CRO: C langage and programming Roots
Slides pour accompagner (éventuellement) le TD5: Arbres et GDB

tanguy.risset@insa-lyon.fr
Lab CITI, INSA de Lyon
Version du October 16, 2025

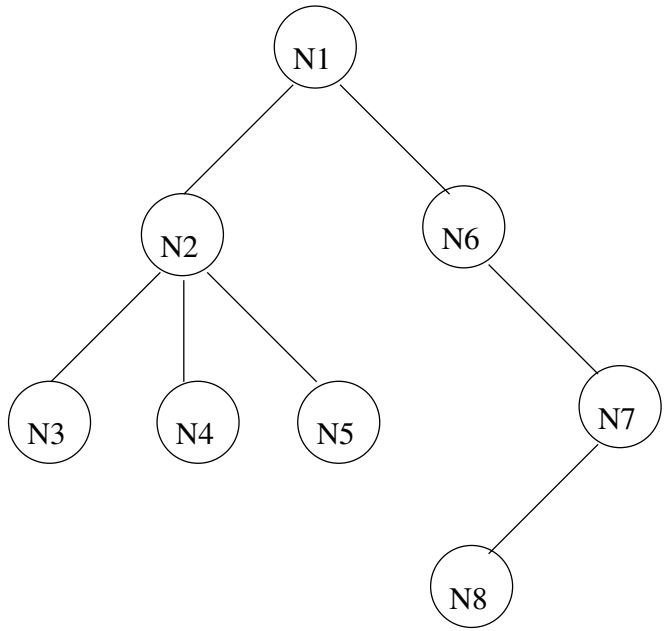
Tanguy Risset

October 16, 2025

Table of Contents

Les arbres

- Un arbre est défini de manière récursive, c'est:
  - Soit un arbre atomique (ou *noeud terminal* ou *feuille*)
  - Soit un *noeud* ayant pour *fils* d'autre arbres.
- En général on étiquete les noeuds avec une valeur.



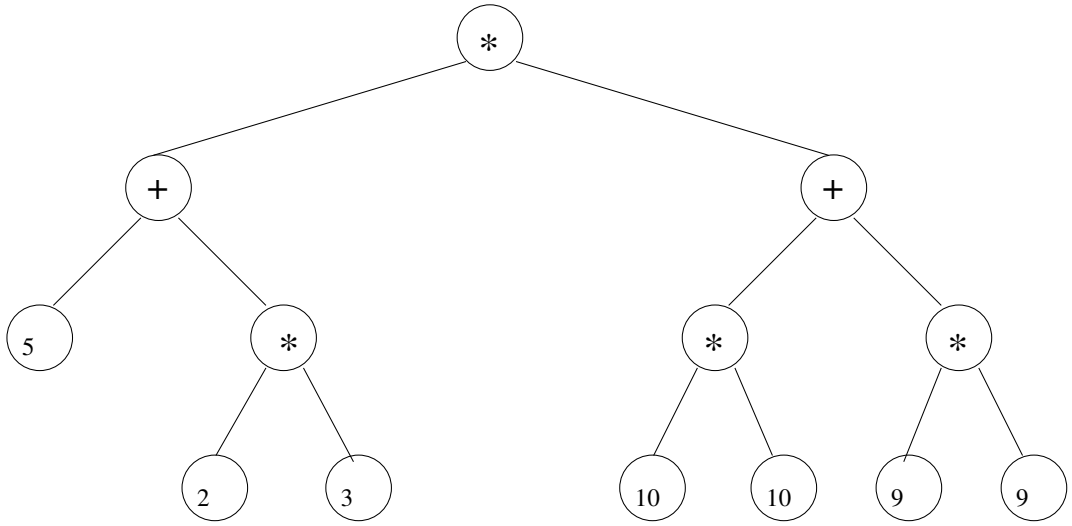
- Le noeud N1 est la *racine* de l'arbre.
- Les noeuds N3, N4, N5 et N8 sont des *feuilles* de l'arbre.
- Les noeuds N2, N6, N7 sont des *noeuds internes* de l'arbre.
- Le noeud N1 est le *père* des noeuds N2 et N6.
- La racine est un *ancêtre* de tous les noeuds de l'arbre.

Parcours d'un arbre binaire

- Un arbre binaire est un arbre dont les noeuds ont au plus deux fils
- Les algorithmes travaillant sur des arbres sont généralement récursifs.
- En langage de description algorithmique, on suppose disposer de fonctions d'accès: à partir de chaque noeud N on peut atteindre son père pere(N) et ses fils filsDroit(N), filsGauche(N)

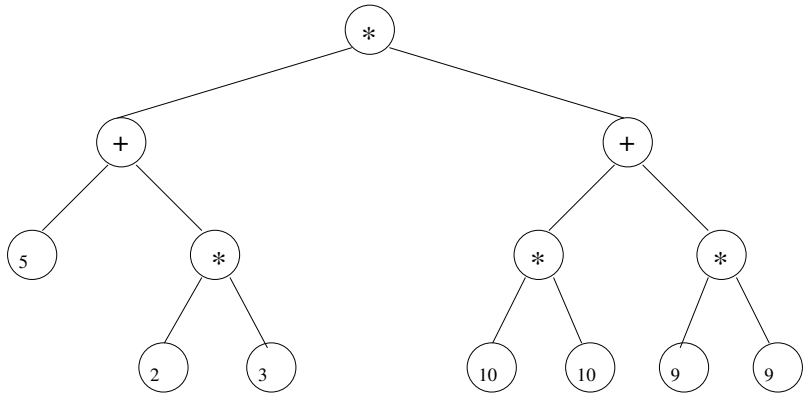
```
PRODECURE parcours(ARBRE N)
    SI filsGauche(N)≠ NULL ALORS parcours(filsGauche(N)) FINSI
    traiter(N);
    SI filsDroit(N)≠ NULL ALORS parcours(filsDroit(N)) FINSI
FIN
```

Arbre d'une expression arithmétique



- Si la procédure traiter(N) affiche la valeur d'un noeud, on obtient:
- $5 + 2 * 3 * 10 * 10 + 9 * 9$
- Est ce l'expression représentée par l'arbre?

Affichage d'une expression arithmétique



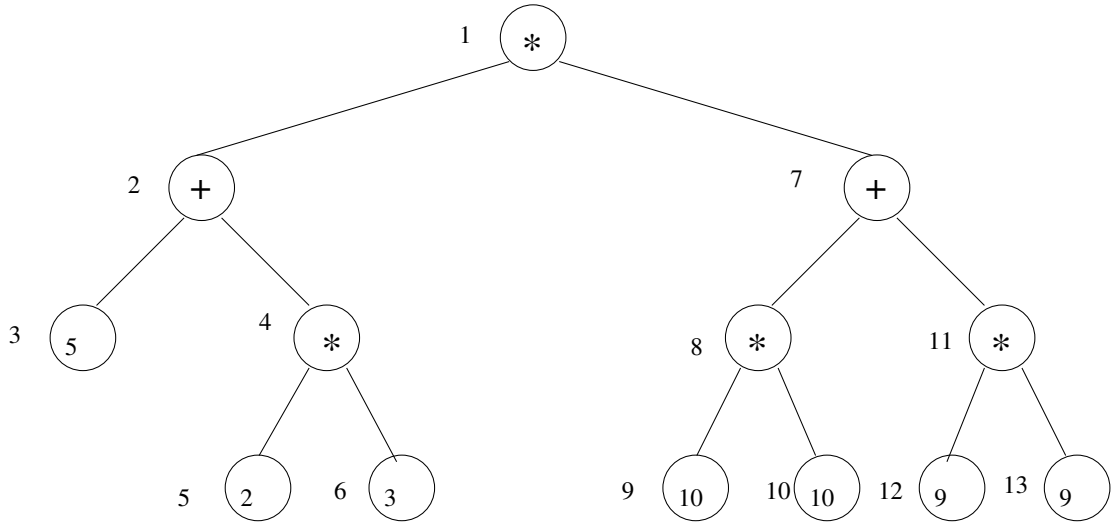

```
PRODECURE parcours(ARBRE N)
    SI (valeur(N)=='+') ALORS print('(') FINSI
    SI filsGauche(N)≠ NULL ALORS parcours(filsGauche(N)) FINSI
    PRINT(valeur(N));
    SI filsDroit(N)≠ NULL ALORS parcours(filsDroit(N)) FINSI
    SI (valeur(N)=='+') ALORS print(')') FINSI
FIN
```

- En rajoutant des parenthèses autour d'une addition:
- $(5 + 2 * 3) * (10 * 10 + 9 * 9)$

Différents parcours d'un arbre

- Suivant l'ordre dans lequel on écrit les instructions de la fonction de parcours, l'ordre de parcours des noeuds est différent.
- Trois possibilités principales:
  - Parcours préfixe : racine, sous-arbre gauche, sous-arbre droit
  - Parcours infixe : sous-arbre gauche, racine, sous-arbre droit
  - Parcours postfixe : sous-arbre gauche, sous-arbre droit, racine
- Il existe d'autres types de parcours moins facile a réaliser:
  - parcours en largeur d'abord.

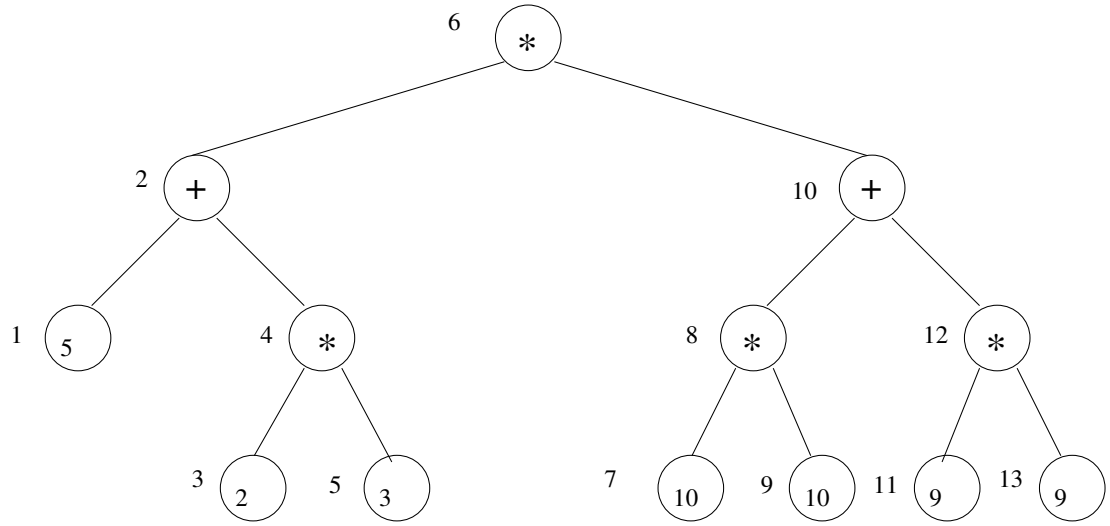
Parcours préfixe




```
PRODECURE parcoursPrefixe(ARBRE N)
    traiter(N);
    SI filsGauche(N)≠ NULL ALORS parcoursPrefixe(filsGauche(N)) FINSI
    SI filsDroit(N)≠ NULL ALORS parcoursPrefixe(filsDroit(N)) FINSI
FIN
```

```
* + 5 * 2 3 + * 10 10 * 9 9
```

Parcours infixe

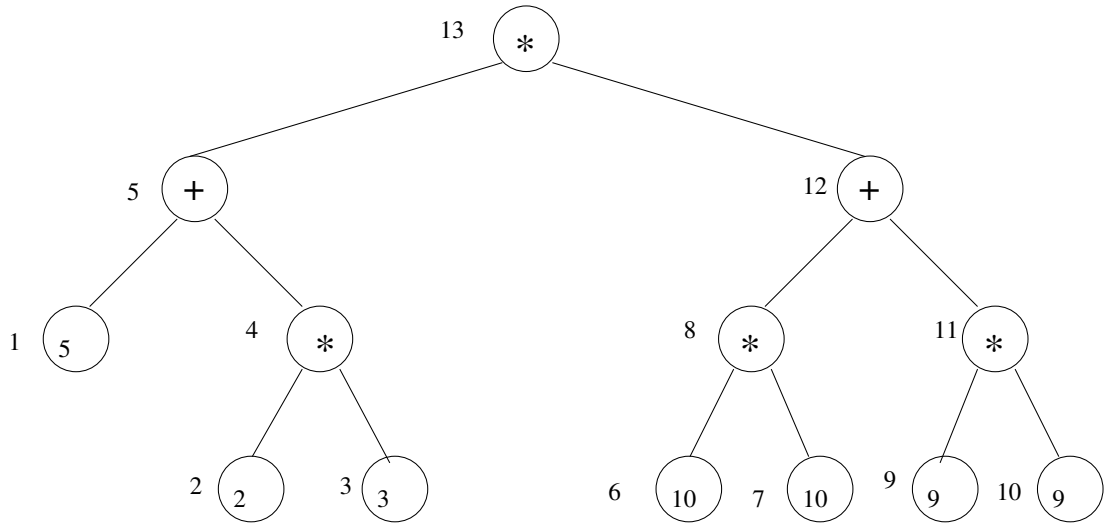



```
PRODECURE parcoursInfixe(ARBRE N)
    SI filsGauche(N)≠ NULL ALORS parcoursInfixe(filsGauche(N)) FINSI
    traiter(N);
    SI filsDroit(N)≠ NULL ALORS parcoursInfixe(filsDroit(N)) FINSI
FIN
```

5 + 2 * 3 * 10 * 10 + 9 * 9

(attention parenthèse nécessaires)

Parcours postfixe




```
PRODECURE parcoursPostfixe(ARBRE N)
    SI filsGauche(N)≠ NULL ALORS parcoursPostfixe(filsGauche(N)) FINSI
    SI filsDroit(N)≠ NULL ALORS parcoursPostefixe(filsDroit(N)) FINSI
    traiter(N);
FIN
```

5 2 3 * + 10 10 * 9 9 * + *

Implémentation en C

- Pour un arbre binaire:

```
struct model_noeud
{
  int val;
  struct model_noeud *filsGauche;
  struct model_noeud *filsDroit;
} ;

typedef struct model_noeud NOEUD;

typedef NOEUD *ARBRE;
```

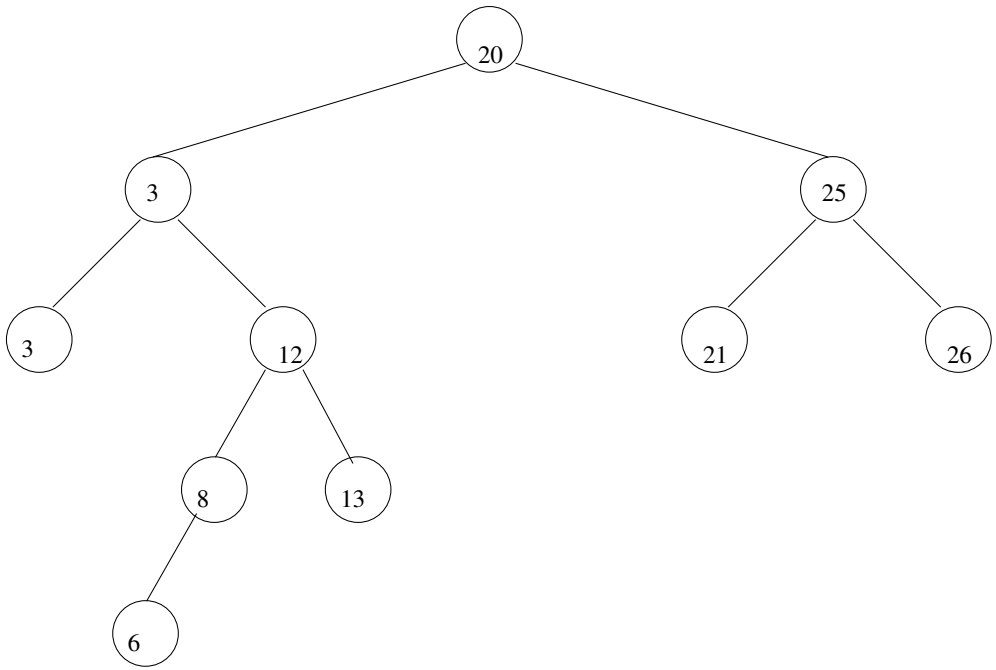
Construction d'un arbre en C

```
ARBRE nouvelArbre(int val, ARBRE fg, ARBRE fd)
{
  ARBRE temp;

  temp= (ARBRE)malloc(sizeof(NOEUD));
  temp->val=val;
  temp->filsGauche=fg;
  temp->filsDroit=fd;
  return(temp);
}
```

```
int main()
{

  ARBRE arbre,temp1,temp2;

  temp1=nouvelArbre(10,NULL,NULL);
  temp2=nouvelArbre(20,NULL,NULL);
  arbre=nouvelArbre(30,temp1,temp2);
  fprintf(stdout,"filsdroit->val=%d\n",arbre->filsDroit->val);
}
```

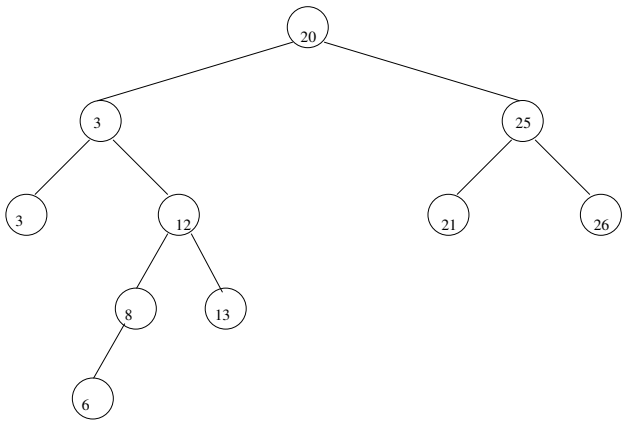
Arbres binaires de recherche

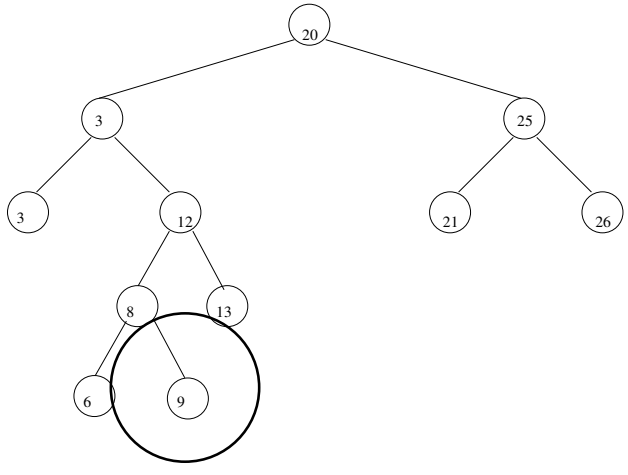
- Ce sont des arbres binaires de valeurs ordonnées avec les propriétés:
  - Tous les noeuds du sous arbre gauche d'un noeud ont une valeur inférieure (ou égale) à la sienne
  - Tous les noeuds du sous arbre droit d'un noeud ont une valeur supérieure (ou égale) à la sienne
- Utilisé pour stocker et rechercher rapidement des éléments dans une table qui évolue rapidement.



Insertion en ABR

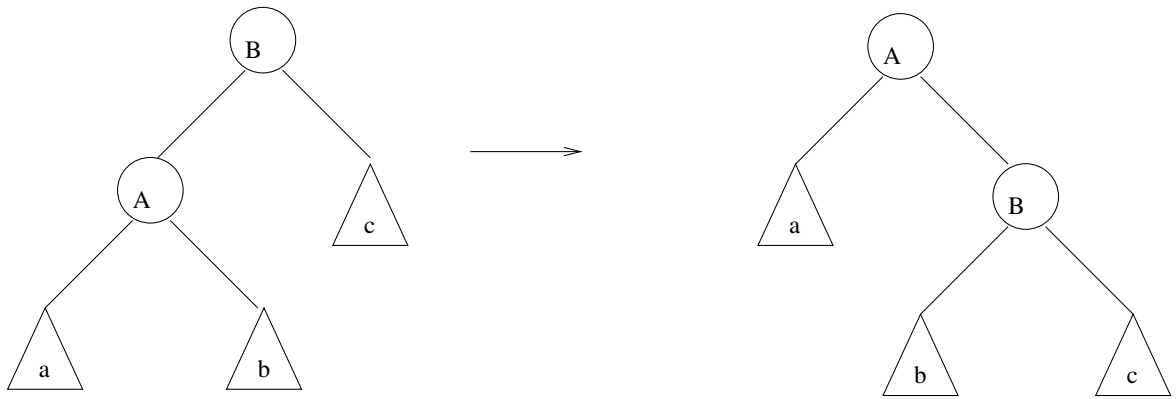
```
PRODECURE insertionABR(ELEMENT v, ARBRE A)
    SI (A==NULL) ALORS
        A=nouvelArbre(v,NULL,NULL)
    SINON
        SI (v<valeur(A)) ALORS
            insertionABR(v,filsGauche(A))
        SINON
            insertionABR(v,filsDroit(A))
        FINSI
    FINSI
    FIN
```





Arbre équilibré

- Un arbre binaire est *équilibré* (AVL pour Adel'son, Vel'Skii et Landis) si pour tout noeud, la différence de profondeur entre l'arbre de son fils gauche et l'arbre de son fils droit est au plus 1 (i.e. 0 ou 1).
- On peut montrer que la hauteur d'un AVL de N noeuds est de l'ordre de $Log(N)$
- On peut aussi modifier la procédure d'insertion dans un arbre de recherche pour que l'arbre résultant soit toujours équilibré.
- Pour cela on utilise la transformation de *rotation*

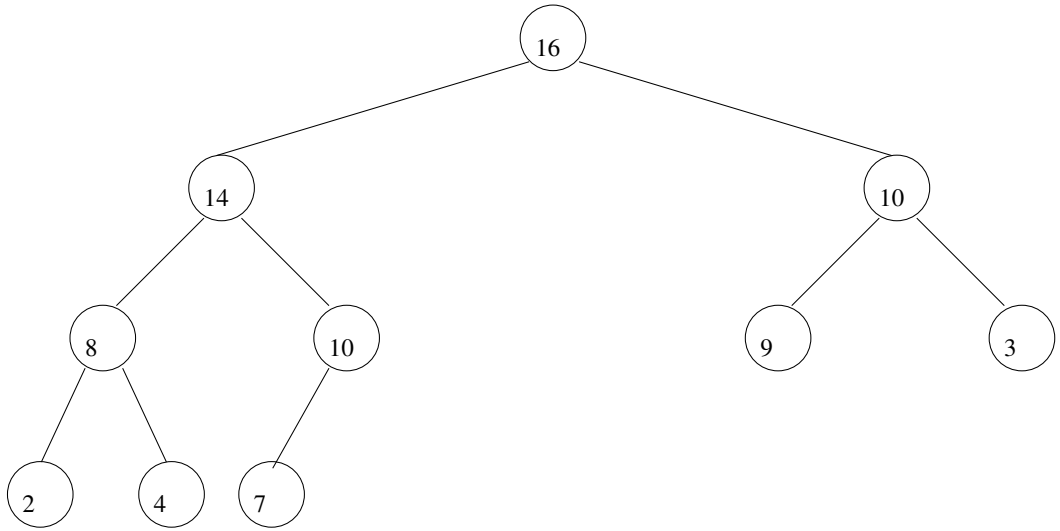


File de priorité

- Une file de priorité est une liste dans laquelle on peut ajouter ou supprimer des éléments, chaque élément ayant une priorité. Lors d'une suppression on prend toujours l'élément de priorité maximum.
- Comment implémenter une file de priorité pour que les opérations d'insertion et de suppression aient une complexité de $O(LogN)$
- On va utiliser une structure de tas.

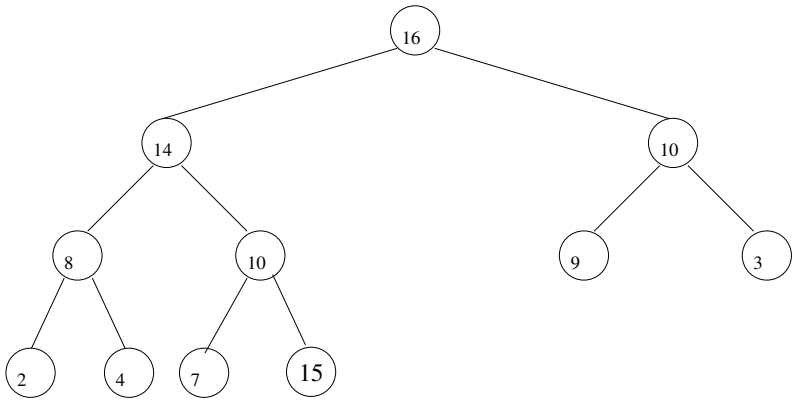
Tas

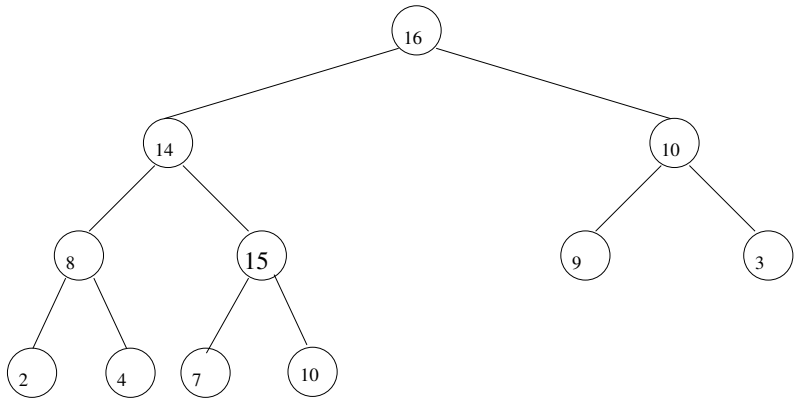
- Un tas est un arbre binaire avec les propriétés suivante:
  - La valeur de chaque noeud est supérieure ou égales à celle de ces fils.
  - L'arbre est quasi complet: tous les étages sont complets sauf éventuellement l'étage des feuilles et toutes les feuilles de l'étage des feuilles sont regroupées à gauche.

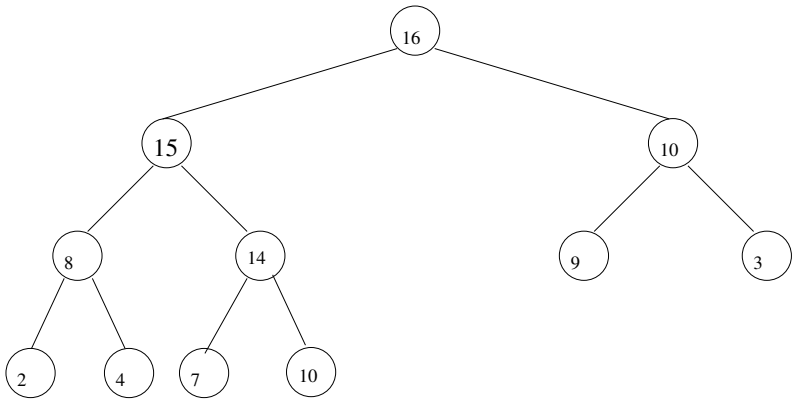


Algorithme d'insertion dans un tas

- On sait quelle noeud doit être ajouté: c'est le premier noeud libre de l'étage des feuilles, c.a.d La première case libre du tableau.
- Mais la valeur de ce nouveau noeud est peut être supérieure à son père. Dans ce cas on fait *remonter* le noeud en l'échangeant avec son père.

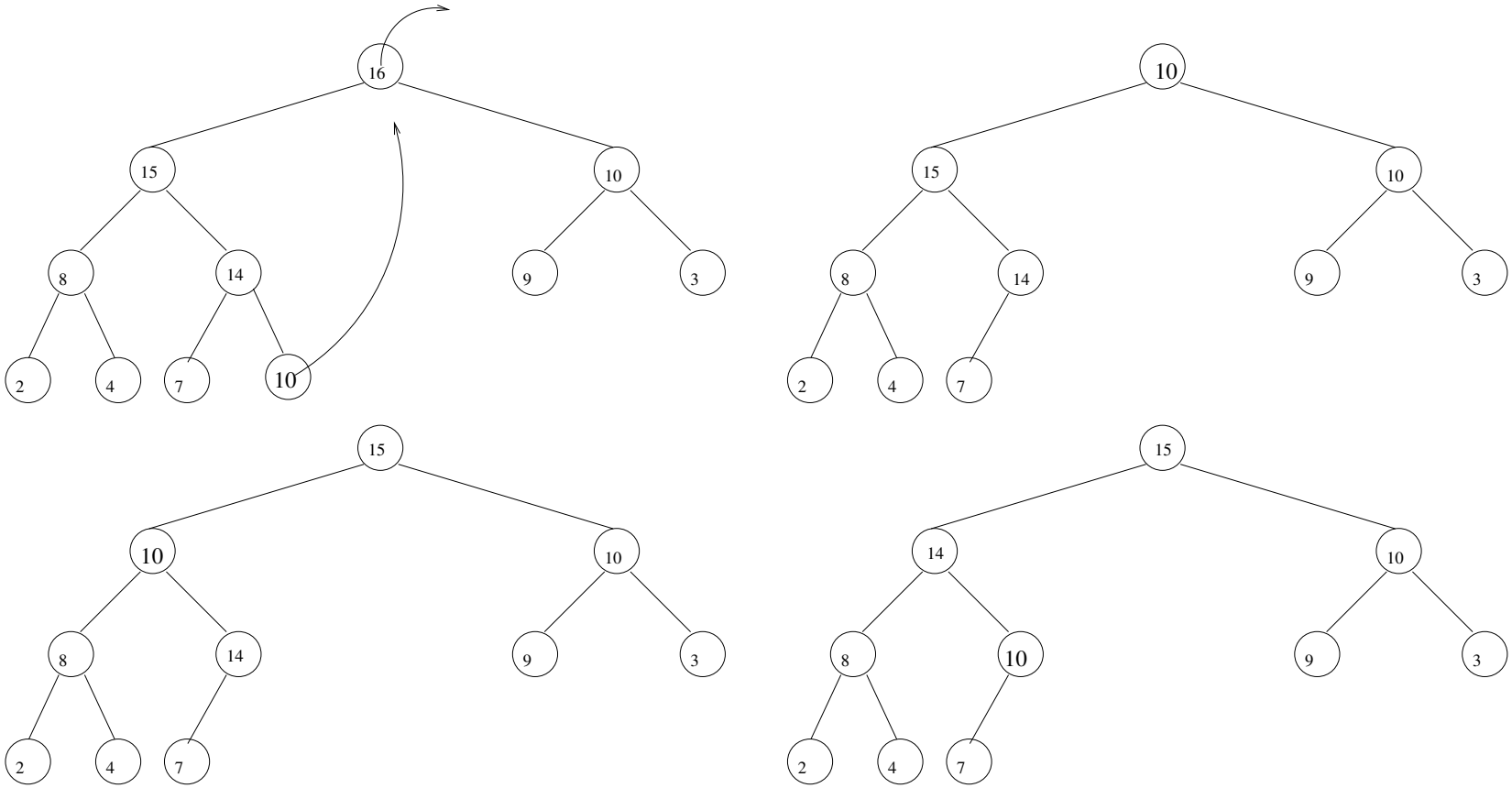






Suppression dans un tas

- L'élément à supprimer est toujours la racine (il reste deux arbres)
- On place le dernier élément de la ligne des feuille en racine et on le redescend jusqu'à sa place:



Complexité

- Les opérations d'insertion et de retrait dans un tas de taille N se font en temps $Log(N)$
- On peut utiliser ces propriétés pour construire un algorithme de tri qui est *toujours* en $NLog(N)$
- On insère les éléments dans un tas, puis on les retire un à un.
- En pratique, cette technique est moins bonne que QuickSort: les constantes sont trop grandes.

Arbre de syntaxe abstrait

- Lorsque l'on veut stocker une expression arithmétique, on a deux types de noeuds: entiers et opérateurs.
  $(5 + 2 * 3) * (10 * 10 + 9 * 9)$
- Quelle structure de donnée utiliser pour stocker une telle expression?
- Une solution souvent utilisée est une hiérarchie de structure combinée avec des unions

Retour sur les constructeurs de type

- Énumération: Permet de définir des constantes:
  `enum {LUNDI, MARDI, MERCREDI}`
  Définit trois constantes entières avec les valeur `LUNDI=0`, `MARDI=1` et `MERCREDI=2`. On peut forcer leurs valeur:
  `enum {LUNDI=12, MARDI=13, MERCREDI=14 }`
- Utilisé pour manipuler des informations symboliques.

Retour sur les constructeurs de type

- Union: permet de manipuler des variables pouvant prendre deux types différents: par exemple, soit un entier, soit une chaîne de caractère. La définition est similaire à celle d'une structure (`info` est l'étiquette de l'union):

```
union info
{
  int val;
  char *oper;
};
union info n;
```

- L'utilisation se fait comme pour une structure (`struct`) sauf qu'une variable `n` ne possède que l'un des deux champs: `val` ou `oper`:

```
n.val=1   ou   n.oper=(char *)malloc(2*sizeof(char));
               strcpy(oper,"-");
```

Type pour stocker une expression

```
typedef enum
    {VALEUR, MULT, ADD, DIV, SUB} typeNoeud;

struct model_noeud
{
  typeNoeud type;
  struct model_noeud *filsGauche ;
  struct model_noeud *filsDroit ;
  union
  {
    int val;
    char *oper;
  } info;
};

typedef struct model_noeud NOEUD;

typedef NOEUD *ARBRE;
```

Allocation des feuilles

Le champ type aide a determiner si la valeur est entière ou chaîne de caractère.

```
ARBRE nouvelFeuille(int val)
{
  ARBRE temp;

  temp= (ARBRE)malloc(sizeof(NOEUD));
  temp->info.val=val;
  temp->type=VALEUR;
  temp->filsGauche=NULL;
  temp->filsDroit=NULL;
  return(temp);
}
```

Allocation des noeud internes

```
ARBRE nouvelOper(typeNoeud type, ARBRE fg, ARBRE fd)
{
  ARBRE temp;
  char *oper;

  temp= (ARBRE)malloc(sizeof(NOEUD));
  oper=(char *)malloc(2*sizeof(char));
  switch (type)
    {
    case MULT:
      strcpy(oper,"*");
      temp->type=MULT;
      break;
  case ADD:
      [...]
  default: fprintf(stderr,"operateur non implementé\n");exit(-1);
    }
  temp->info.oper=oper;
  temp->filsGauche=fg;
  temp->filsDroit=fd;
  return(temp);
}
```

Parcours de la structure

```
int afficherArbre(ARBRE arbre)
{
  if ((arbre->type==ADD)||(arbre->type==SUB))
    fprintf(stdout,"(");
  switch (arbre->type)
    {
    case VALEUR: fprintf(stdout,"%d",arbre->info.val);
      break;
    default:
      afficherArbre(arbre->filsGauche);
      fprintf(stdout,"%s",arbre->info.oper);
      afficherArbre(arbre->filsDroit);
    };
  if ((arbre->type==ADD)||(arbre->type==SUB))
    fprintf(stdout,")");
}
```

Exemple d'utilisation

```
int main()
{

  ARBRE arbre, temp1,temp2,temp3;


  temp1=nouvelFeuille(10);
  temp2=nouvelFeuille(20);
  temp3=nouvelFeuille(400);
  arbre=nouvelOper(ADD,temp1,temp2);
  arbre=nouvelOper(MULT,arbre,temp3);

  afficherArbre(arbre);
  fprintf(stdout,"\n");

  return(0);
}
```

- resultat:
  (10+20)*400

Table of Contents

gdb: GNU symbolic debugger

- gdb est un debugger symbolique, c'est-à-dire un utilitaire Unix permettant de contrôler le déroulement de programmes C.
- gdb permet (entre autres) de mettre des points d'arrêt dans un programme, de visualiser l'état de ses variables, de calculer des expressions, d'appeler interactivement des fonctions, etc.
- xxgdb ou ddd sont des interfaces graphiques qui facilitent l'utilisation de gdb sous X-Window.
- gdb ne nécessite aucun système de fenêtrage, il peut s'exécuter sur un simple terminal shell (mode *console*).
- Il est indispensable de comprendre le fonctionnement en mode de gdb pour pouvoir utiliser les ddd.

Exemple de session gdb

```
int main()
{int i,*p;

  i=1;
  p=(int *)malloc(sizeof(int));
  *p=2;
  *p+=i;
  fprintf(stdout,"i=%d, p=%X, *p=%d\n",i,p,*p);
  free(p);
  return(0);
}
```

- compilation avec -g:
  `shell$ gcc -g exgdb.c -o exgdb`
- lancement de gdb: `shell$ gdb exgdb`
  `GNU gdb 6.3-debian`
  `(gbd)`

Exemple de session gdb

- Lorsque l'on lance `gdb` avec la commande `gdb exgdb.c`, `gdb` a chargé l'exécutable, il attend alors une commande `gdb`, comme par exemple `run` (pour exécuter le programme), `break` (pour mettre un point d'arret dans le programme), `step` (pour avancer d'une instruction dans le programme), etc.
- Les points d'arrêt peuvent se positionner au début des fonctions (`break main`, par exemple), ou à une certaine ligne (`break 6`, par exemple), ou lorsqu'une variable change de valeur (`watch i`, par exemple).

Exemple de session gdb

```
(gdb) break main
Breakpoint 1 at 0x8048424: file exgdb.c, line 6.
(gdb) run
Starting program: /home/trisset/cours/2005/AGP/cours_tri/exgd

Breakpoint 1, main () at exgdb.c:6
6         i=1;
(gdb)
```

gdb a lancé l'exécutable et arrêté l'exécution à la ligne 6 du fichier (à la première ligne de la fonction `main`), le code de cette ligne apparaît à l''écran.

- On peut avancer d'un pas dans le programme:

```
(gdb) step
7         p=(int *)malloc(sizeof(int));
(gdb)
(gdb) print i
$1 = 1
(gdb)
```

Exemple de session gdb

```
(gdb) display i
1: i = 1
(gdb)
```

- Affichage de pointeurs (comme une variable):

```
(gdb) display p
2: p = (int *) 0xb8000540
(gdb)
```

- Affichage d'objet pointés par les pointeurs :

```
(gdb) display *p
3: *p = -1208053760
(gdb)
```

```
(gdb) step
8           *p=2;
3: *p = 0
2: p = (int *) 0x804a008
1: i = 1
(gdb)
```

Exemple de session gdb

```
(gdb) step
9           *p+=i;
3: *p = 2
2: p = (int *) 0x804a008
1: i = 1
```

```
(gdb) step
10          fprintf(stdout,"i=%d, p=%X, *p=%d\n",i,p,*p);
3: *p = 3
2: p = (int *) 0x804a008
1: i = 1
```

```
(gdb) next
i=1, p=804A008, *p=3
11          free(p);
3: *p = 3
2: p = (int *) 0x804a008
1: i = 1
```

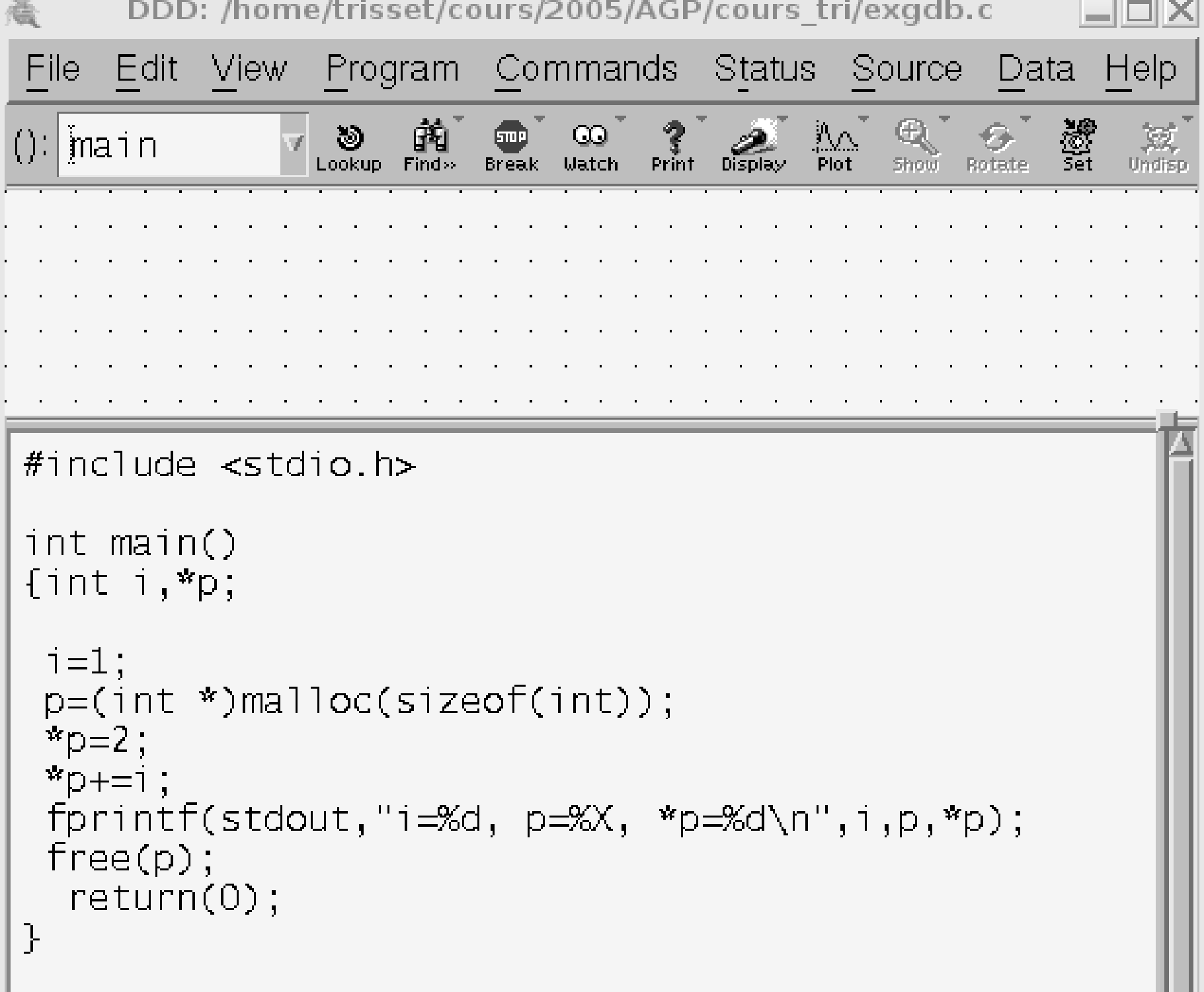
```
(gdb) cont
```

gdb commandes abrégées

- On peut taper la première lettres des commandesx:
- `r` est équivalent à `run`
- `b main` est équivalent à `break main`
- `p var` est équivalent à `print var`
- `d var` est équivalent à `display var`
- `s` est équivalent à `step`
- `n` est équivalent à `next`
- `c` est équivalent à `continue`
- La commande `run` peut prendre des arguments, le programme est alors exécuté avec les arguments donnés
- la commande `info` permet d'afficher des informations sur l'état du programme dans le débugger. Par exemple `info b` liste les points d'arrêt.
- La commande `help` est l'aide en ligne de gdb

ddd: data display debugger

- ddd est une interface graphique pour gdb.
- Après avoir compilé avec l'option -g, on tape:
  `ddd nomduprog`
  Par exemple:
  `ddd exgdb`


DDD: /home/trisset/cours/2005/AGP/cours_tri/exgdb.c
File Edit View Program Commands Status Source Data Help
(): main
Lookup Find» Break Watch Print Display Plot Show Rotate Set Undisp
#include <stdio.h>
int main()
{int i,*p;
i=1;
p=(int *)malloc(sizeof(int));
*p=2;
*p+=i;
fprintf(stdout,"i=%d, p=%X, *p=%d\n",i,p,*p);
free(p);
return(0);
}

d

```
#include <stdio.h>

int main()
{int i,*p;

 i=1;
 p=(int *)malloc(sizeof(int));
 *p=2;
 *p+=i;
 fprintf(stdout,"i=%d, p=%X, *p=%d\n",i,p,*p);
 free(p);
  return(0);
}
```

```
Using host libthread_db library
"/lib/tls/libthread_db.so.1".
(gdb)
```

ddd: fenêtre de commande

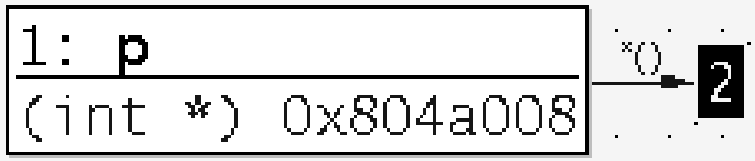
d

```
int main()
{int i,*p;

=1;
 p=(int *)malloc(sizeof(int));
 *p=2;
 *p+=i;
 fprintf(stdout,"i=%d, p=%X, *p=%d\n",i,p,*p);
 free(p);
  return(0);
}
```

```
(gdb) graph display p
No symbol "p" in current context.
(gdb) run

Breakpoint 1, main () at exgdb.c:6
(gdb)
```

Breakpoint 1, main () at exgdb.c:6

d




```
#include <stdio.h>

int main()
{int i,*p;

 i=1;
 p=(int *)malloc(sizeof(int));
 *p=2;
 *p+=i;
 fprintf(stdout,"i=%d, p=%X, *p=%d\n",i,p,*p);
 free(p);
  return(0);
}
```

```
(gdb) graph display p
(gdb) graph display *p dependent on 1
```