Grâce aux pointeurs, on sépare le contenu (une valeur) du contenant (une adresse).

Organisation de la mémoire

3 parties

Pour faire simple, la mémoire est divisée en 3 parties :

Organisation de la mémoire

3 parties

Pour faire simple, la mémoire est divisée en 3 parties :

- Une partie statique contenant principalement le code des fonctions.

Organisation de la mémoire

3 parties

Pour faire simple, la mémoire est divisée en 3 parties :

- Une partie statique contenant principalement le code des fonctions.
- Une Pile (Stack) qui contient principalement les variables créés statiquement (au niveau de la compilation).

Organisation de la mémoire

3 parties

Pour faire simple, la mémoire est divisée en 3 parties :

- Une partie statique contenant principalement le code des fonctions.
- Une Pile (Stack) qui contient principalement les variables créés statiquement (au niveau de la compilation).
- Un Tas (Heap) qui contient les variables créées dynamiquement.

Gestion dynamique de la mémoire

Déclaration

```
Type *Nom_de_la_variable;
```

Gestion dynamique de la mémoire

Déclaration

```
Type *Nom_de_la_variable;
```

- Le type peut-être de type simple (entier, réel, ...) ou complexe.

Gestion dynamique de la mémoire

Déclaration

Type *Nom_de_la_variable;

- Le type peut-être de type simple (entier, réel, ...) ou complexe.
- La présence de "*" indique que l'on ne travaille plus sur une variable normale, mais sur un pointeur.

Gestion dynamique de la mémoire

Déclaration

Type *Nom_de_la_variable;

- Le type peut-être de type simple (entier, réel, ...) ou complexe.
- La présence de "*" indique que l'on ne travaille plus sur une variable normale, mais sur un pointeur.
- Au niveau de la compilation (allocation statique) :
  - seul l'espace dédié au pointeur est réservé.
  - La taille de cet espace est fixe et
  - elle est complètement indépendante du type de données "pointé" par la variable pointeur.

Taille d'une variable de type pointeur

```
2
3  void taille_pointeurs()
4  {
5   int *pointeur1;
6   long double *pointeur2;
7   printf ("la taille d'un pointeur sur un type \n");
8   printf ("entier est de : %lu Bytes \n", sizeof(pointeur1));
9   printf ("long double est de : %lu Bytes \n", sizeof(pointeur2));
0  }
```

Taille d'une variable de type pointeur

Exemple

- L'exécution du programme donne : la taille d'un pointeur
  sur un type entier est de : 8 Bytes
  sur un type long double est de : 8 Bytes

Taille d'une variable de type pointeur

Exemple

- L'exécution du programme donne : la taille d'un pointeur
  sur un type entier est de : 8 Bytes
  sur un type long double est de : 8 Bytes

- La taille réservée à une variable de type pointeur est la même.

Adresse et valeur

- Nous avons déjà vu que si A est une variable de type T, alors

  &T

  donne l'adresse de l'emplacement de T dans la mémoire.

Adresse et valeur

- Nous avons déjà vu que si A est une variable de type T, alors

  &T

  donne l'adresse de l'emplacement de T dans la mémoire.

- Si p est une variable de type pointeur vers une donnée de type T, alors

  *p

  donne le contenu de l'emplacement mémoire dont l'adresse est p.

Adresse et valeur

- Nous avons déjà vu que si A est une variable de type T, alors

  &T

  donne l'adresse de l'emplacement de T dans la mémoire.

- Si p est une variable de type pointeur vers une donnée de type T, alors

  *p

  donne le contenu de l'emplacement mémoire dont l'adresse est p.

- Il est possible des opérations sur les pointeurs.

Affectations

- Soit p_1 et p_2 deux pointeurs du même type T. Soit A une variable de type T. Alors, les opérations suivantes sont permises :
- **Affectation d'adresse à un pointeur** :

Affectations

- Soit p_1 et p_2 deux pointeurs du même type T. Soit A une variable de type T. Alors, les opérations suivantes sont permises :
- **Affectation d'adresse à un pointeur** :
  - $p_1 = p_2$

Affectations

- Soit p_1 et p_2 deux pointeurs du même type T. Soit A une variable de type T. Alors, les opérations suivantes sont permises :
- **Affectation d'adresse à un pointeur** :
  - $p_1 = p_2$
  - $p_1 = \&A$

Affectations

- Soit p_1 et p_2 deux pointeurs du même type T. Soit A une variable de type T. Alors, les opérations suivantes sont permises :
- **Affectation d'adresse à un pointeur** :
  - $p_1 = p_2$
  - $p_1 = \&A$
  - $p_1 = \text{NULL}$

Affectations

- Soit p_1 et p_2 deux pointeurs du même type T. Soit A une variable de type T. Alors, les opérations suivantes sont permises :
- **Affectation d'adresse à un pointeur** :
  - $p_1 = p_2$
  - $p_1 = \&A$
  - $p_1 = \text{NULL}$
- **Affectation de valeur** :
  - $*p_1 = *p_2$

Affectations

- Soit p_1 et p_2 deux pointeurs du même type T. Soit A une variable de type T. Alors, les opérations suivantes sont permises :
- **Affectation d'adresse à un pointeur** :
  - $p_1 = p_2$
  - $p_1 = \&A$
  - $p_1 = \text{NULL}$
- **Affectation de valeur** :
  - $*p_1 = *p_2$
  - $*p_1 = A$

Affectations

- Soit p_1 et p_2 deux pointeurs du même type T. Soit A une variable de type T. Alors, les opérations suivantes sont permises :
- **Affectation d'adresse à un pointeur** :
  - $p_1 = p_2$
  - $p_1 = \&A$
  - $p_1 = \text{NULL}$
- **Affectation de valeur** :
  - $*p_1 = *p_2$
  - $*p_1 = A$
  - $A = *p_1$

Allocations

Soit p_1 un pointeur vers des données type T.

Allocations

Soit p_1 un pointeur vers des données type T.

- Allouer (p_1);

Allocations

Soit p_1 un pointeur vers des données type T.

- Allouer (p_1);
- Libérer (p_1);

Exemple

Trace

Regardons la trace de la séquence suivante.

```
int i;
int *p;
i=30;
p=&i;
i=412;
p=(int *) malloc (sizeof (int));
*p=20;
free(p);
```

- On imprimera les emplacements des variables (tas, pile).
- Pour la variable entière *i* on imprimera son adresse mémoire ”&i” et son contenu ”i”.
- Pour la variable de type pointeur *p* on imprimera son adresse mémoire ”&p”, son contenu ”p” et l'élément sur lequel elle pointe ”*p”.

Déroulement du programme

Exécution de l'instruction

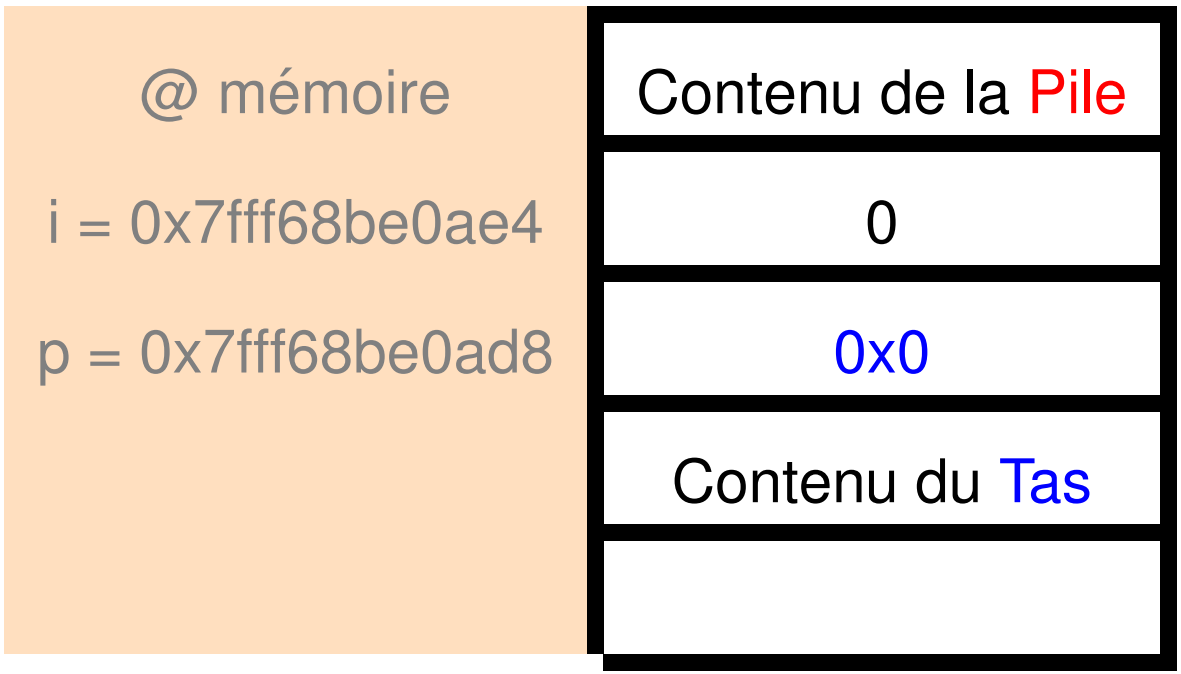
```
int i;
```

@ mémoire	Contenu de la Pile
i = 0x7fff68be0ae4	0
	Contenu du Tas

Déroulement du programme

Exécution de l'instruction

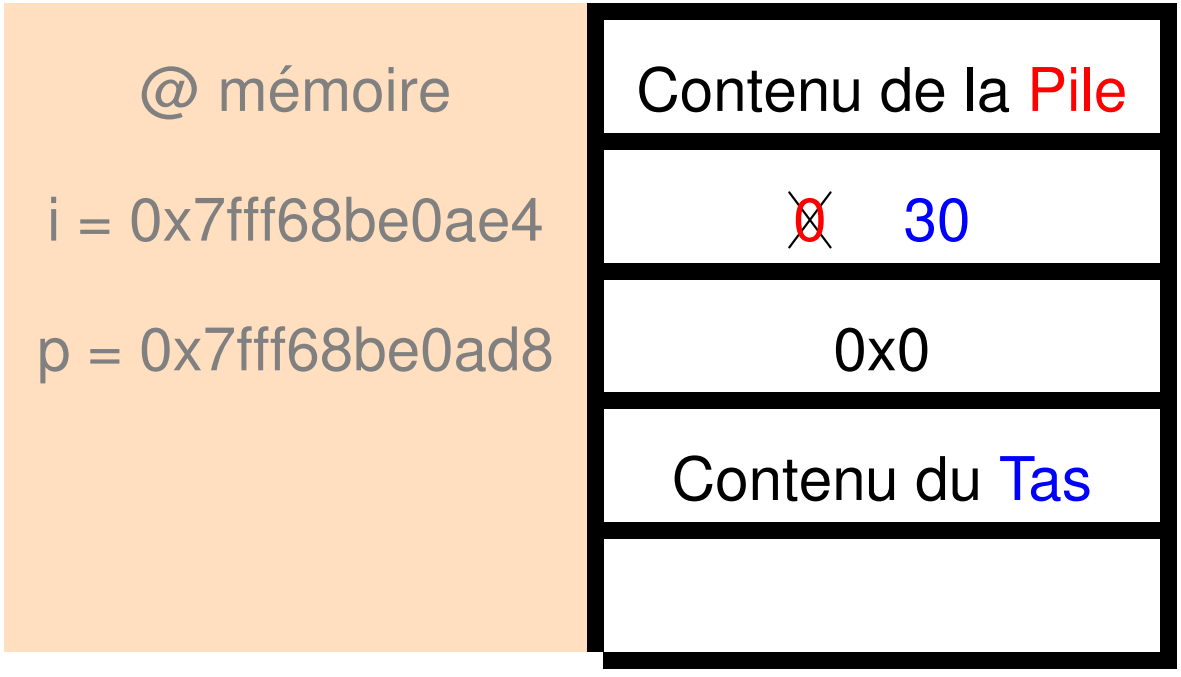
```
int *p;
```



Déroulement du programme

Exécution de l'instruction

i=30;



Déroulement du programme

Exécution de l'instruction

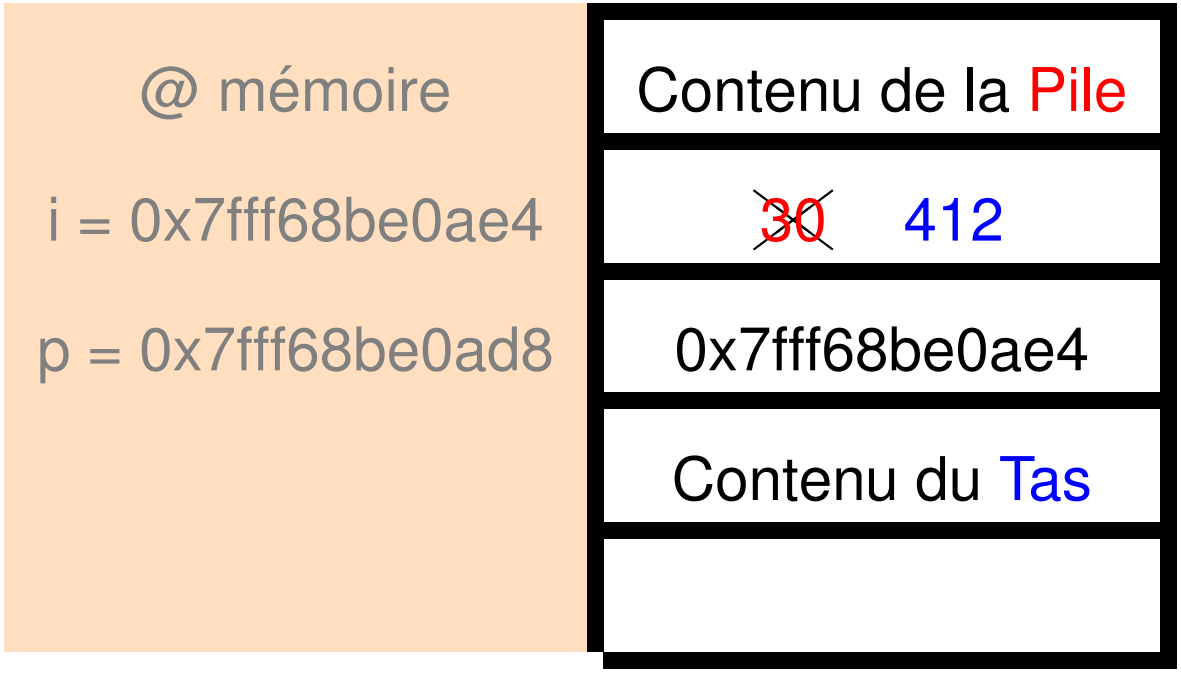
p=&i;

@ mémoire	Contenu de la Pile
i = 0x7fff68be0ae4	30
p = 0x7fff68be0ad8	~~0x0~~ 0x7fff68be0ae4
	Contenu du Tas

Déroulement du programme

Exécution de l'instruction

i=412;



Déroulement du programme

Exécution de l'instruction

```
p=(int *) malloc (sizeof (int));
```

@ mémoire	Contenu de la Pile
i = 0x7fff68be0ae4	412
p = 0x7fff68be0ad8	~~0x7fff68be0ae4~~ 0x109100900
	Contenu du Tas
0x109100900	0

Déroulement du programme

Exécution de l'instruction

`*p=20;`

@ mémoire	Contenu de la Pile
i = 0x7fff68be0ae4	412
p = 0x7fff68be0ad8	0x109100900
	Contenu du Tas
0x109100900	~~0~~ 20

Déroulement du programme

Exécution de l'instruction

free(p);

@ mémoire	Contenu de la Pile
i = 0x7fff68be0ae4	412
p = 0x7fff68be0ad8	0x109100900
	Contenu du Tas
~~0x109100900~~	~~20~~

Enregistrement et liste chaînée

Enregistrement ou structure

Définitions

La structure ou l'enregistrement permet de décrire un type complexe et de regrouper différentes types en un seul.

- On déclare une structure avec le mot cléf "struct". Par exemple :

```
struct etudiant
{
 int age;
 char *niveau_etude;
};
```

Enregistrement ou structure

Remarques

Enregistrement ou structure

Remarques

- Il n'y a pas d'espace mémoire réservé pour la définition d'un type enregistrement.

Enregistrement ou structure

Remarques

- Il n'y a pas d'espace mémoire réservé pour la définition d'un type enregistrement.
- On déclare une variable A de type enregistrement par :
  - struct etudiant A;
- L'accès à un champs de A se fait grâce à l'opérateur ".". Par exemple, A.age permet d'accéder au champs "age" et se comporte comme une variable de type entier.

Enregistrement ou structure

Remarques

- Il n'y a pas d'espace mémoire réservé pour la définition d'un type enregistrement.
- On déclare une variable A de type enregistrement par :
  - struct etudiant A;
- L'accès à un champs de A se fait grâce à l'opérateur ".". Par exemple, A.age permet d'accéder au champs "age" et se comporte comme une variable de type entier.
- Bien sûr on peut déclarer un pointeur vers une variable de type enregistrement.

Structure récursive avec pointeur

Une structure récursive avec pointeur *r* est une structure composée de deux parties :

Structure récursive avec pointeur

Une structure récursive avec pointeur *r* est une structure composée de deux parties :

- La première partie contient l'élément à stocker qui peut-être de type simple (un entier par exemple) ou de type complexe (un enregistrement ou un tableau).

Structure récursive avec pointeur

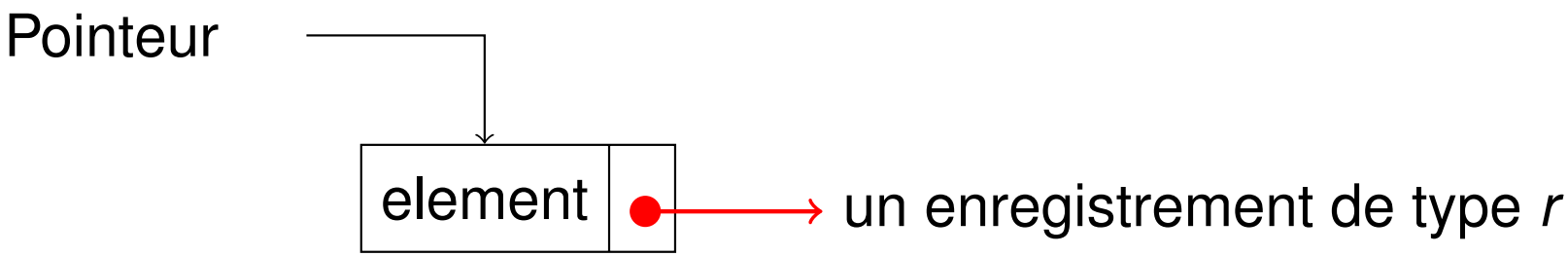
Une structure récursive avec pointeur *r* est une structure composée de deux parties :

- La première partie contient l'élément à stocker qui peut-être de type simple (un entier par exemple) ou de type complexe (un enregistrement ou un tableau).
- La deuxième partie contient une variable de type pointeur, qui au fait ”pointe” vers un élément de type *r*. Ce pointeur est essentiel pour enchainer différents éléments de type *r*.

Structure récursive avec pointeur

Une structure récursive avec pointeur *r* est une structure composée de deux parties :

- La première partie contient l'élément à stocker qui peut-être de type simple (un entier par exemple) ou de type complexe (un enregistrement ou un tableau).
- La deuxième partie contient une variable de type pointeur, qui au fait "pointe" vers un élément de type *r*. Ce pointeur est essentiel pour enchainer différents éléments de type *r*.



Pomme, pêche, poire, abricot

Jeu avec une liste chaînée

Ecrire une fonction qui code le jeu avec une représentation par la liste chaînée circulaire.
Il s'agit d'écrire une fonction qui prend en paramètre le nombre de joueurs, les stocke dans une liste chaînée.

Jeu avec une liste chaînée : partie déclaration

```
2  struct noeud_joueur
3  {
4    int numero_joueur;
5    struct noeud_joueur *suivant;
6  };
7
8  typedef struct noeud_joueur joueur;
```

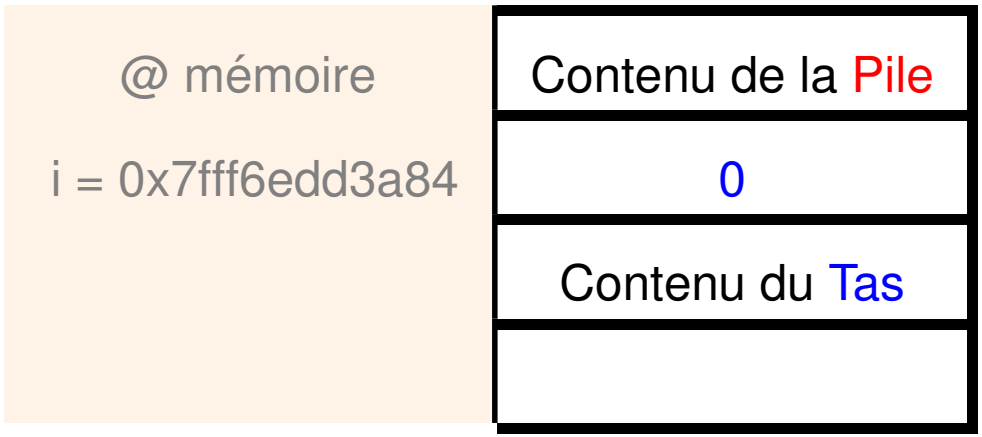
Jeu avec une liste chaînée : création de la liste

```
joueur *creation_de_liste(int N)
{
 int i;
 joueur *p, *debut;
 for (i=0; i<N; i++)
 {
  if (i == 0)
  {
   debut = (joueur *)malloc(sizeof(joueur));
   p = debut;
  }
  else
  {
   p->suivant = (joueur *) malloc(sizeof(joueur));
   p = p->suivant;
  }
  p->numero_joueur=i;
 }
 p->suivant = debut;
 return (debut);
}
```

Trace de la fonction création

- Pour la trace on prendra :
  - Nombre de joueur = 3.
  - M=2
- On imprimera les emplacements des variables (tas, pile).
- Pour la variable entière *i* on imprimera son adresse mémoire ”&i” et son contenu ”i”.
- Pour les variables de type pointeur *p* et *debut* on imprimera leur adresse mémoire, leur contenu et les éléments sur lesquels elles pointent.

Instruction : int i=0;



Instruction : joueur *p;

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	0
p = 0x7fff6edd3a78	0x1
	Contenu du Tas

Instruction : joueur *debut;

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	0
p = 0x7fff6edd3a78	0x1
debut = 0x7fff6edd3a70	0x0
	Contenu du Tas

Instruction : for (i=0; i<N; i++)

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	0
p = 0x7fff6edd3a78	0x1
debut = 0x7fff6edd3a70	0x0
	Contenu du Tas

Instruction : debut = (joueur *)malloc(sizeof(joueur));

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	0
p = 0x7fff6edd3a78	0x1
debut = 0x7fff6edd3a70	~~0x0~~ 0x10f300900
	Contenu du Tas
0x10f300900	0 0x0

Instruction : p = debut;

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	0
p = 0x7fff6edd3a78	~~0x1~~ 0x10f300900
debut = 0x7fff6edd3a70	0x10f300900
	Contenu du Tas
0x10f300900	0 0x0

Instruction : p− >numero_joueur=i;

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	0
p = 0x7fff6edd3a78	0x10f300900
debut = 0x7fff6edd3a70	0x10f300900
	Contenu du Tas
0x10f300900	0 0x0

Instruction : for (i=0; i<N; i++)

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	~~0~~ 1
p = 0x7fff6edd3a78	0x10f300900
debut = 0x7fff6edd3a70	0x10f300900
	Contenu du Tas
0x10f300900	0 0x0

Instruction : p– >suivant = (joueur *) malloc(sizeof(joueur));

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	1
p = 0x7fff6edd3a78	0x10f300900
debut = 0x7fff6edd3a70	0x10f300900
	Contenu du Tas
0x10f300910	0 0x0
0x10f300900	0 ~~0x0~~ 0x10f300910

Instruction : p = p– >suivant;

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	1
p = 0x7fff6edd3a78	~~0x10f300900~~ 0x10f300910
debut = 0x7fff6edd3a70	0x10f300900
	Contenu du Tas
0x10f300910	0
	0x0
0x10f300900	0
	0x10f300910

Instruction : p– >numero_joueur=i;

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	1
p = 0x7fff6edd3a78	0x10f300910
debut = 0x7fff6edd3a70	0x10f300900
	Contenu du Tas
0x10f300910	~~0~~ 1 0x0
0x10f300900	0 0x10f300910

Instruction : for (i=0; i<N; i++)

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	~~1~~ 2
p = 0x7fff6edd3a78	0x10f300910
debut = 0x7fff6edd3a70	0x10f300900
	Contenu du Tas
0x10f300910	1
	0x0
0x10f300900	0
	0x10f300910

Instruction : p– >suivant = (joueur *) malloc(sizeof(joueur));

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	2
p = 0x7fff6edd3a78	0x10f300910
debut = 0x7fff6edd3a70	0x10f300900
	Contenu du Tas
0x10f300920	0 0x0
0x10f300910	1 ~~0x0~~ 0x10f300920
0x10f300900	0 0x10f300910

Instruction : p = p– >suivant;

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	2
p = 0x7fff6edd3a78	~~0x10f300910~~ 0x10f300920
debut = 0x7fff6edd3a70	0x10f300900
	Contenu du Tas
0x10f300920	0 0x0
0x10f300910	1 0x10f300920
0x10f300900	0 0x10f300910

Instruction : p– >numero_joueur=i;

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	2
p = 0x7fff6edd3a78	0x10f300920
debut = 0x7fff6edd3a70	0x10f300900
	Contenu du Tas
0x10f300920	~~0~~ 2 0x0
0x10f300910	1 0x10f300920
0x10f300900	0 0x10f300910

Instruction : p– >suivant = debut;

@ mémoire	Contenu de la Pile
i = 0x7fff6edd3a84	3
p = 0x7fff6edd3a78	0x10f300920
debut = 0x7fff6edd3a70	0x10f300900
	Contenu du Tas
0x10f300920	2 ~~0x0~~ 0x10f300900
0x10f300910	1 0x10f300920
0x10f300900	0 0x10f300910

Jeu avec une liste chaînée

Ecrire une fonction qui :

- prend en paramètre une liste chaînée circulaire qui contient les identifiants des joueurs,
- supprime, au fur et à mesure, de la liste les joueurs éliminés et
- retourne l'identifiant du gagnant.

Jeu avec une liste chaînée : partie jeu

```
int jeu(int M, joueur *debut)
{
 int i;
 joueur *p=debut, *precedent;
 while (p->suivant !=p)
 {
  for (i=0; i<M; i++)
  {
   precedent=p;
   p=p->suivant;
  }
  precedent->suivant=p->suivant;
  free (p);
  p=precedent->suivant;

 }
 return p->numero_joueur;
}
```

Déroulement détaillé de la fonction jeu

Trace très détaillée de la fonction jeu

- Pour la trace on utilisera une représentation graphique à base de maillons. Nous choisirons :
  - Nombre de joueur = 5.
  - M=3
- Pour chaque instruction :
  - On affichera les valeurs des variables entières ”i” et *M*.
  - On suivra graphiquement l'évolution des pointeurs *p* et *precedent*.

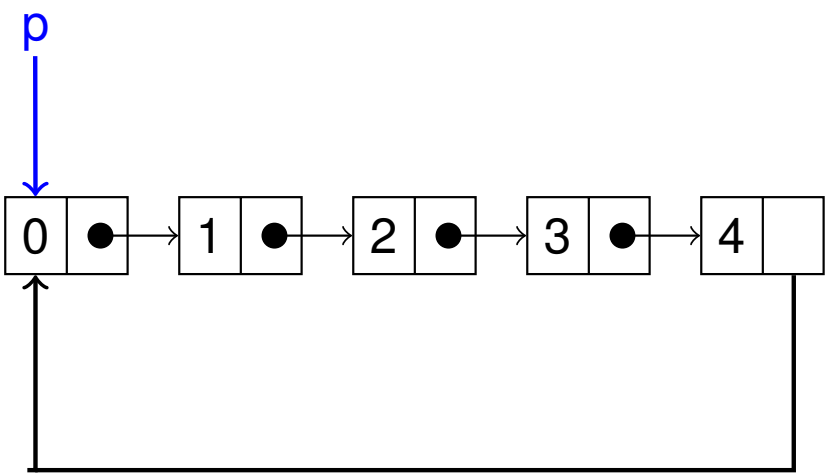
Déroulement de l'algorithme

Exécution de l'instruction

Partie Déclaration :

```
int       i=0;
joueur    *p=debut;
joueur    *precedent=NULL;
```

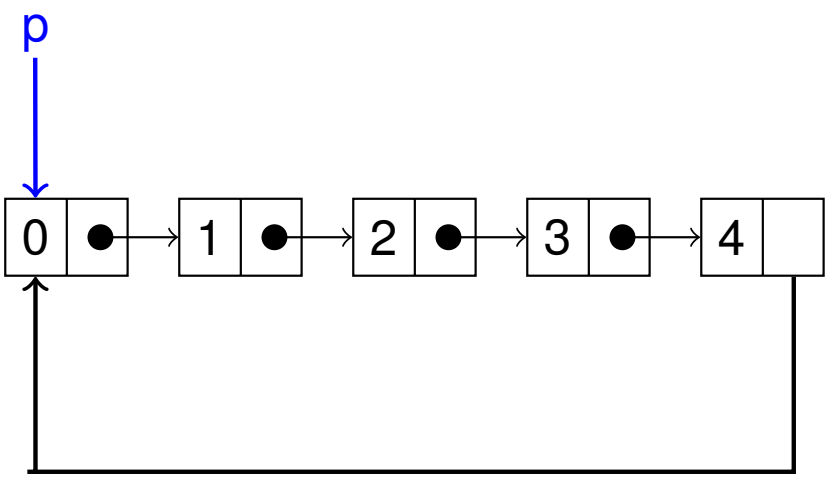
i	M
0	3



Déroulement de l'algorithme

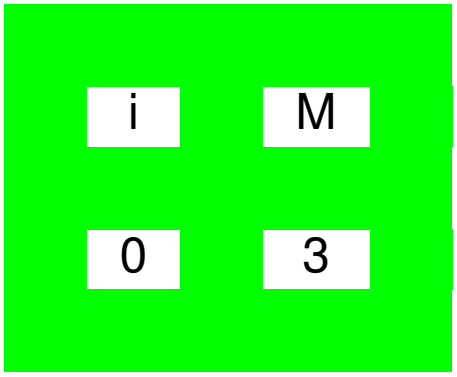
```
1   while (p->suivant !=p) {
2     for (i=0; i<M; i++) {
3       precedent=p;
4       p=p->suivant; }
5     precedent->suivant=p->suivant;
6     free (p);
7     p=precedent->suivant; }
8   return p->numero_joueur; }
```

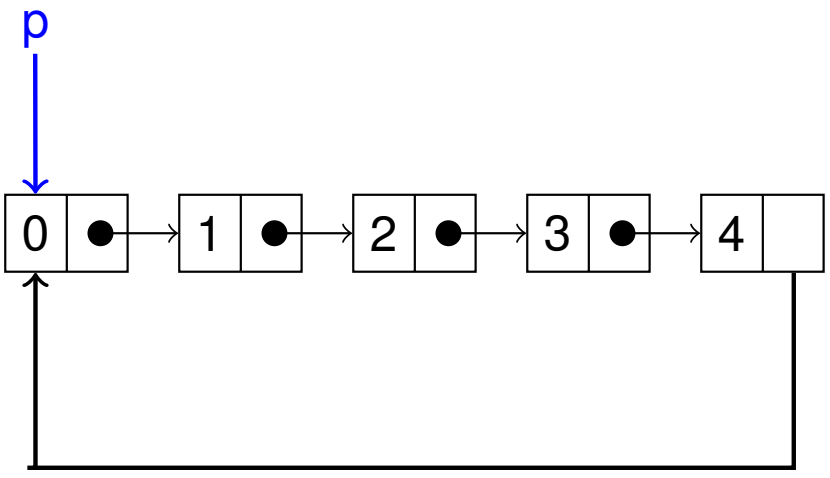
i	M
0	3



Déroulement de l'algorithme

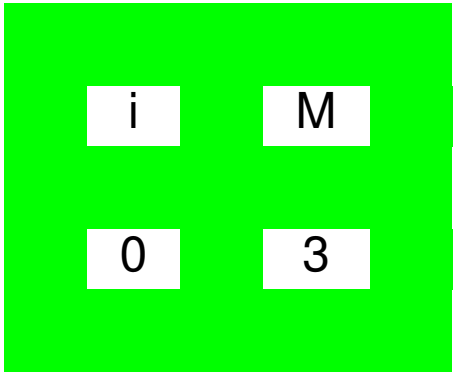
```
1  while (p->suivant !=p) {
   for (i=0; i<M; i++) {
2
3     precedent=p;
4     p=p->suivant; }
5   precedent->suivant=p->suivant;
6   free (p);
7   p=precedent->suivant; }
8  return p->numero_joueur; }
```



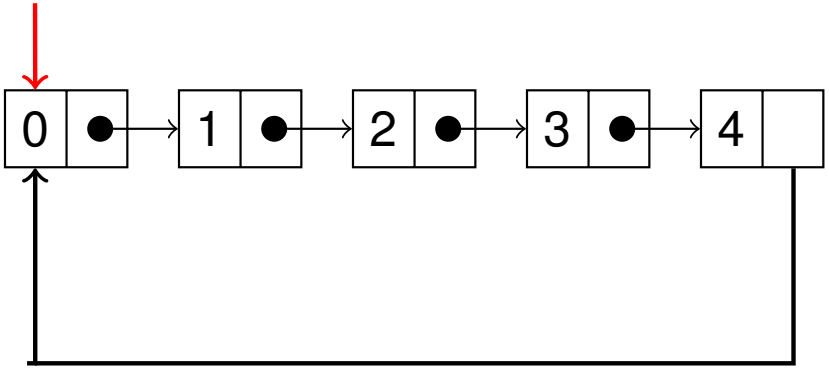


Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

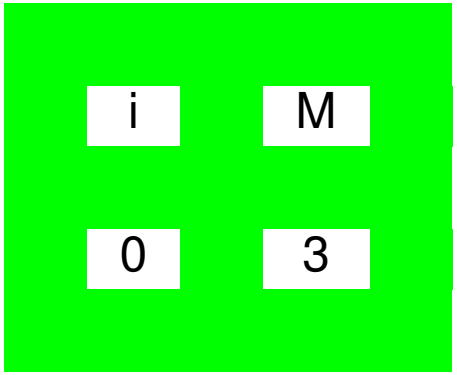


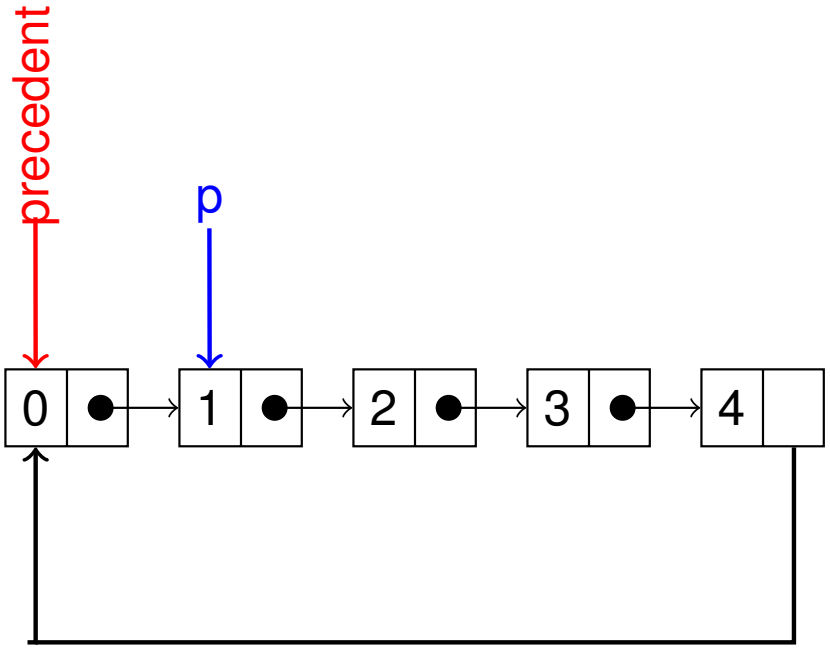
p, precedent



Déroulement de l'algorithme

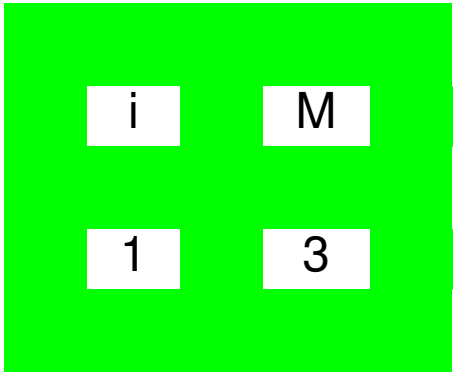
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
       p=p->suivant; }
4
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

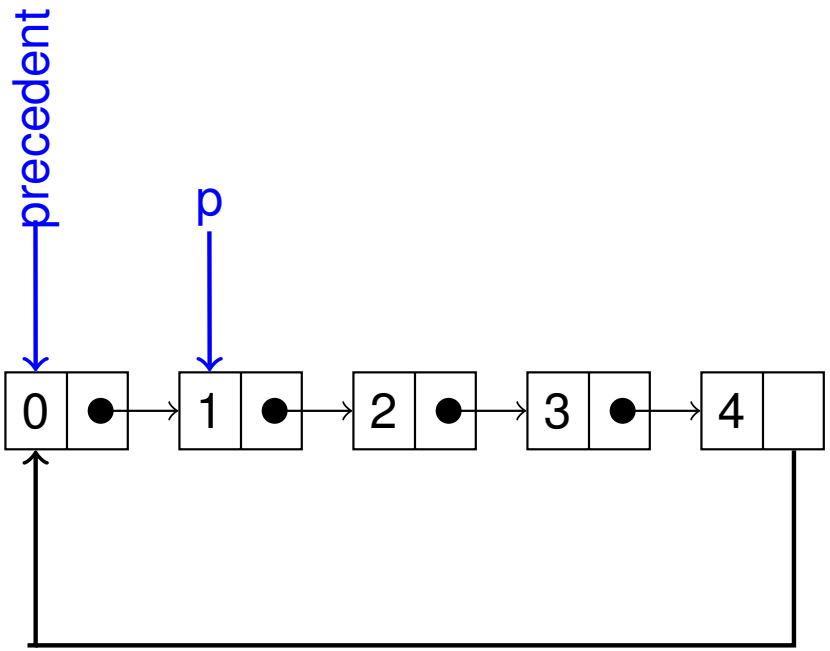




Déroulement de l'algorithme

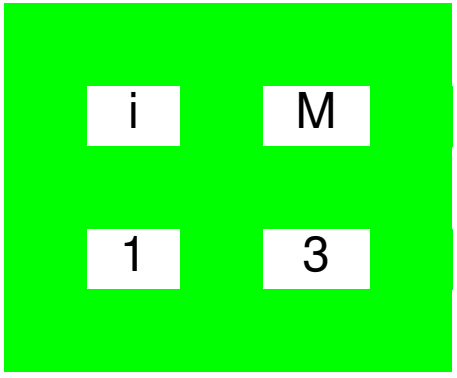
```
1  while (p->suivant !=p) {
2     for (i=0; i<M; i++) {
3       precedent=p;
4       p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```



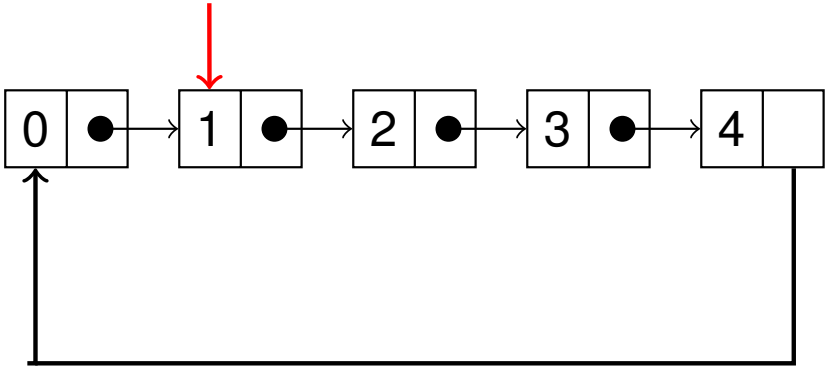


Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

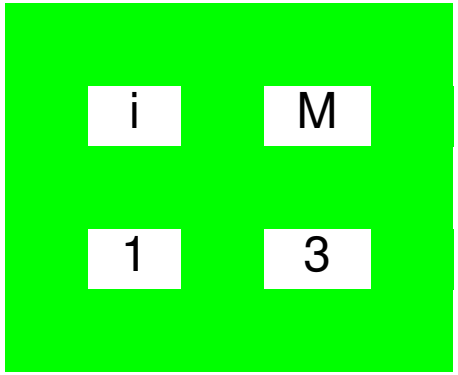


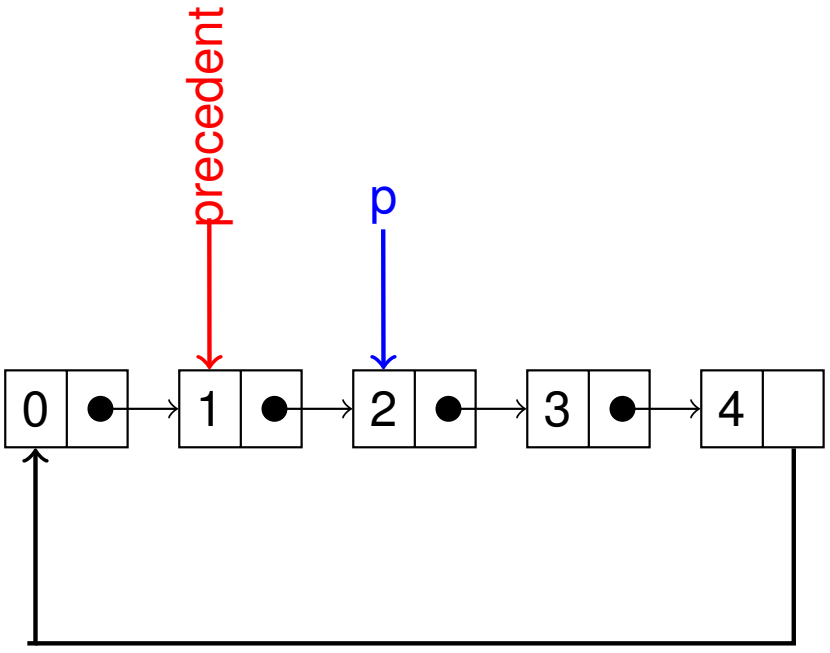
p, precedent



Déroulement de l'algorithme

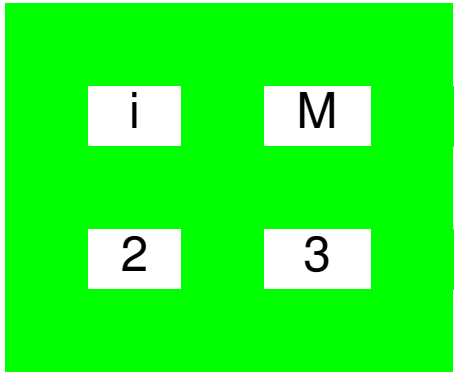
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
       p=p->suivant; }
4
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

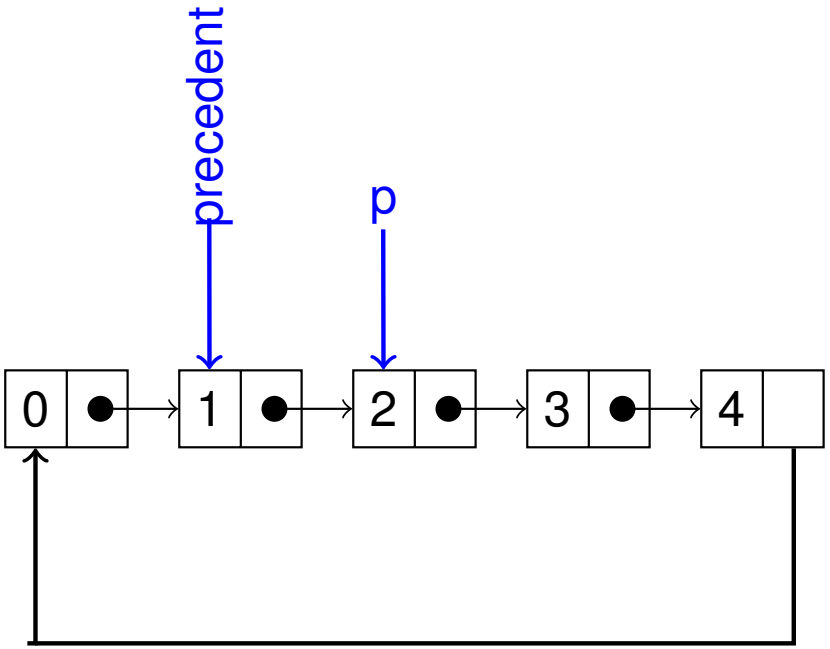




Déroulement de l'algorithme

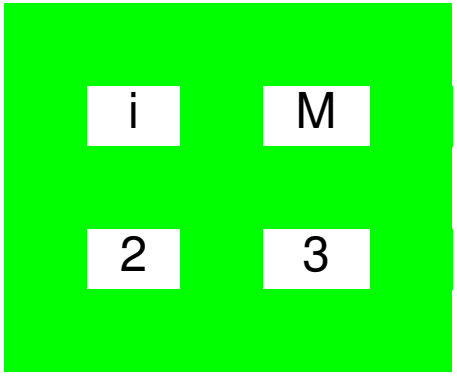
```
1  while (p->suivant !=p) {
   for (i=0; i<M; i++) {
2
3     precedent=p;
4     p=p->suivant; }
5   precedent->suivant=p->suivant;
6   free (p);
7   p=precedent->suivant; }
8  return p->numero_joueur; }
```

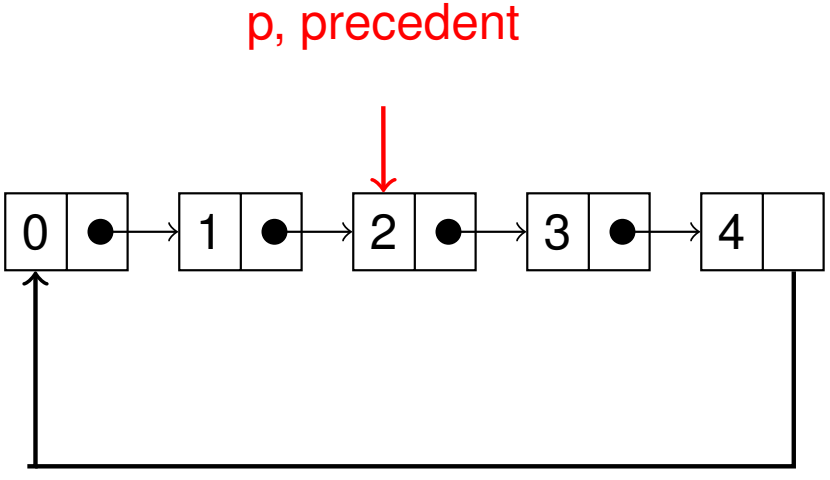




Déroulement de l'algorithme

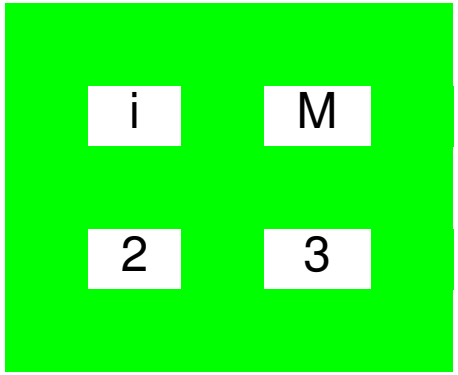
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

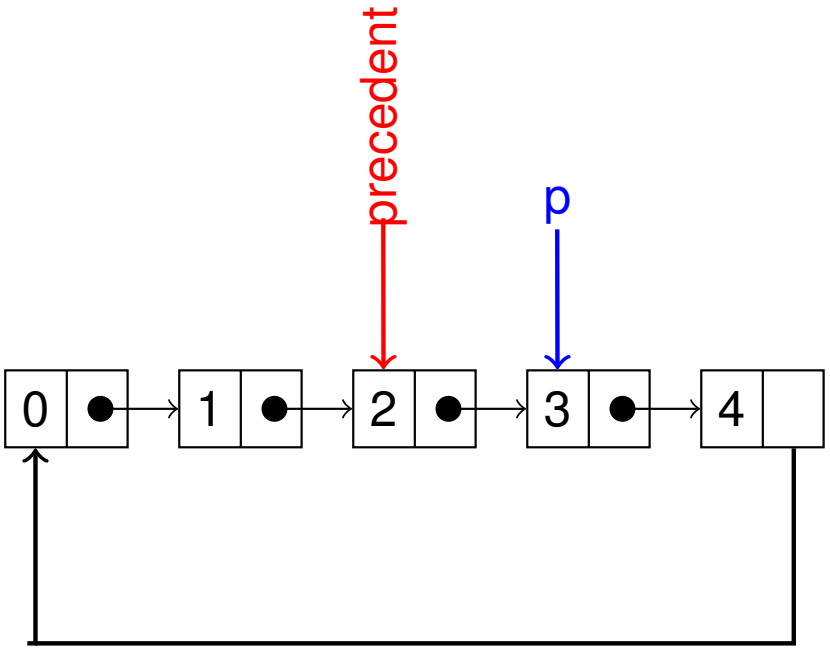




Déroulement de l'algorithme

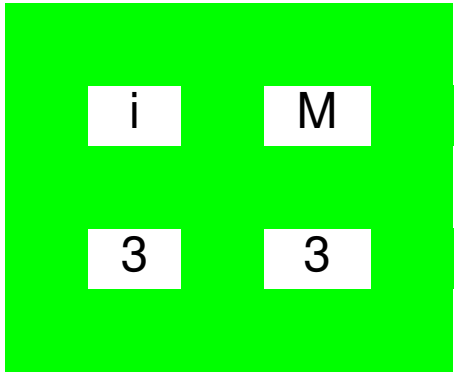
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
       p=p->suivant; }
4
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

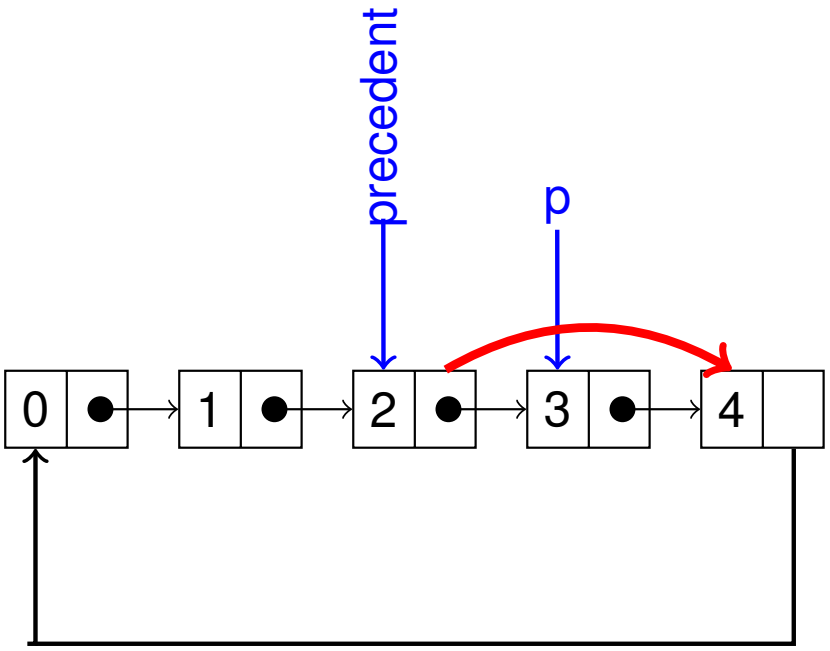




Déroulement de l'algorithme

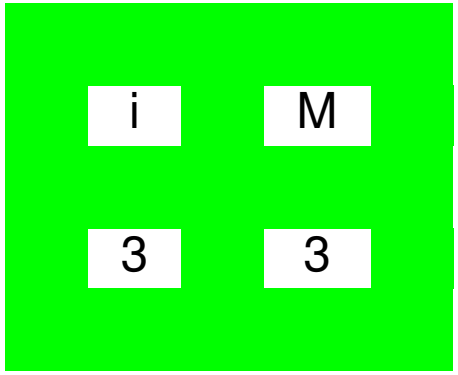
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

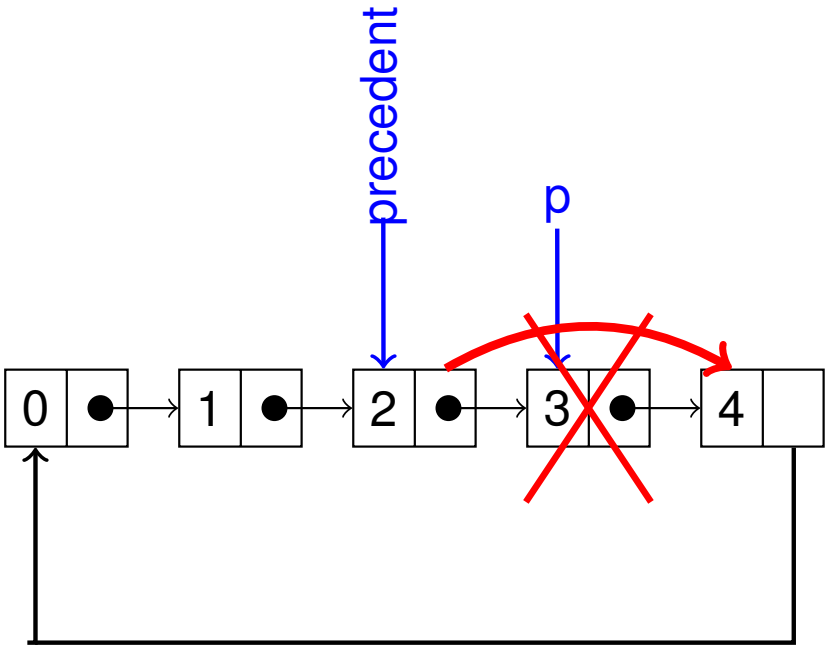




Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6     free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

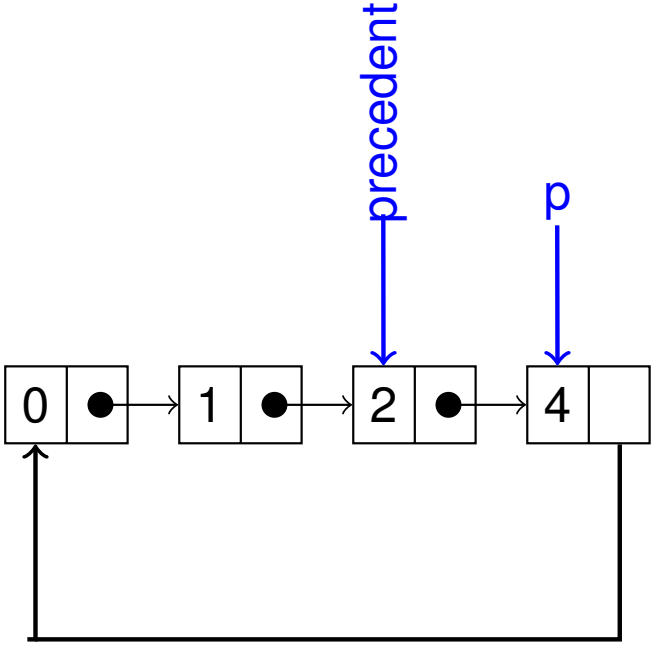




Déroulement de l'algorithme

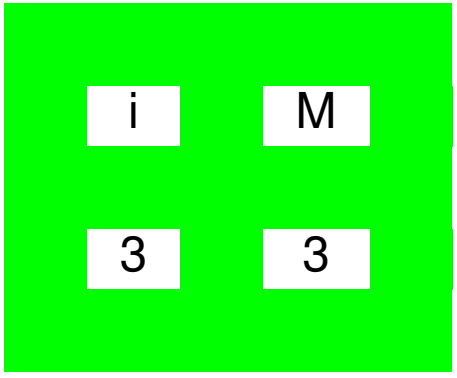
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7     p=precedent->suivant; }
8   return p->numero_joueur; }
```

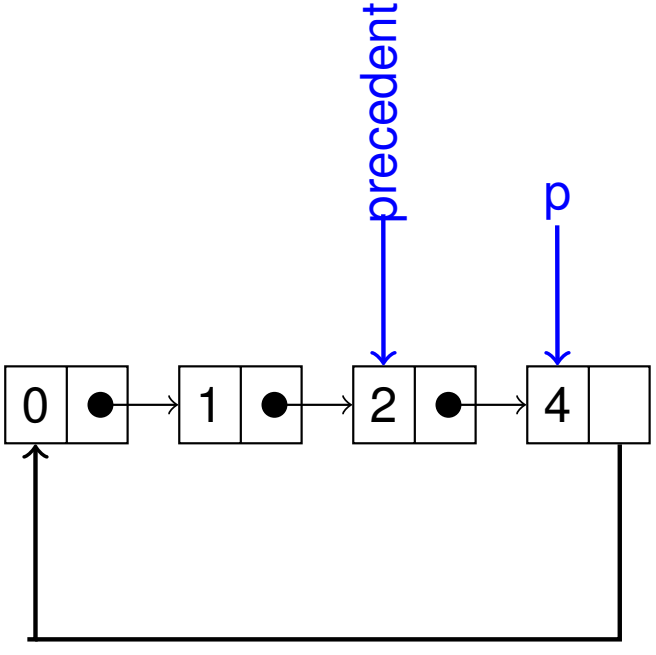
i	M
3	3



Déroulement de l'algorithme

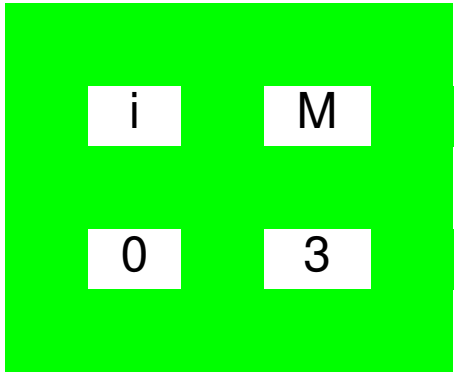
```
1   while (p->suivant !=p) {
2     for (i=0; i<M; i++) {
3       precedent=p;
4       p=p->suivant; }
5     precedent->suivant=p->suivant;
6     free (p);
7     p=precedent->suivant; }
8   return p->numero_joueur; }
```

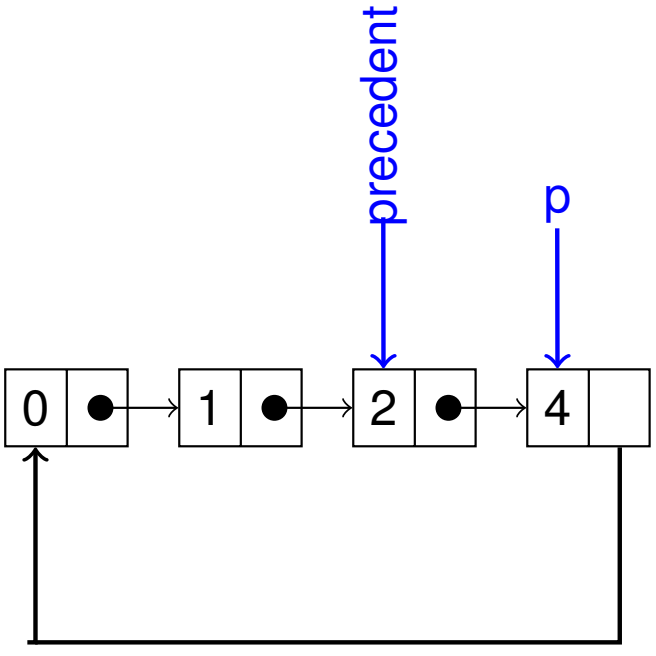




Déroulement de l'algorithme

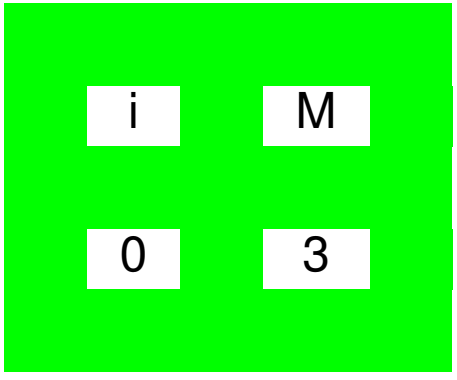
```
1  while (p->suivant !=p) {
2     for (i=0; i<M; i++) {
3       precedent=p;
4       p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

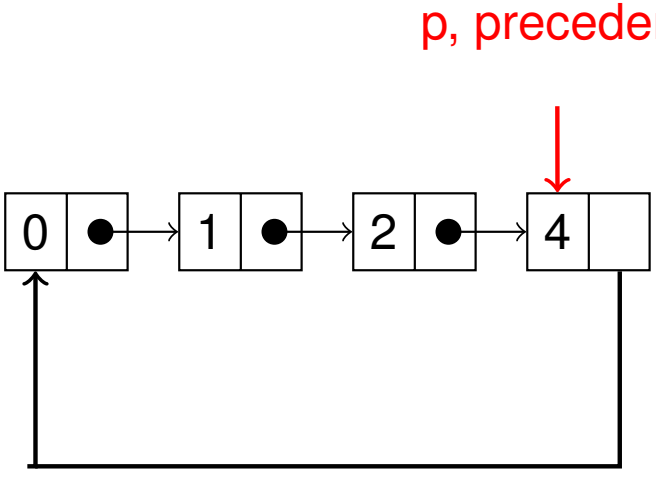




Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

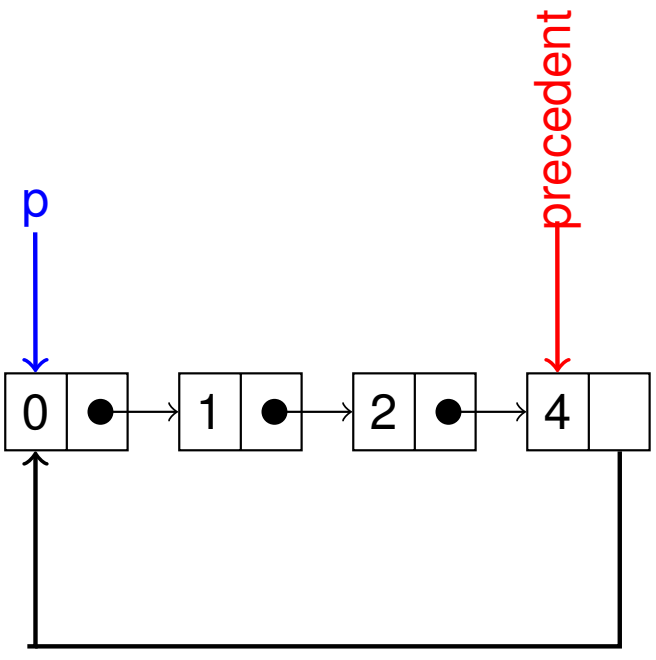




Déroulement de l'algorithme

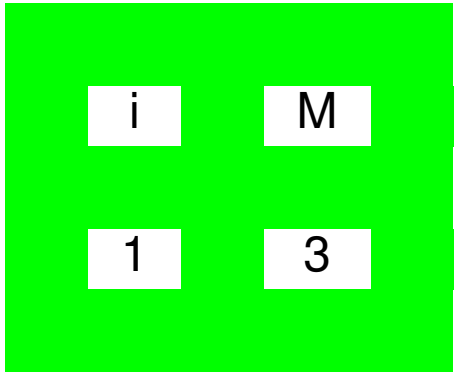
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
       p=p->suivant; }
4
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

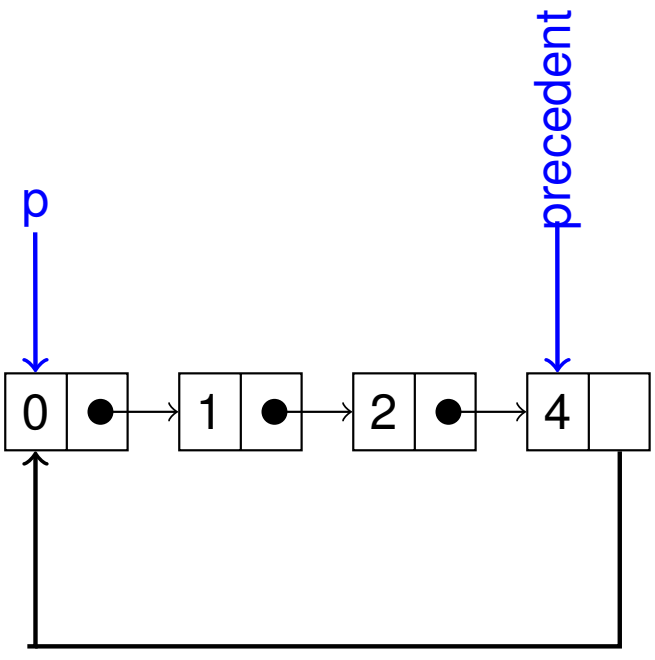
i	M
0	3



Déroulement de l'algorithme

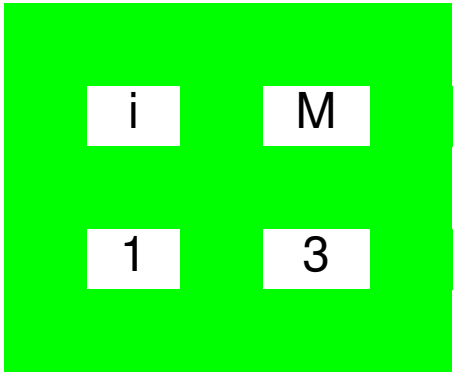
```
1  while (p->suivant !=p) {
2     for (i=0; i<M; i++) {
3       precedent=p;
4       p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```



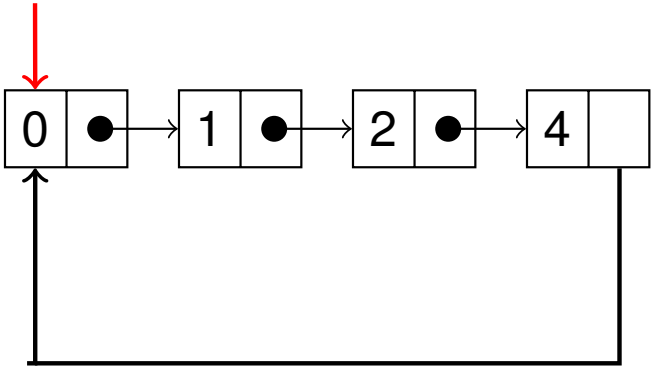


Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

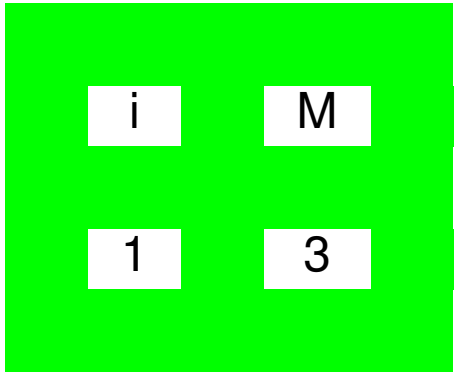


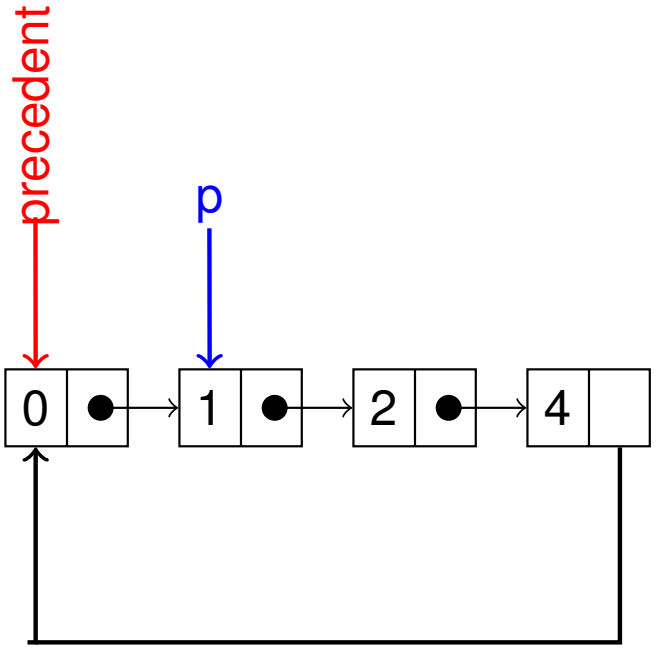
p, precedent



Déroulement de l'algorithme

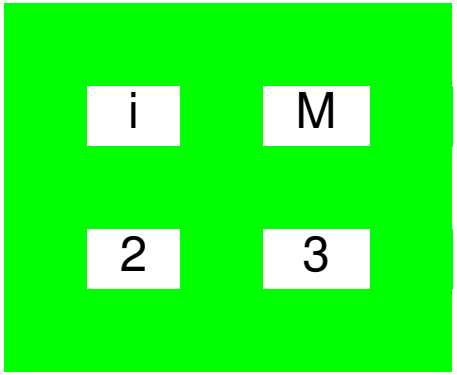
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
       p=p->suivant; }
4
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

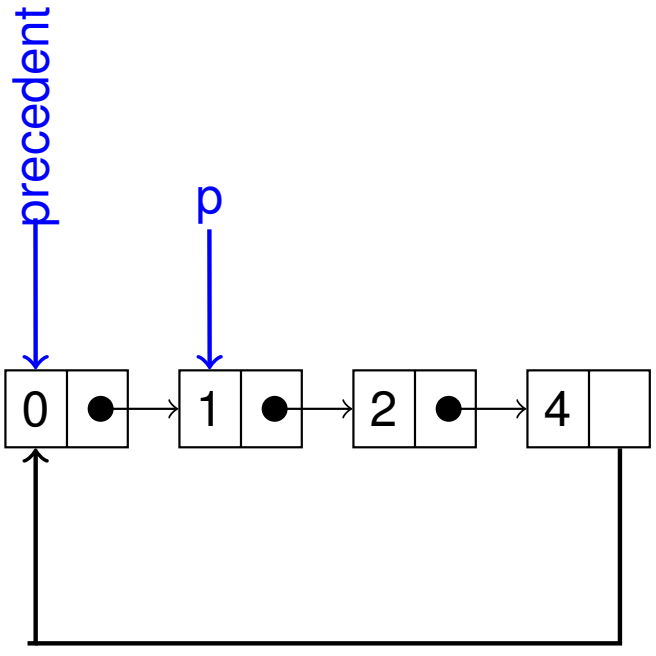




Déroulement de l'algorithme

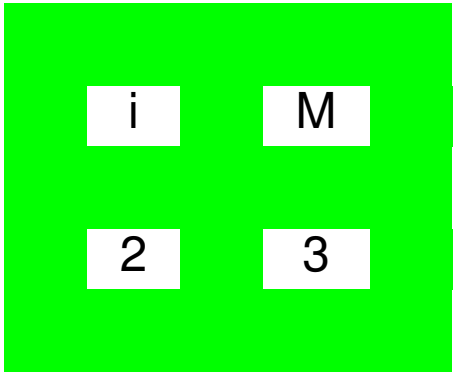
```
1  while (p->suivant !=p) {
2     for (i=0; i<M; i++) {
3       precedent=p;
4       p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```



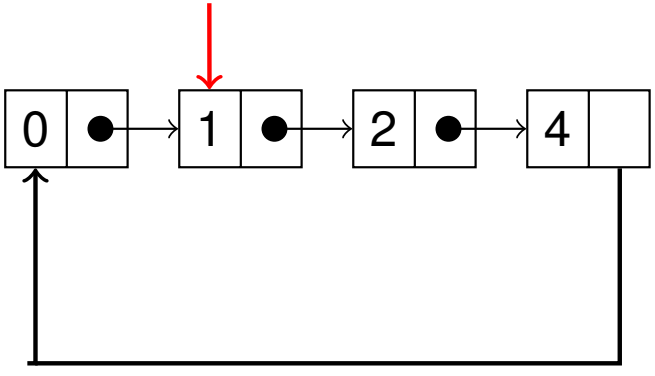


Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

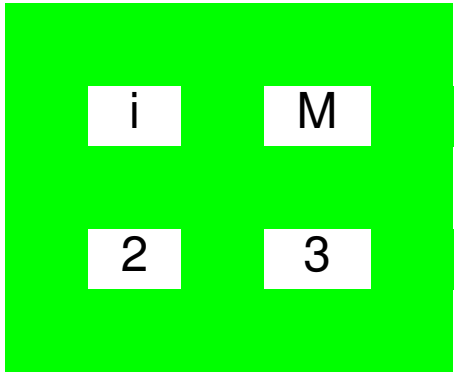


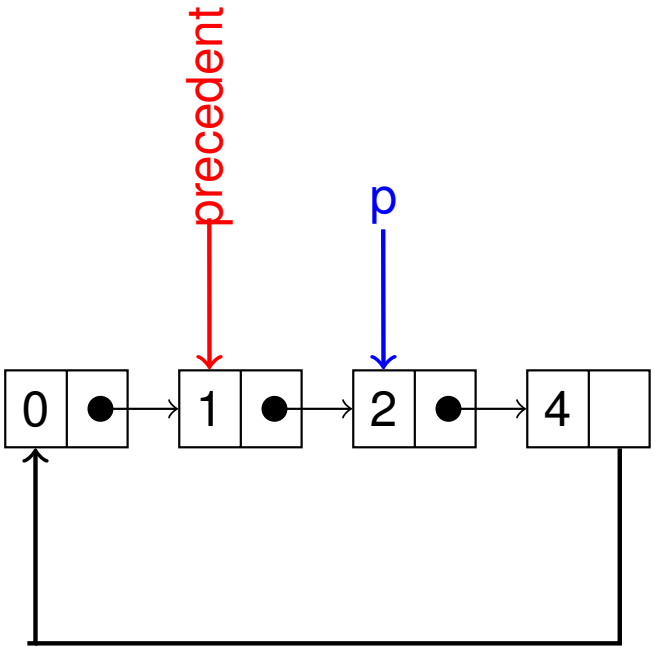
p, precedent



Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
       p=p->suivant; }
4
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

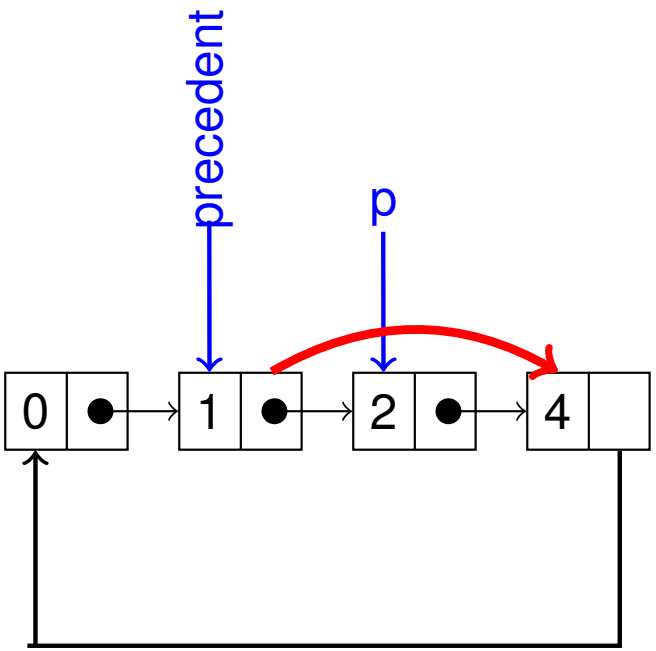




Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

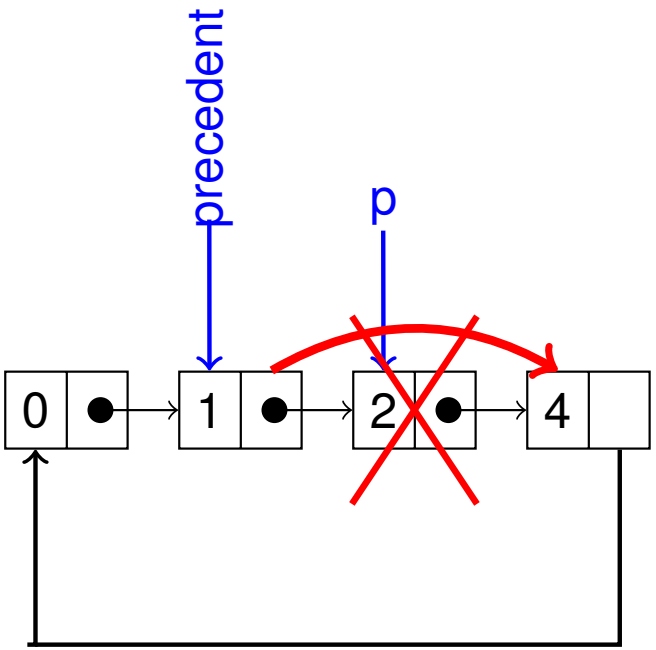
i	M
3	3



Déroulement de l'algorithme

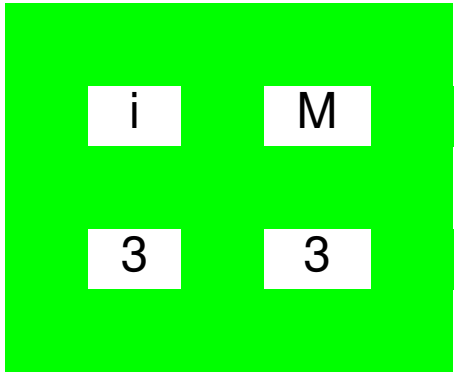
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6     free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

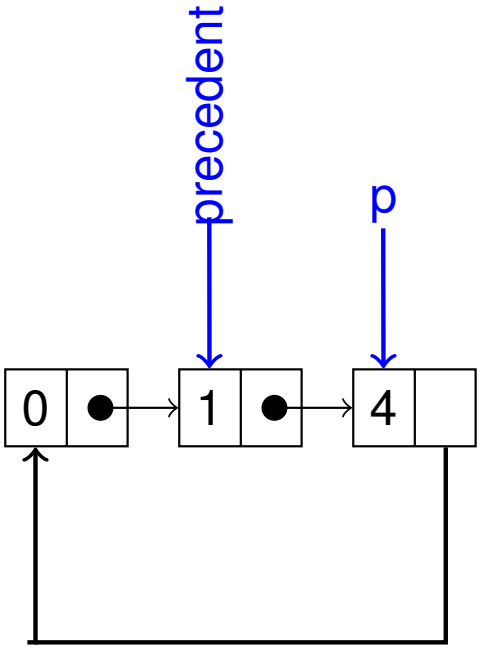
i	M
3	3



Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7     p=precedent->suivant; }
8   return p->numero_joueur; }
```

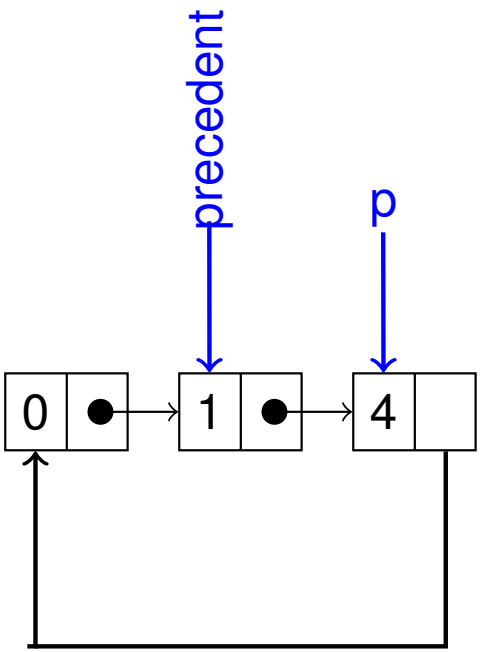




Déroulement de l'algorithme

```
1   while (p->suivant !=p) {
2     for (i=0; i<M; i++) {
3       precedent=p;
4       p=p->suivant; }
5     precedent->suivant=p->suivant;
6     free (p);
7     p=precedent->suivant; }
8   return p->numero_joueur; }
```

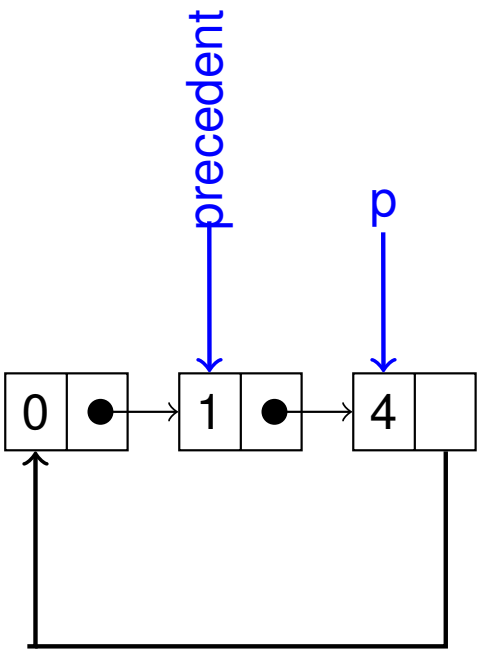
i	M
3	3



Déroulement de l'algorithme

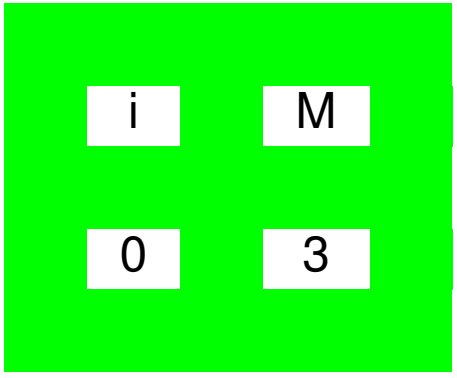
```
1  while (p->suivant !=p) {
2     for (i=0; i<M; i++) {
3       precedent=p;
4       p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

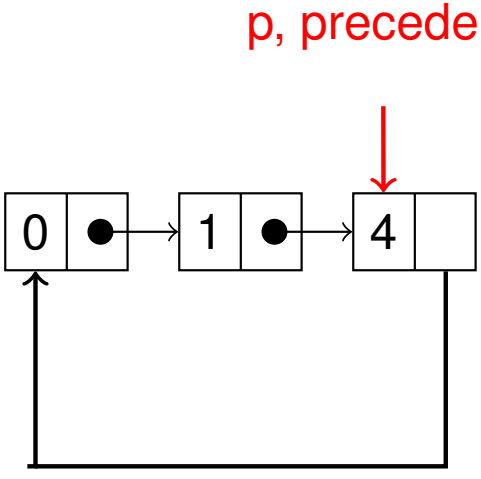
i	M
0	3



Déroulement de l'algorithme

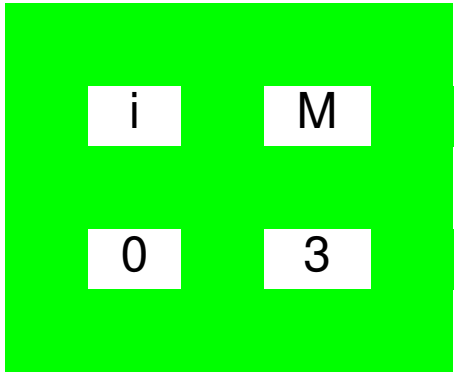
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

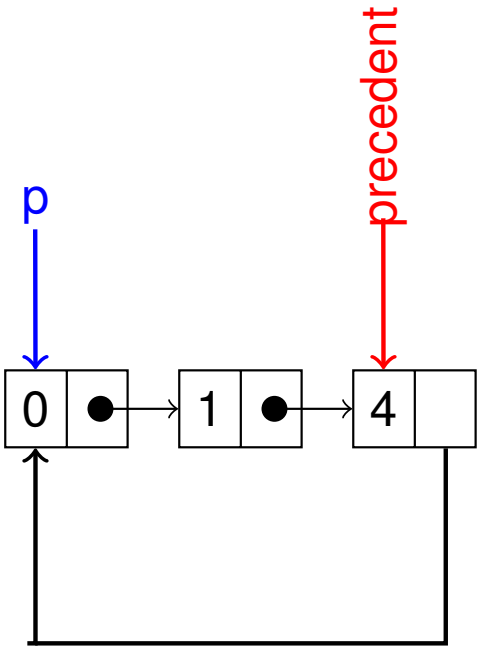




Déroulement de l'algorithme

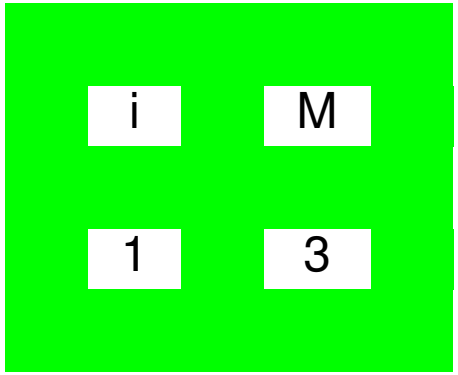
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
       p=p->suivant; }
4
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

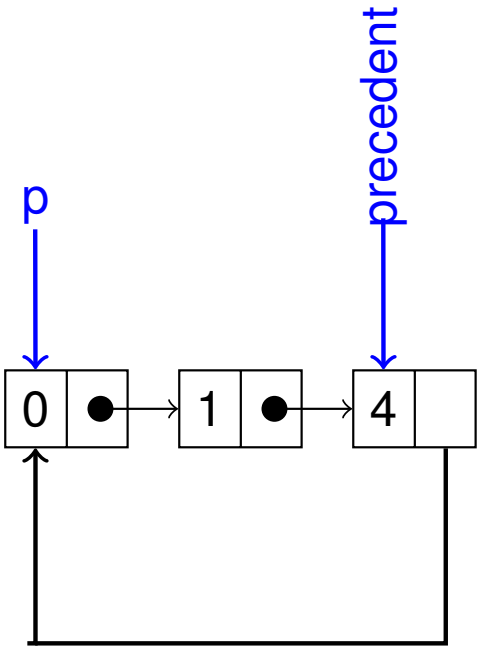




Déroulement de l'algorithme

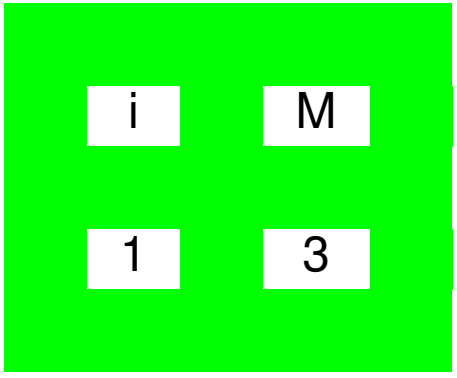
```
1  while (p->suivant !=p) {
2     for (i=0; i<M; i++) {
3       precedent=p;
4       p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```



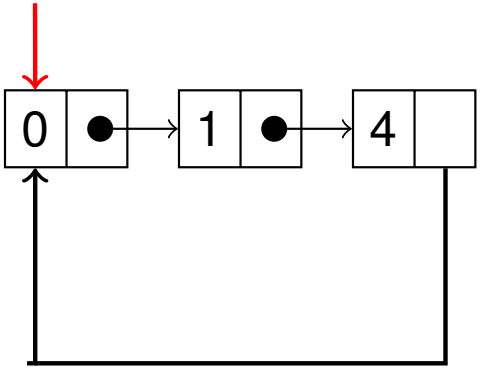


Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

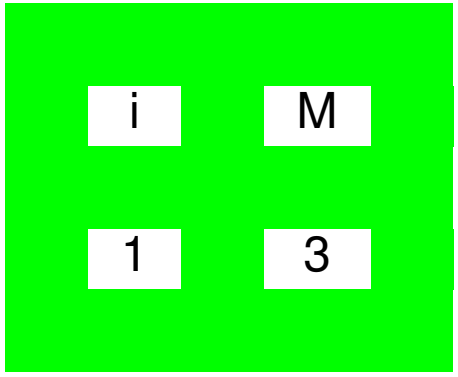


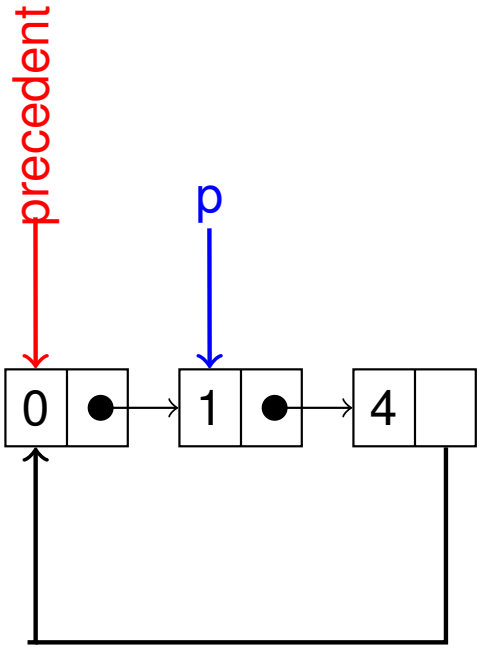
p, precedent



Déroulement de l'algorithme

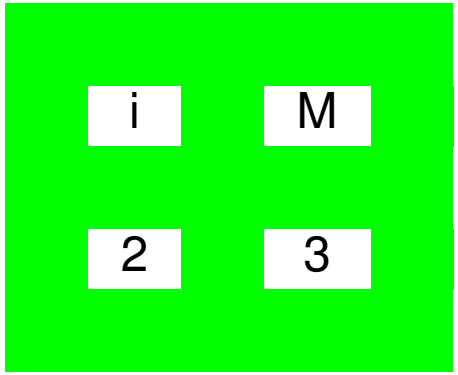
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
       p=p->suivant; }
4
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

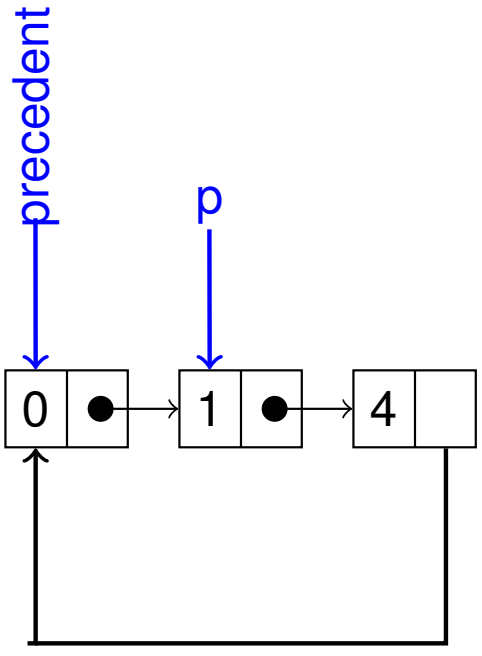




Déroulement de l'algorithme

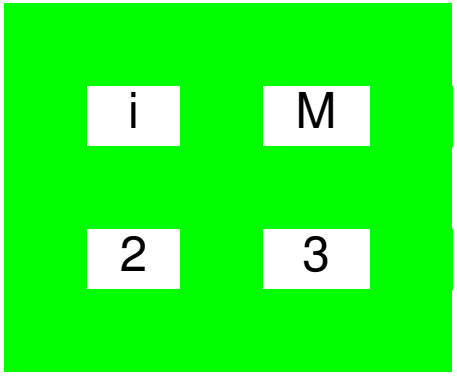
```
1  while (p->suivant !=p) {
   for (i=0; i<M; i++) {
2
3     precedent=p;
4     p=p->suivant; }
5   precedent->suivant=p->suivant;
6   free (p);
7   p=precedent->suivant; }
8  return p->numero_joueur; }
```



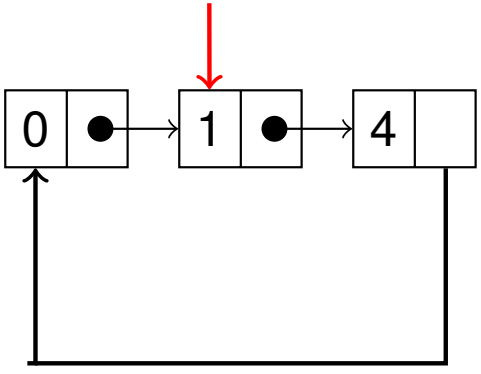


Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

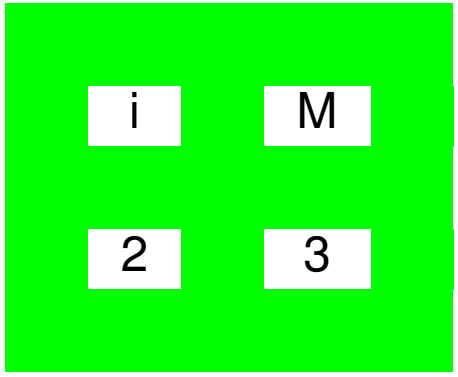


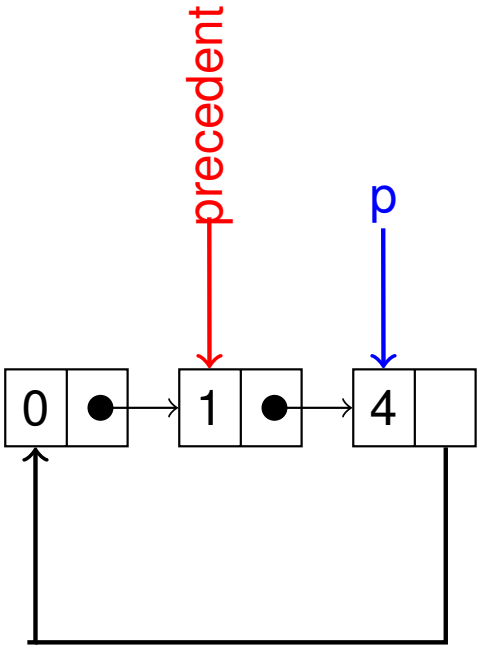
p, precedent



Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
       p=p->suivant; }
4
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

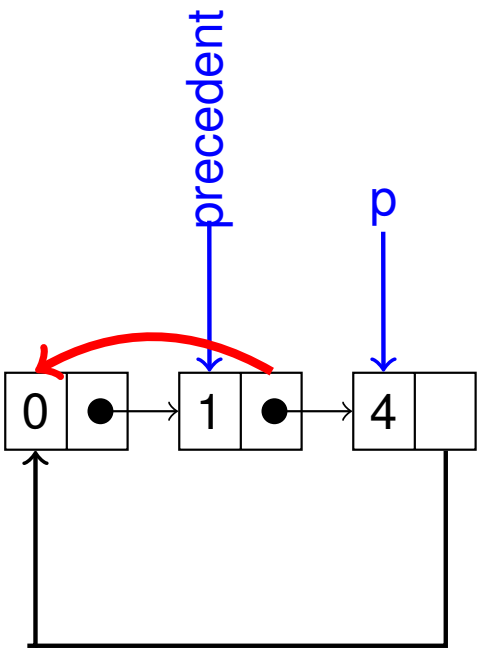




Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5     precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

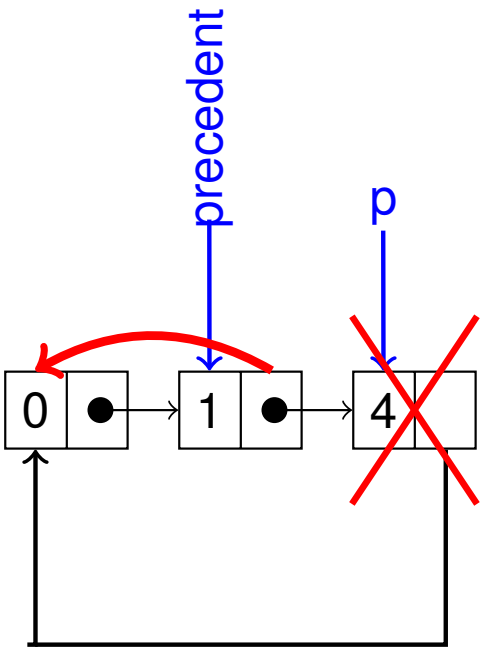
i	M
3	3



Déroulement de l'algorithme

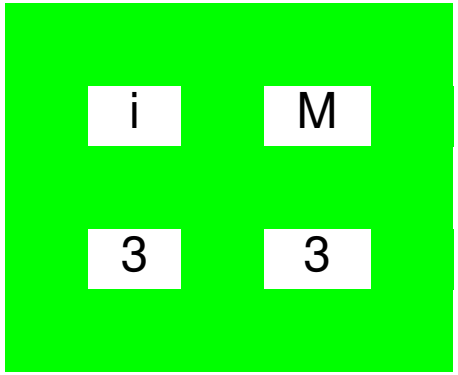
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6     free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

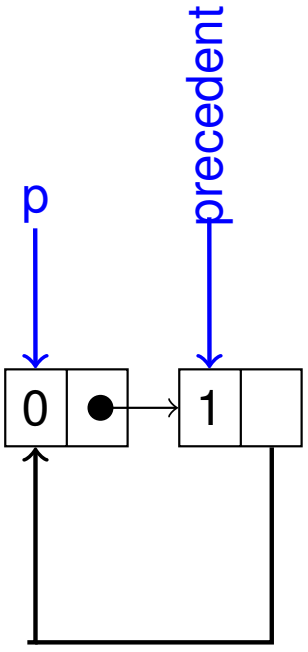
i	M
3	3



Déroulement de l'algorithme

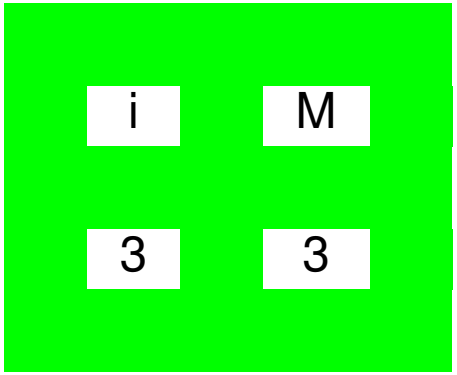
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7     p=precedent->suivant; }
8   return p->numero_joueur; }
```

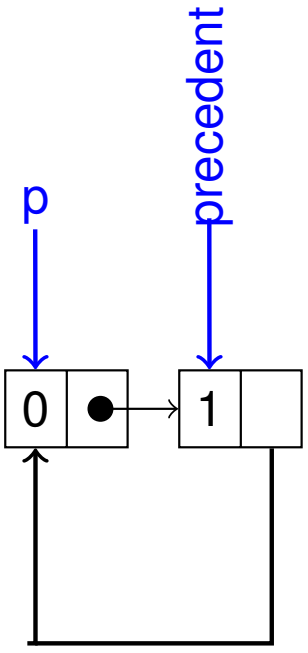




Déroulement de l'algorithme

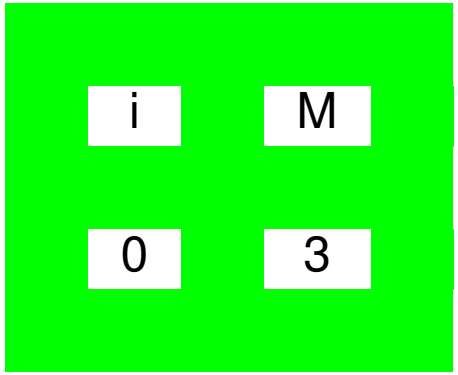
```
1   while (p->suivant !=p) {
2     for (i=0; i<M; i++) {
3       precedent=p;
4       p=p->suivant; }
5     precedent->suivant=p->suivant;
6     free (p);
7     p=precedent->suivant; }
8   return p->numero_joueur; }
```

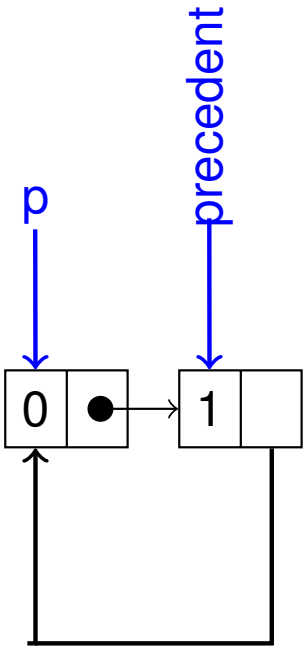




Déroulement de l'algorithme

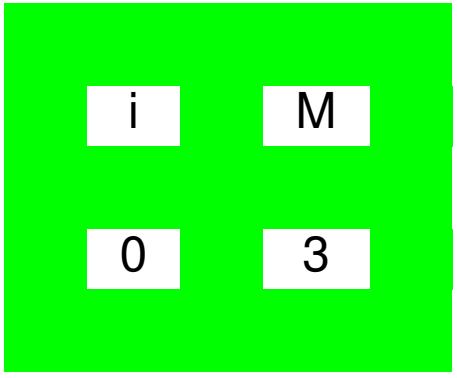
```
1  while (p->suivant !=p) {
   for (i=0; i<M; i++) {
2
3     precedent=p;
4     p=p->suivant; }
5   precedent->suivant=p->suivant;
6   free (p);
7   p=precedent->suivant; }
8  return p->numero_joueur; }
```



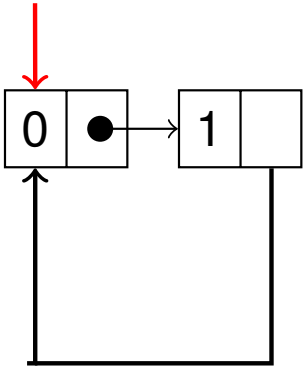


Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

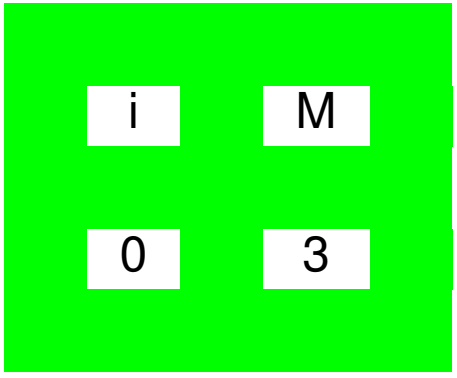


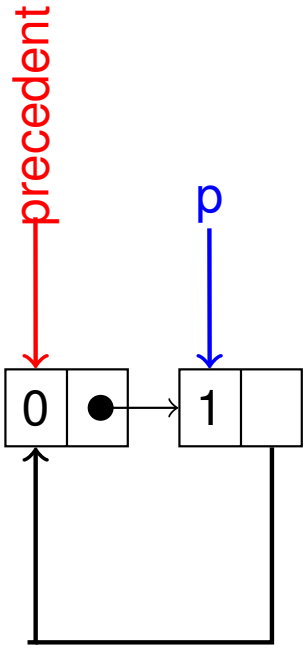
p, precedent



Déroulement de l'algorithme

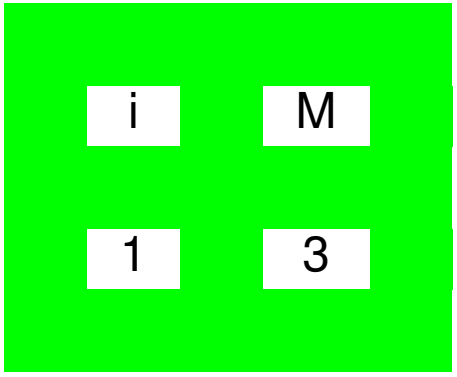
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
       p=p->suivant; }
4
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

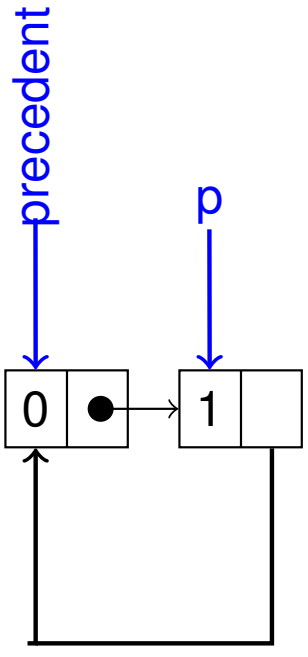




Déroulement de l'algorithme

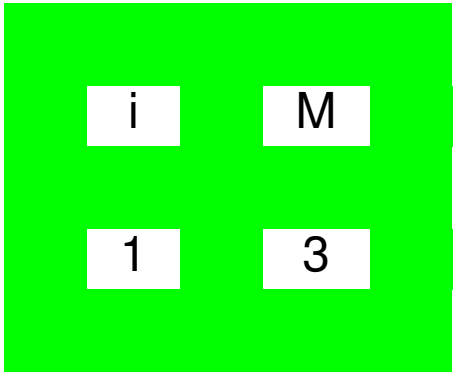
```
1  while (p->suivant !=p) {
2     for (i=0; i<M; i++) {
3       precedent=p;
4       p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```



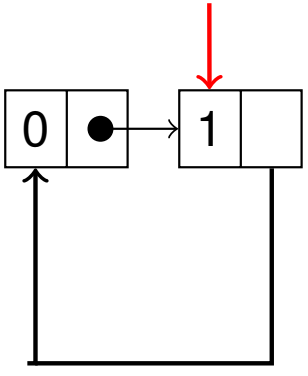


Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

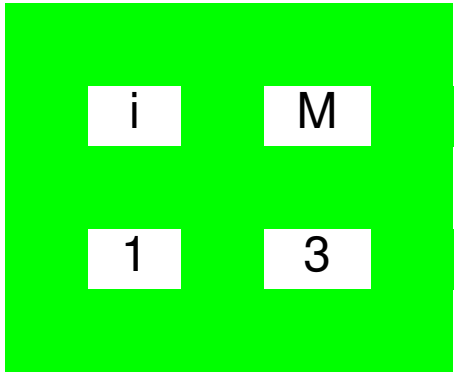


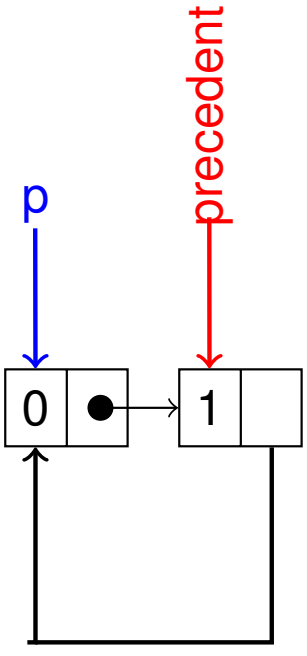
p, precedent



Déroulement de l'algorithme

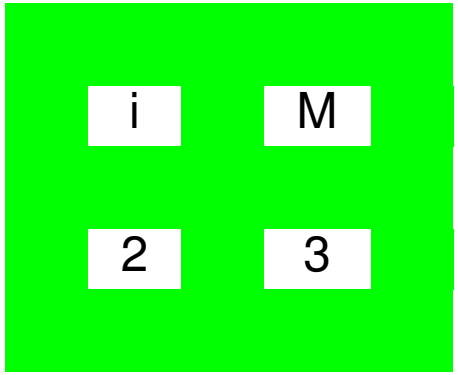
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
       p=p->suivant; }
4
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

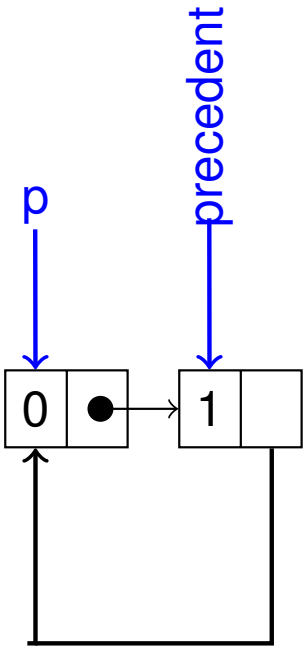




Déroulement de l'algorithme

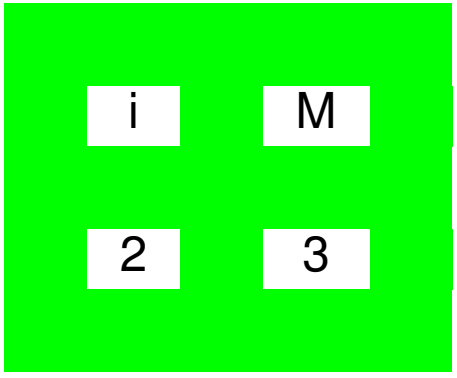
```
1  while (p->suivant !=p) {
   for (i=0; i<M; i++) {
2
3     precedent=p;
4     p=p->suivant; }
5   precedent->suivant=p->suivant;
6   free (p);
7   p=precedent->suivant; }
8  return p->numero_joueur; }
```



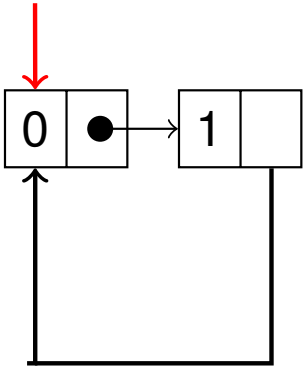


Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

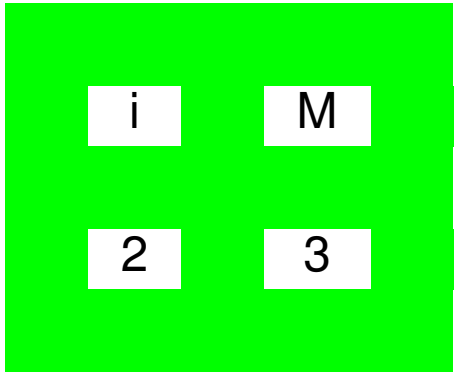


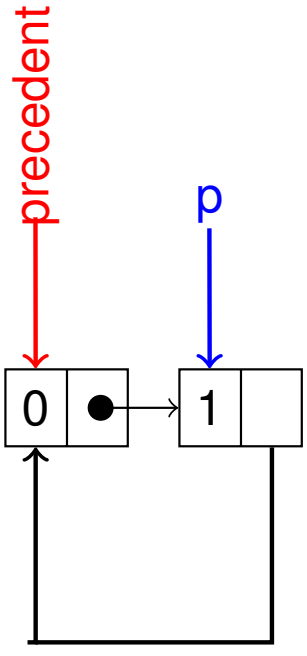
p, precedent



Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
       p=p->suivant; }
4
5    precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

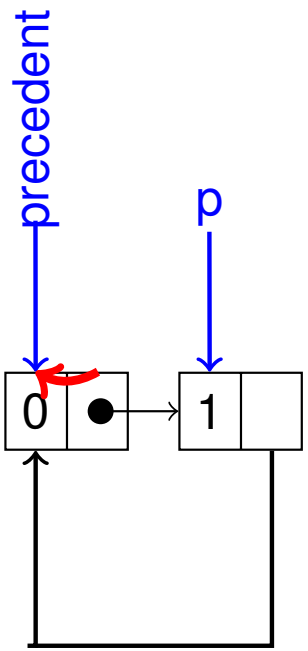




Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5     precedent->suivant=p->suivant;
6    free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

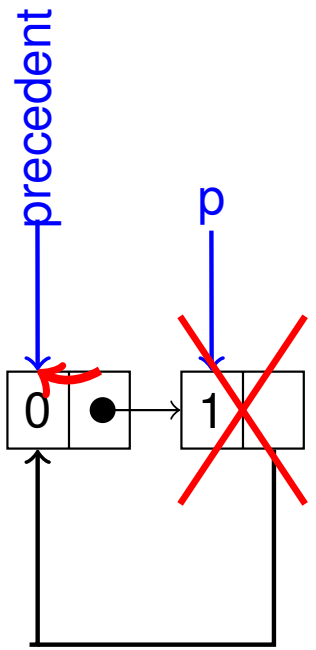
i	M
3	3



Déroulement de l'algorithme

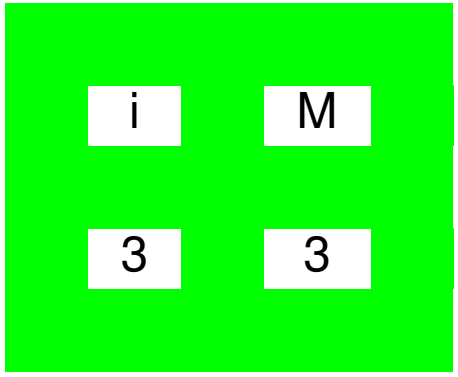
```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6     free (p);
7    p=precedent->suivant; }
8   return p->numero_joueur; }
```

i	M
3	3

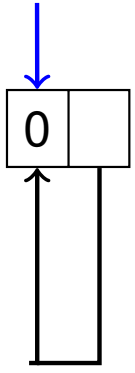


Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6    free (p);
7     p=precedent->suivant; }
8   return p->numero_joueur; }
```

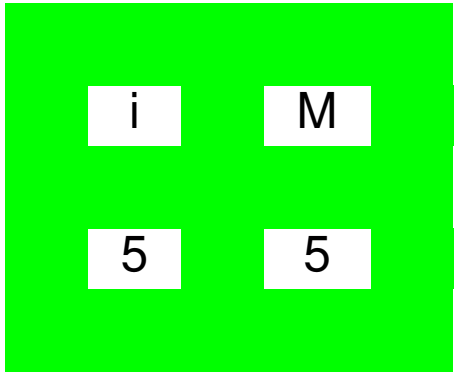


p, precedent



Déroulement de l'algorithme

```
1  while (p->suivant !=p) {
2    for (i=0; i<M; i++) {
3      precedent=p;
4      p=p->suivant; }
5    precedent->suivant=p->suivant;
6     free (p);
7    p=precedent->suivant; }
8     return p->numero_joueur; }
```



Le gagnant est :