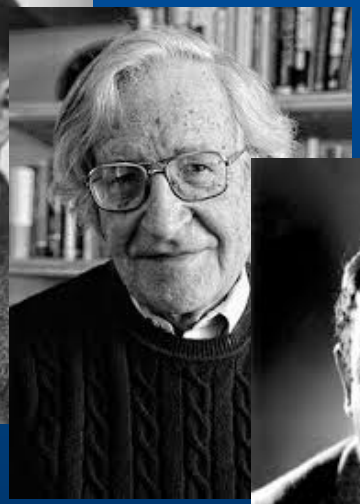
Algorithmique - fonctions

Informatique et Science du Numérique

Les 4 fondamentaux de l'informatique

1. Données
2. Algorithme
3. Langage
4. Machine






Algorithmique - fonctions

Informatique et Science du Numérique

Exemple : la balle

Une balle est lâchée d'une hauteur H = 10 m et rebondit sur le sol. À chaque rebond, le sol absorbe 10 % de l'énergétique de la balle. La balle s'arrête de rebondir lorsque la hauteur du rebond devient inférieure à 1 mm.

Déterminer :
1. le nombre N de rebonds effectués
2. la distance d, en mètres, parcourue par la balle

Algorithmique - fonctions

Informatique et Science du Numérique

```
Algorithme Rebond
constantes
        (hauteur : réel) := 10000.0     {en mm}
        (limite : réel) := 1.0          {balle à l'arrêt}
        (abs : réel) := 0.1
variables
        (rebond : entier) := 0
        (distance : réel) := hauteur
        (h : réel) := hauteur
début
        tantque (h > limite) ET (abs < 1) faire
        début
                h := h * (1 - abs)
                distance := distance + (2 * h)
                rebond := rebond + 1
        fin

        afficher(rebond, distance / 1000)
fin
```

Algorithmique - fonctions

Informatique et Science du Numérique

hauteur
limite
abs

Rebond
distance

généricité
Abstraction IHM

Algorithmique - fonctions

Informatique et Science du Numérique

```
fonction Calcul_Rebond(hauteur : réel, limite : réel, abs : réel) : entier
variables
        (rebond : entier) := 0
        (distance : réel) := hauteur
        (h : réel) := hauteur
début
        tantque (h > limite) ET (abs < 1) faire
        début
                h := h * (1 - abs)
                distance := distance + (2 * h)
                rebond := rebond + 1
        fin

        Calcul_Rebond := rebond
Fin

Algorithme Rebond
début
        afficher( Calcul_Rebond(10000, 1, 0.1) )
fin
```

preuve de l'arrêt ?

convergence si $0 < abs \leq 1$

et la distance ?

Algorithmique - fonctions

Informatique et Science du Numérique

```c
#include <stdio.h>

// fonction prototype
static unsigned int calcul_rebonds(const float, const float, const float);

int main()
{
   printf("%d", calcul_rebonds(10000, 1, 0.1));
   return 0;
}

static unsigned int calcul_rebonds(const float hauteur, const float limite, const float abs)
{
                unsigned int rebond = 0;
                float distance = hauteur;
                float h = hauteur;

                while ( h > limite && abs < 1 )
                {
                                h           = h * (1 - abs);
                                distance = distance + (2 * h);
                                rebond   = rebond + 1;
                }

   return rebond;
}
```

Algorithmique - fonctions

Informatique et Science du Numérique

```c
#include <stdio.h>

// fonction prototype
static unsigned int calcul_rebonds(const float, const float, const float);

int main()
{
   printf("%d", calcul_rebonds(10000, 1, 0.1));
   return 0;
}

static unsigned int calcul_rebonds(const float hauteur, const float limite, const float abs)
/*
 * function : calcul le nombre de rebonds d'une balle et sa distance parcourue
 * in : hauteur : hauteur initiale en mm, limite : hauteur avant arrêt en mm, abs : coefficient d'absorption de l'Ec du sol
 * out : nombre de rebonds
 */
{
                unsigned int rebond = 0;
                float distance = hauteur;
                float h = hauteur;

                while ( h > limite && abs < 1 )
                {
                                h           = h * (1 - abs);
                                distance = distance + (2 * h);
                                rebond   = rebond + 1;
                }

   return rebond;
}
```

Algorithmique - fonctions

Informatique et Science du Numérique

```
static int calcul_rebonds(const float hauteur, const float limite, const float abs)
/*
 * function : calcul le nombre de rebonds d'une balle et sa distance parcourue
 * in : hauteur : hauteur initiale en mm, limite : hauteur avant arrêt en mm, abs : coefficient d'absorption de l'Ec du sol
 * out : nombre de rebonds si OK, -1 si erreur
 */
{
        // gestion des erreurs
        if ( hauteur < 0 ||             // pb de cohérence si hauteur négative
             limite <= 0 ||             // pb de condition d'arrêt si limite nulle
             abs <= 0)                  // pb de convergence si coef d'absorption nul
                        return -1;

        int    rebond   = 0;            // 2 à 4 octets : max = 2 147 483 647
        float distance = hauteur;       // 4 octets : max = 1.10^37
        float h          = hauteur;

        while ( h > limite && abs < 1 )
        {
                        h          = h * (1 - abs);
                        distance = distance + (2 * h);      // pb de modulo si distance = distance + (2 * h) > 1.10^37
                        rebond   = rebond + 1;              // pb si convergence lente :  abs <  1.10^-8
        }

   return rebond;
}
```

Algorithmique - fonctions

Informatique et Science du Numérique

```
static int calcul_rebonds(const float hauteur, const float limite, const float abs)
/*
 * function : calcul le nombre de rebonds d'une balle et sa distance parcourue
 * in : hauteur : hauteur initiale en mm, limite : hauteur avant arrêt en mm, abs : coefficient d'absorption de l'Ec du sol
 * out : nombre de rebonds si OK, < 0 si erreur
 */
{
    // gestion des erreurs
    if ( hauteur < 0 || hauteur > 100000 )      // pb de cohérence si hauteur négative
                return -1;
    if ( limite <= 0 )                          // pb de condition d'arrêt si limite nulle
                return -2;
    if ( abs < 0 )                              // pb de cohérence si coefficient nul
                return -3;
    if ( abs == 0 )                             // pb de convergence si coef d'absorption nul
                return -4;

    int    rebond   = 0;            // 2 à 4 octets : max = 2 147 483 647
    float distance = hauteur;       // 4 octets : max = 1.10^37
    float h        = hauteur;

    while ( h > limite && abs < 1 )
    {
                h        = h * (1 – abs);
                distance = distance + (2 * h);  // pb de modulo si distance = distance + (2 * h) > 1.10^37
                rebond   = rebond + 1;          // pb si convergence lente :  abs <  1.10^-8
    }

    return rebond;
}
```

pb aux limites

Algorithmique - fonctions

Informatique et Science du Numérique

```c
#include <stdio.h>

// fonction prototype
static int calcul_rebonds(const float, const float, const float);

int main()
{
        int rebond = calcul_rebonds(10000, 1, 0.1);

        switch ( rebond )
        {
            case -1 :
            case -2 :
            case -3 :
               printf("erreur paramètre");
               return -1;

            case -4 :
               printf("rebond infini");
               break;

            default:
               printf("%d rebonds", rebond);
               break;
        }

        return 0;
}
```

```
variables
        (rebond : entier) := calcul_rebonds(10000, 1, 0.1)

selon rebond faire
début
        cas -1 :
        cas -2 :
        cas -3 :
                afficher(« erreur paramètre »)
                finselon

        cas -4 :
                afficher(« rebond infini »)
                finselon

        défaut :
                afficher(rebond)
                finselon
fin
```

Algorithmique - fonctions

Informatique et Science du Numérique

Et la distance ?

Problème : une fonction retourne 1 seule valeur…

Solution : utiliser des pointeurs qui travaillent directement en mémoire !

Algorithmique - fonctions

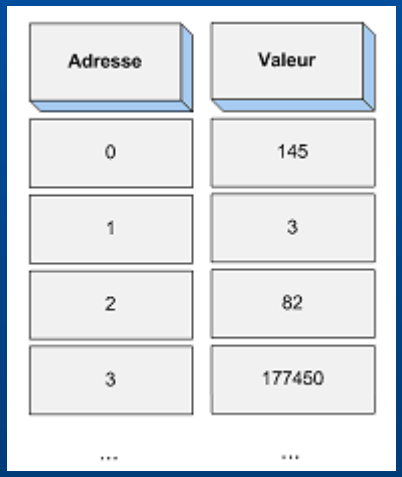
Informatique et Science du Numérique

Adresse & pointeur

Un pointeur pointe sur une @ en mémoire.

pi →

pi = 2
*pi = 82




```
int     i = 82;          // i est un entier
int  *pi = &i;           //p i est un pointeur sur un entier

printf("%ld / %ld\n", pi, &i);      // affiche « 2 / 2 »
printf("%d / %d", *pi, i);          // affiche « 82 / 82 »
```

Algorithmique - fonctions

Informatique et Science du Numérique

Exemple

Écrire une fonction qui échange deux valeurs : echange(a,b)

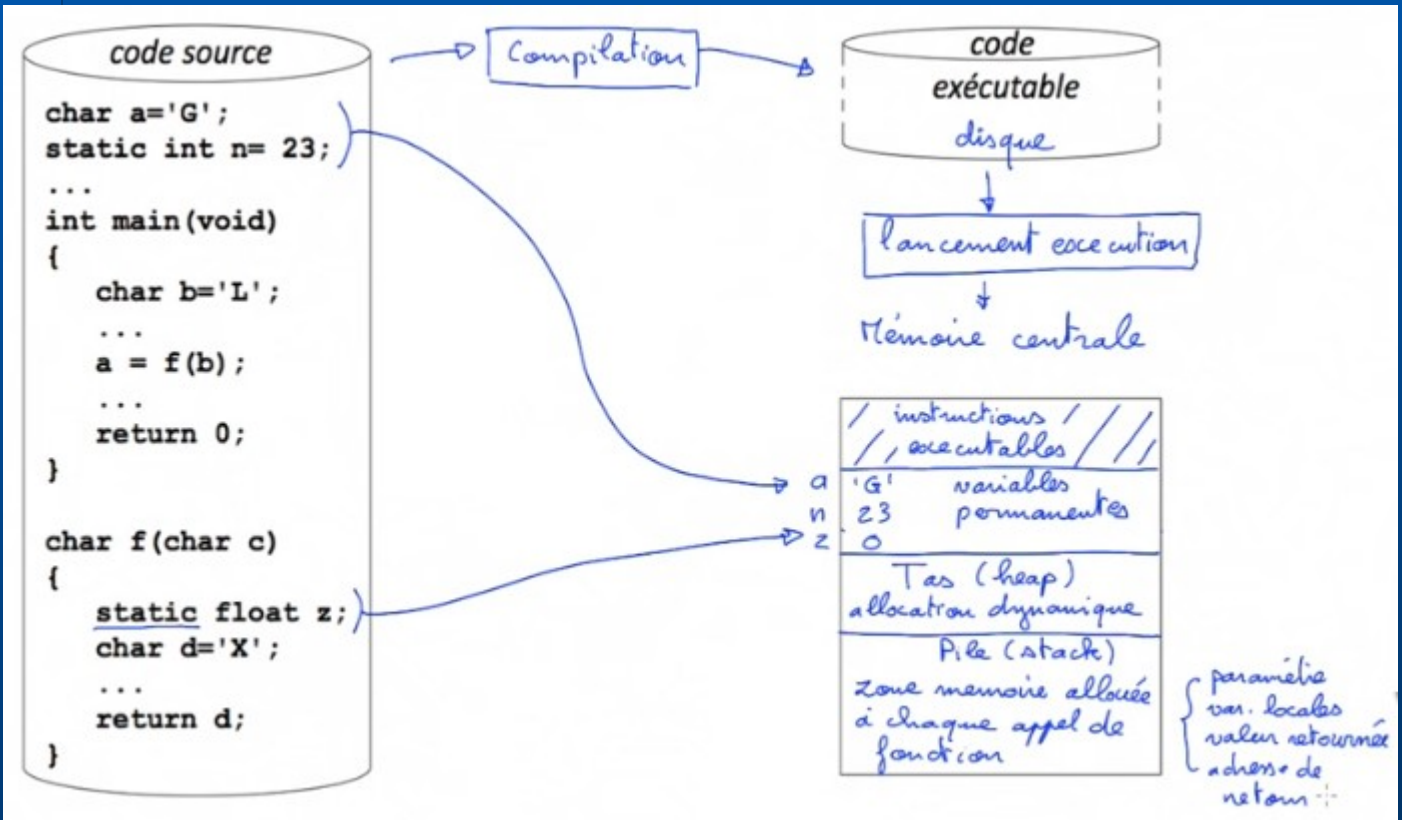
```
// fonction prototype
static void echange(int, int);

int main()
{
        int a = 1, b = 2;

        echange(a, b);
        printf("%d %d", a, b);

        return 0;
}

static void echange(int a, int b);
{
        int c = a;
        a = b;
        b = c;
}
```

```
// fonction prototype
static void echange(int*, int*);

int main()
{
        int a = 1, b = 2;

        echange(&a, &b);
        printf("%d %d", a, b);

        return 0;
}

static void echange(int* a, int* b);
{
        int c = *a;
        *a = *b;
        *b = c;
}
```

Algorithmique - fonctions

Informatique et Science du Numérique

Comment ça marche ?

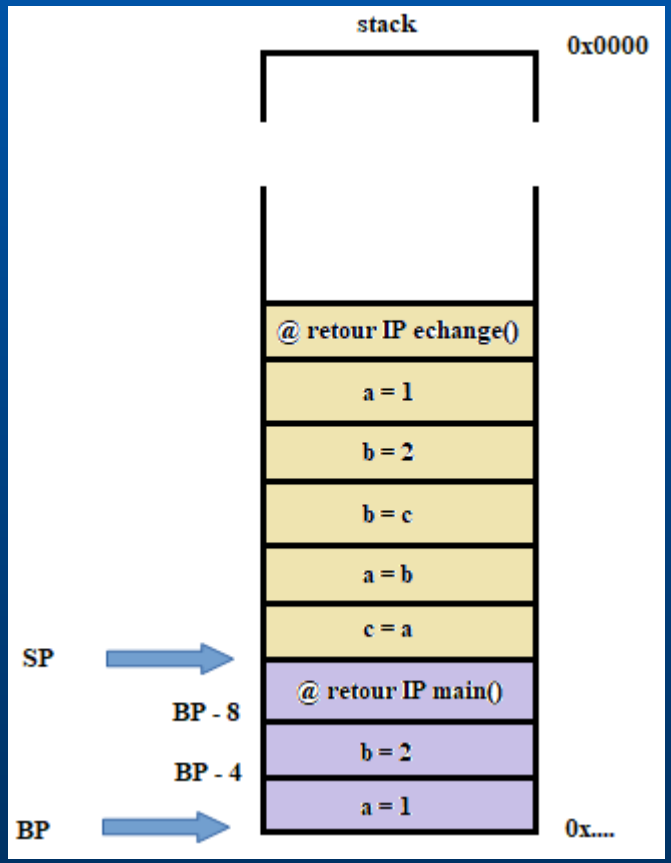


Algorithmique - fonctions

Informatique et Science du Numérique

Comment ça marche ?

```
static void echange(int* a, int* b);
{
        int c = *a;
        *a = *b;
        *b = c;
}

int main()
{
        int a = 1, b = 2;

        echange(&a, &b);

        return 0;
}
```



Algorithmique - fonctions

Informatique et Science du Numérique

```
// fonction prototype
static unsigned int calcul_rebonds(const float, const float, const float, unsigned int *, float *);

int main()
{
        unsigned int rebond;
        float distance;

        calcul_rebonds(1000, 1, 0.1, &rebond, &distance) ;

        printf("%d -> %.2f m", rebond, distance / 1000);

        return 0;
}

static void calcul_rebonds(const float hauteur, const float limite, const float abs, unsigned int *rebond, float *distance)
{
        *rebond   = 0;
        *distance = hauteur;

        float h = hauteur;

        while ( h > limite && abs < 1 )
        {
                h         = h * (1 - abs);
                *distance = *distance + (2 * h);
                *rebond   = *rebond + 1;
        }
}
```

on utilise des pointeurs

on passe les @ des variables

on met à jour les valeurs passées par @

Algorithmique - fonctions

Informatique et Science du Numérique

Conclusion

Une fonction est une boite noire

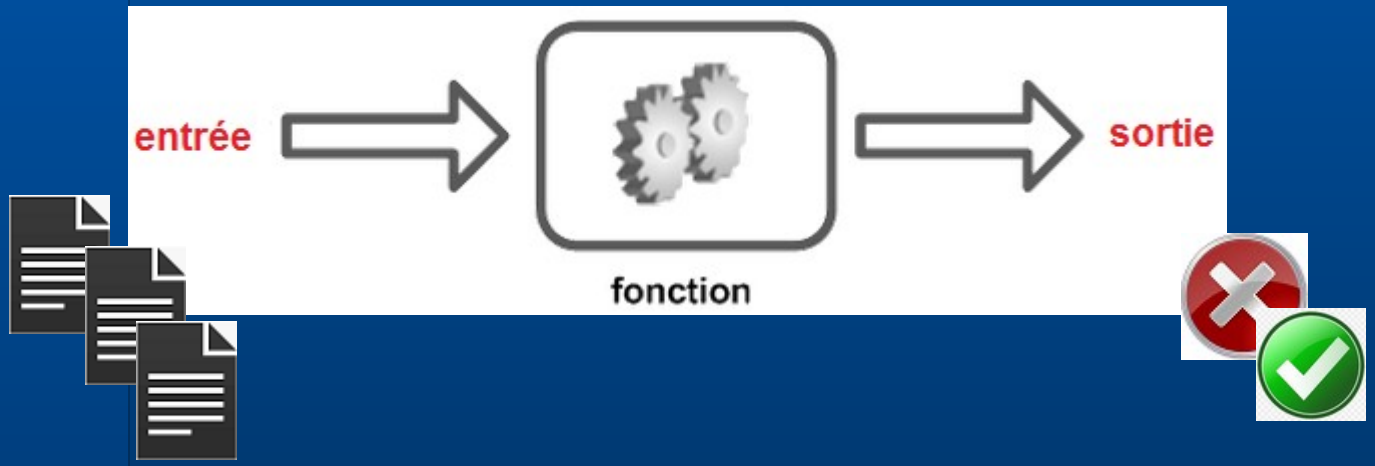
1. documenter la fonction
2. faire remonter les erreurs vers les couches supérieures
3. Écrire de façon générique
4. Utiliser les pointeurs à partir de 2 données en retour

Algorithmique - fonctions

Informatique et Science du Numérique

Les tests unitaires

Une série de tests ne prouvera jamais qu'une fonction est correcte



Classes de tests

FAILED ou PASSED

Algorithmique - fonctions

Informatique et Science du Numérique

Vecteur

```
int main()
{
   const float test[5][3] = {
         {100, .1, .1},
         {100, 1, .01},
         {1000, .1, .1},
         {1000, 1, .1},
         {10000, 1, .1}
      };

   for (int i = 0; i < 5; i++)
   {
      unsigned int rebond;
      float distance;

      printf("%d : ", i);

      if ( calcul_rebonds(test[i][0], test[i][1], test[i][2],
                  &rebond, &distance) < 0 )
         printf("FAILED");
      else
         printf("PASSED");

      printf("\n");
   }

   return 0;
}
```

```
int main()
{
      array<array<float, 3>, 5>   test = {
         100, .1, .1,
         100, 1, .01,
         1000, .1, .1,
         1000, 1, .1,
         10000, 1, .1
          };

   for (int i(0); i < 5; i++)
   {
      unsigned int rebond;
      float distance;

      cout << i << " : ";

      if ( calcul_rebonds(test[i][0], test[i][1], test[i][2],
                  rebond, distance) < 0 )
         cout << "FAILED";
      else
         cout << "PASSED";

      cout << endl;
   }

   return 0;
}
```

Algorithmique - fonctions

Informatique et Science du Numérique

Générateur

```
int main()
{
   srand(time(NULL));  // initialisation générateur aléatoire

   for (int i = 0; i < 5; i++)
   {
      unsigned int rebond;
      float distance;

      printf("%d : ", i);

      const float hauteur = 100000 * (float) rand() / RAND_MAX;
      if ( calcul_rebonds(
                  hauteur,
                  hauteur / (rand() + 1),
                  (const float) rand() / RAND_MAX,
                  &rebond, &distance) < 0 )
         printf("FAILED");
      else
         printf("PASSED");

      printf("\n");
   }

   return 0;
}
```

```
int main()
{
   default_random_engine generator;
   generator.seed(time(NULL));

   uniform_real_distribution<float> hauteur(0.0, 100000.0);
   uniform_real_distribution<float> limite(0.0, 10.0);
   uniform_real_distribution<float> abs(0.0, 1.0);

   for (int i(0); i < 5; i++)
   {
      unsigned int rebond;
      float distance;

      cout << i << " : ";

      if ( calcul_rebonds(
                  hauteur(generator), limite(generator), abs(generator),
                  rebond, distance) < 0 )
         cout << "FAILED";
      else
         cout << "PASSED";

      cout << endl;
   }

   return 0;
}
```

Algorithmique - fonctions

Informatique et Science du Numérique

Fichier

```
int main()
{
   FILE *fp = fopen("test.txt", "r");
   if ( fp == NULL )
      return -1;

   float hauteur, limite, abs;
   while ( fscanf(fp, "%f %f %f\n", &hauteur, &limite, &abs) != EOF )
   {
      printf("%6.0f %6.2f %.5f : ", hauteur, limite, abs);

      unsigned int rebond;
      float distance;
      if ( calcul_rebonds(hauteur, limite, abs, &rebond, &distance) < 0 )
         printf("FAILED");
      else
         printf("PASSED");

      printf("\n");
   }

   if ( fclose(fp) == EOF )
     return -2;

   return 0;
}
```

Algorithmique - fonctions

Informatique et Science du Numérique

Fichier

```
static const int generation(const char* fichier, const unsigned int total)
/*
 * function : génération d'un fichier de tests unitaires
 * in : fichier,  nom du fichier test; total, nombre de tests à générer
 * out : 0 si OK, < 0 si erreur
 */
{
   if ( total == 0 )
      return -1;

   FILE *fp = fopen(fichier, "w");
   if ( fp == NULL )
      return -2;

   for (unsigned int i=0; i < total; i++)
   {
      const float hauteur = 100000 * (float) rand() / RAND_MAX;

      if ( fprintf(fp, "%f %f %f\n", hauteur, hauteur / (rand() + 1), (const float) rand() / RAND_MAX) == EOF )
         return -3;
   }

   if ( fclose(fp) == EOF )
      return -4;

   return 0;
}
```