Université de Namur

Faculté d'Informatique

www.info.fundp.ac.be

Introduction au langage C – 2e partie

Formation donnée dans le cadre de l'Olympiade belge d'informatique et des cours ouverts de carnaval

Xavier Devroey
Seweryn Dynerowicz
Nicolas Genon
Fabian Gilson

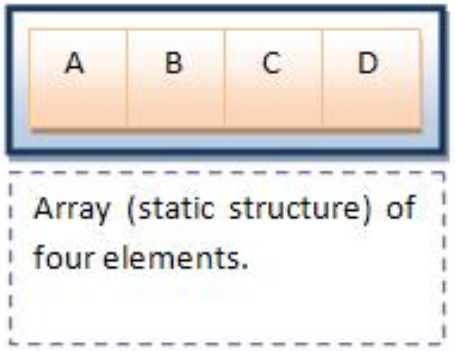
ven. 15 février 2013

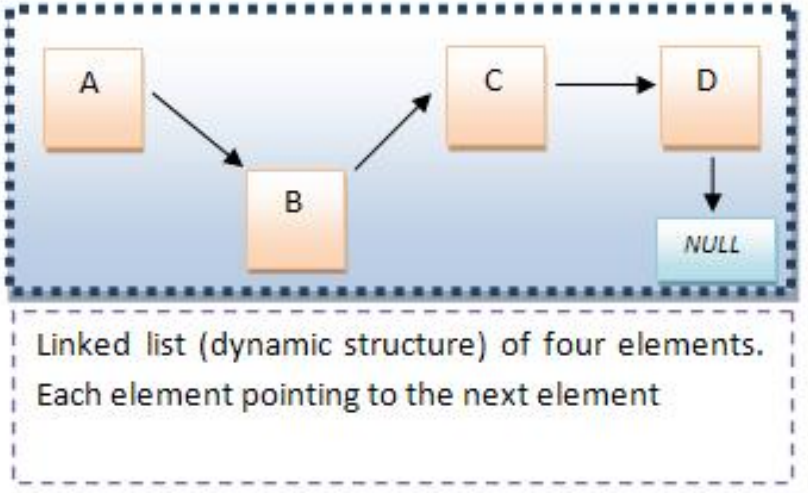
Concepts « avancés » de programmation en C

- Structures dynamiques
- Pointeurs
- Chaînes de caractères
- Précisions et rappels sur les I/O
- Utilisation des fichiers

Structures dynamiques - introduction

- Limitations des structures statiques
  - taille fixe, donc lourde à agrandir
  - utilisation d'espace mémoire parfois inutile
- Contraintes et risques des structures dynamiques
  - allocation et libération de la mémoire manuellement
  - déplacement et accès dans la structure plus lents
- Représentation simplifiée

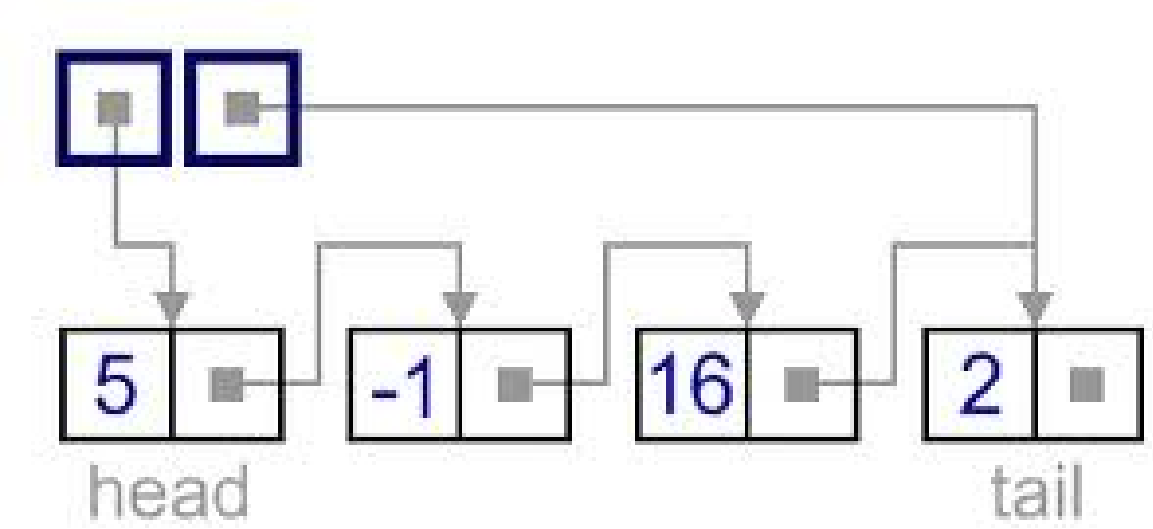




source : http://en.wikipedia.org/wiki/File:Static_and_dynamic_data_structures.svg

Structures dynamiques – liste chaînée

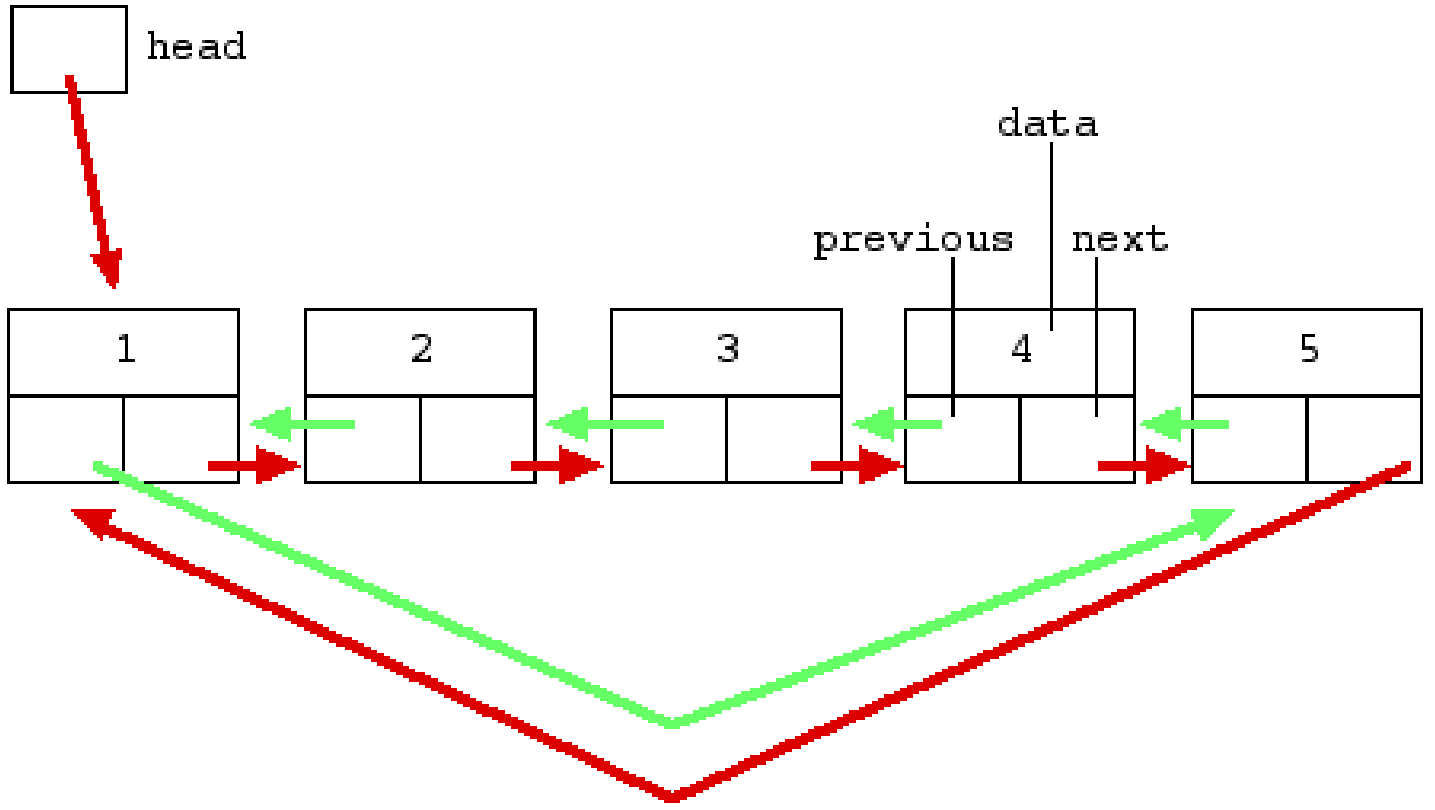
- La plus simple
  - une tête (`head`)
  - une queue (`tail`)
  - chaque valeur pointe vers son successeur (`next`)



source : http://www.algolist.net/Data_structures/Singly-linked_list/Internal_repr

Structures dynamiques – listes doublement chaînées

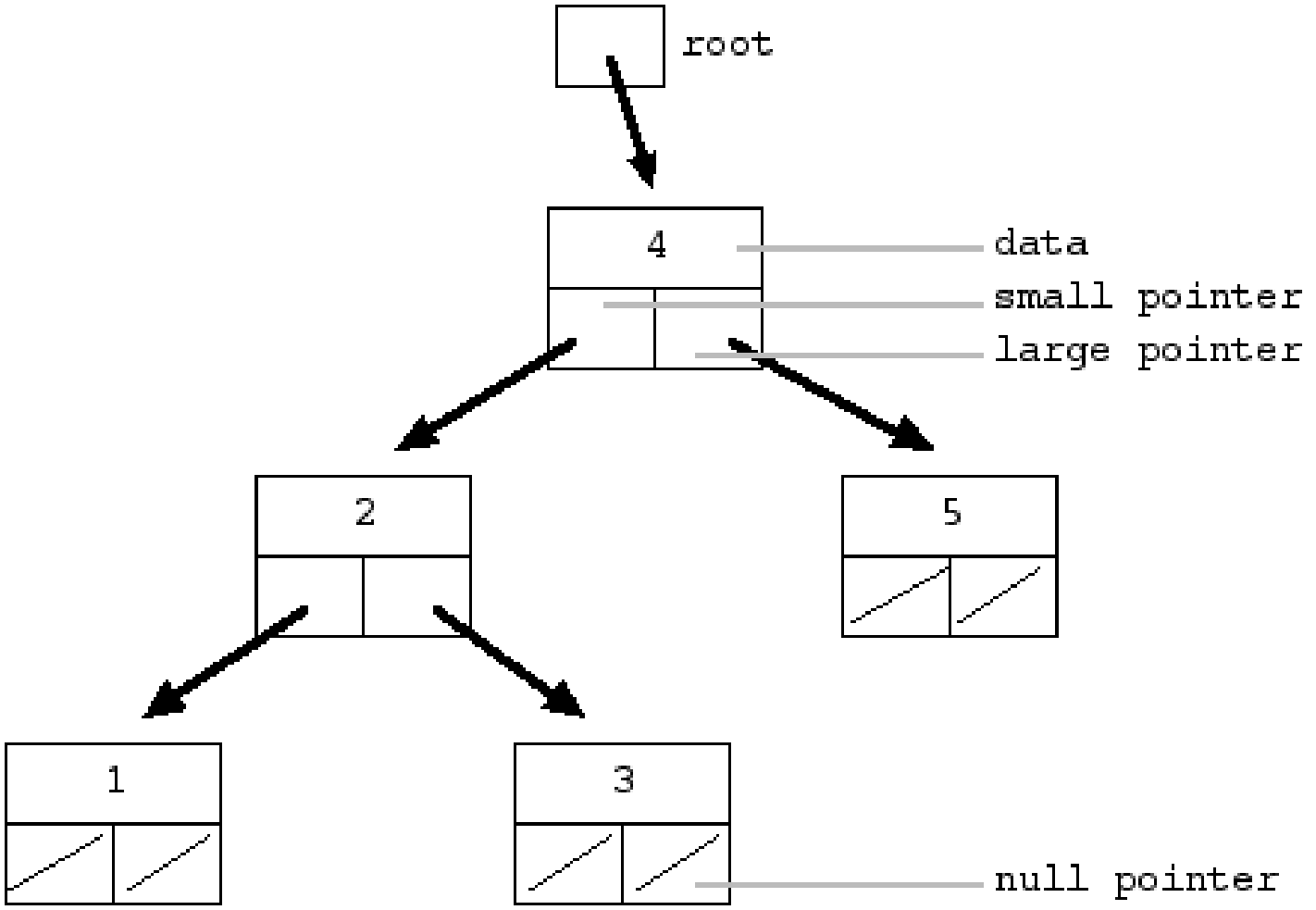
- Même principe qu'une liste chaînée, mais dans les deux sens
  - `next` pointe sur l'élément suivant
  - `previous` pointe sur l'élément précédent



source : http://cslibrary.stanford.edu/109/TreeListRecursion.html

Structures dynamiques - arbres

- Structure récursive
- Est habituellement binaire, mais pas seulement



source : http://cslibrary.stanford.edu/110/BinaryTrees.html

Pointeurs

- En C, deux opérateurs *unaires* permettent de manipuler des pointeurs
  - `&a` obtenir l'adresse de la variable `a`
  - `*a` obtenir la valeur à l'adresse de la variable `a`

- Déclaration

```
int variable;
int *pointeur;
```

- Exemple

```
pointeur = &variable;
printf("la valeur du pointeur est %p\n", pointeur);
/* imprime l'adresse mémoire */
printf("la valeur à cette adresse est %d\n", *pointeur);
/* imprime la valeur */
```

Pointeurs et tableaux

- Les éléments d'un tableau occupent toujours une mémoire contiguë
- On peut incrémenter/décrémenter un pointeur
- Exemple, sur une machine 32bit avec pointeur sur `int`
  - `ptr++` faire pointer sur les 4 prochains octets dans la mémoire
  - `ptr--` faire pointer sur les 4 octets précédents
- Il est donc possible de se déplacer dans un tableau avec un pointeur
  - attention à la taille des données (type)
  - attention à ne pas se « balader » n'importe où

Pointeurs et tableaux – démonstration (1/2)

```c
#include <stdio.h>

int main()
{
  int my_variable = 6, other_variable = 10;
  int *my_pointer;

  printf("the address of my_variable is    : %p\n", &my_variable);
  printf("the address of other_variable is : %p\n", &other_variable);

  my_pointer = &my_variable;

  printf("\nafter \"my_pointer = &my_variable\":\n");
  printf("\tthe value of my_pointer is %p\n", my_pointer);
  printf("\tthe value at that address is %d\n", *my_pointer);

  my_pointer = &other_variable;

  printf("\nafter \"my_pointer = &other_variable\":\n");
  printf("\tthe value of my_pointer is %p\n", my_pointer);
  printf("\tthe value at that address is %d\n", *my_pointer);

  return 0;
}
```

Pointeurs et tableaux – démonstration (2/2)

```c
#include <stdio.h>

int main() {
  int *ptr;
  int arrayInts[10] = {1,2,3,4,5,6,7,8,9,10};

  ptr = arrayInts; /* ptr = &arrayInts[0]; is also fine */

  printf("The pointer is pointing to the first ");
  printf("array element, which is %d.\n", *ptr);
  printf("Let's increment it.....\n");

  ptr++;

  printf("Now it should point to the next element,");
  printf(" which is %d.\n", *ptr);
  printf("But suppose we point to the 3rd and 4th: %d %d.\n", *(ptr+1),*(ptr+2));

  ptr+=2;

  printf("Now skip the next 4 to point to the 8th: %d.\n", *(ptr+=4));

  ptr--;

  printf("Did I miss out my lucky number %d?!\n", *(ptr++));
  printf("Back to the 8th it is then..... %d.\n", *ptr);

  return 0;
}
```

Les chaînes de caractères

- Chaîne de caractère en C
  - tableau de `char`
  - pointeur vers une zone mémoire contenant des caractères ASCII
- Représentée par 0 caractère ou plus terminé par *NULL*
  - attention à laisser une place pour le NULL
  - déclarée avec des « " » autour de la chaîne
- Déclaration

```
char *string_1 = "Hello";
char string_2[] = "Hello";
char string_3[6] = "Hello";
```

- Attention aux espaces lorsqu'on lit avec `scanf`

  => `gets`, mais dangeureux...

Les chaînes de caractères - démonstration

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
  char array[50];
  char *ptr;

  printf("Now enter another string less than 50");
  printf(" characters with spaces: \n");
  gets(array);

  printf("\nYou entered: ");
  puts(array);

  printf("\nTry entering a string less than 50");
  printf(" characters, with spaces: \n");

  ptr = (char *) malloc(sizeof(array));

  scanf("%s", ptr);

  printf("\nYou entered: %s\n", ptr);

  return 0;
}
```

Les chaînes de caractères – fonctions utiles

- Copier deux strings

```
char *strcpy (char *dest, char *src)
```

- Copier les n premiers caractères

```
char *strncpy(char *string1, char *string2, int n)
```

- Comparer deux strings

```
int strcmp(char *string1, char *string2)
```

- Comparer les n premiers caractères

```
int strncmp(char *string1, char *string2, int n)
```

- Déterminer la longueur d'une string

```
int strlen(char *string)
```

- Concatener deux strings

```
char *strcat(char *dest, const char *src)
```

- Segmenter une string suivant un délimiteur

```
char *strtok(char *s, const char *delim)
```

Précisions et rappels sur les I/O

- *Spécificateurs* pour la lecture ou l'écriture de variables
  - `%d` `int`
  - `%f` `float` (ou `%x.yf` avec *x* et *y* des entiers)
  - `%c` `char`
  - `%s` `string`

- Lecture à l'entrée standard (clavier)

```
int x; scanf("%d", &x);
```

- Ecriture à la sortie standard (console)

```
char c; printf("un message %c\n", c);
```

- Alternatives

  `gets` peut lire une string avec des *espaces*

  `puts` ne peut écrire qu'une variable à la fois

Utilisation des fichiers (1/4)

- Pointeur vers un fichier

```
FILE *fp;
```

- Ouvrir un fichier

```
FILE *fopen(const char *filename, const char *mode);
```

  - mode d'ouverture
    - `r` ouvre en lecture
    - `w` ouvre en écriture (fichier peut ne pas exister)
    - `a` ouvre en écriture à la fin du fichier (fichier peut ne pas exister)
  - options
    - `+` mode lecture/écriture
    - `b` mode binaire

- Fermer un fichier

```
int fclose(FILE *a_file);
```

Utilisation des fichiers (2/4)

- Écrire dans un fichier

```
int fprintf(...);
```

- Lire dans un fichier

```
int fscanf(...);
```

- Fermer un fichier

```
int fclose(FILE *a_file);
```

- Exemple en écriture

```
void main()
{
   FILE *fp;
   fp = fopen("/tmp/test.txt", "w");
   fprintf(fp, "This is testing...\n");
   fclose(fp);
}
```

Utilisation des fichiers (3/4)

- Exemple en lecture

```
void main()
{
   FILE *fp;
   char buffer[20];

   fp = fopen("/tmp/test.txt", "r");
   fscanf(fp, "%s", buffer);
   printf("Read Buffer: %s\n", %buffer );
   fclose(fp;);
}
```

Utilisation des fichiers (4/4)

- Lecture caractère par caractère

```
int fgetc (FILE *fp);
```

- Écriture caractère par caractère

```
int fputc( int c, FILE *fp );
```

- Fichier binaire
  - lecture

```
size_t fread(void *ptr, size_t size_of_elements,
size_t number_of_elements, FILE *a_file);
```

  - écriture

```
size_t fwrite(const void *ptr, size_t
size_of_elements, size_t number_of_elements, FILE
*a_file);
```

Exercices (1/3)

- ***Ex 1*** – Addition des éléments d'un tableau dynamique
  - lire en entrée une liste de valeurs entières
  - remplir un tableau
  - calculer la somme des éléments du tableau

- ***Ex 2*** - Coda*ge RLE (run lenght encoding - compression naïve)*
  - adapter le programme afin de lire le contenu d'un fichier
  - le compresser et l'écrire dans un autre

- ***Ex 3.a*** – Compter les mots dans un fichier
  - parcourir un fichier en entrée
  - compter le nombre de mot dans ce fichier

- ***Ex 3.b*** – Compter les mots différents dans ce fichier
  - parcourir un fichier en entrée
  - compter le nombre de mots différents uniquement

Exercices (2/3) - `diff`

- ***Ex 4*** – Ecrire l'équivalent de la commande bash `diff`
  - prendre deux fichiers en entrée
  - les comparer ligne par ligne
  - produire un fichier reprenant les différences
  - indiquer pour chaque différence la ligne et le type de modification
    - a : ajout, d : suppression, c : changement
  - « > » ligne à ajouter
  - « < » ligne à supprimer
  - cfr. exemple page suivante

Exercices (3/3) – `diff` exemple

```c
#include <stdio.h>
int main () {
    int a[] = {2,4,34,23,-2,0};
    int resultat = 0;
    int i = 0;
    for (i = 0; i < sizeof(a)/sizeof(int); i++) {
        resultat = resultat + a[i];
    }
    printf("taille d'un int %d\n", sizeof(int));
    printf("resultat = %d\n", resultat);
    return 0;
}
```

```c
#include <stdio.h>
int main () {
    int a[] = {12,4,34,23,-2,0};
    int resultat = 0;
    int i = 0;
    while (i < sizeof(a)/sizeof(int)) {
        resultat = resultat + a[i];
        i++;
    }
    printf("resultat = %d\n", resultat);
    return 0;
}
```

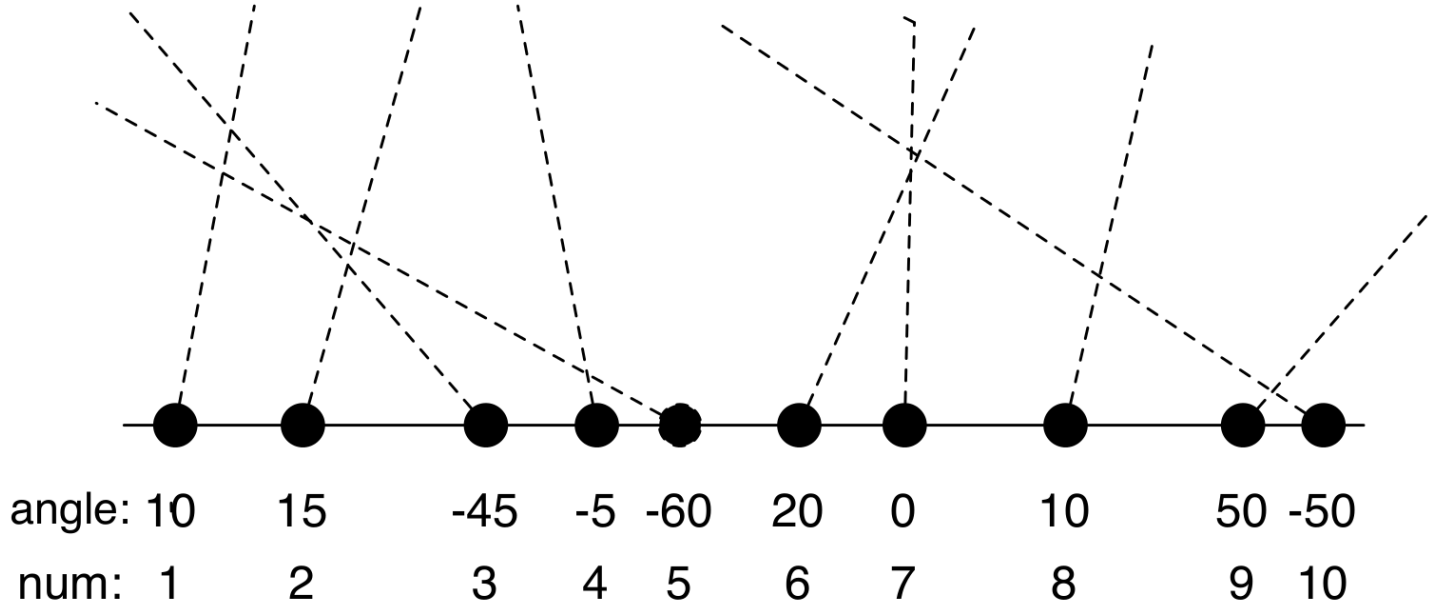
Résultat :

```
3c3
<      int a[] = {2,4,34,23,-2,0};
---
>      int a[] = {12,4,34,23,-2,0};
6c6
<      for (i = 0; i < sizeof(a)/sizeof(int); i++) {
---
>      while (i < sizeof(a)/sizeof(int)) {
7a8
>          i++;
9d9
<      printf("taille d'un int %d\n", sizeof(int));
```

Exercices beoi '12

- Synchronisation de données
  - soient deux listes ordonnées d'entiers `A[1..M]` et `B[1..N]`
  - donner les éléments se trouvant dans `A` mais pas dans `B`

- Ghostbusters
  - soient un nombre de chasseurs de fantôme avec un angle de tir fixe
  - renvoie un sous-ensemble maximal de chasseurs autorisés à tirer
  - sachant que les tirs ne peuvent pas se croiser



mailto: *fgi@info.fundp.ac.be*

Merci de votre attention

Questions? Remarques?