

Partie 6 : Variables

- ▶ Qu'est-ce qu'une variable?
- ▶ Identificateurs
- ▶ Portée
- ▶ Masquage

AlgoProg 2015-16 124

Variables

- ▶ Une variable permet de **mémoriser** des valeurs.
- ▶ Un **emplacement en mémoire (adresse)** lui est attribué.
- ▶ Une définition de variable définit :
 - **son nom** : un « identificateur »
 - **sa portée** : les portions de code où on peut l'utiliser.
 - **son type** : les valeurs qu'elle peut prendre, les opérations possibles.
 - **sa classe d'allocation** : indique la zone mémoire où elle est stockée.
 - **sa valeur initiale**, éventuellement.
- ▶ Certaines de ces caractéristiques peuvent être définies, en partie ou complètement, par la **place** de la définition dans le source C.

Exemple de définition :

▶ `static int x = 1234;`

AlgoProg 2015-16 125

Variables : identificateurs

Le **nom** d'une variable est un **identificateur**.

- ▶ commence par une lettre a,...,z, A,...,Z ou _.
- ▶ peut contenir les caractères a,...,z, A,...,Z, 0,...,9, _.
- ▶ jusqu'à 31 caractères significatifs.
- ▶ les compilateurs actuels différencient majuscules et minuscules.
- ▶ La norme précise des limitations (rarement suivies).

AlgoProg 2015-16 126

Variables : portée

- ▶ **Portée** d'une variable = partie du programme où on peut l'utiliser.
- ▶ Une variable **définie hors** du corps de toute fonction est **globale**.
- ▶ Une variable **définie dans** le corps d'une fonction est **locale**.
- ▶ On ne peut **utiliser** une variable que dans le corps d'une fonction.
- ▶ On peut utiliser une variable globale
 - dans toute fonction du fichier après sa définition, et
 - dans un autre fichier en la **déclarant** avec le mot réservé **extern**.
- ▶ Une déclaration annonce au compilateur qu'une variable (globale) est définie dans un autre fichier, en donnant son type.
- ▶ Exemple de déclaration :

`extern int x;`

AlgoProg 2015-16 127

Variables : portée

- Une variable définie dans le corps d'une fonction est dite **locale**.
- Le corps d'une fonction contient des blocs emboîtés les uns dans les autres, délimités par **{ et }**.
- Chaque bloc contient ses définitions de variables, suivies d'instructions et de sous-blocs.
- La portée d'une variable locale est **limitée au bloc** dans laquelle elle est définie, et à ses sous-blocs.
- La **position** des définitions de variables dans le programme influe donc sur leur **portée**.
- En C99, on peut définir des variables de boucles.

AlgoProg 2015-16 128

Variables : portée

```
int x;      /* portée de x*/
void f(int a) {
    int y;
    /*
    */
{
    int z;
    /*
    */
}
/*
*/
}

int t;
void g(void)
{
/*
*/
}
```

AlgoProg 2015-16 129

Variables : portée

```
int x;
void f(int a) { /* portée de a*/
    int y;
    /*
    */
{
    int z;
    /*
    */
}
/*
*/
}

int t;
void g(void)
{
/*
*/
}
```

AlgoProg 2015-16 130

Variables : portée

```
int x;
void f(int a) {
    int y;      /* portée de y*/
    /*
    */
{
    int z;
    /*
    */
}
/*
*/
}

int t;
void g(void)
{
/*
*/
}
```

AlgoProg 2015-16 131

Variables : portée

```
int x;
void f(int a) {
    int y;
    /*
     */
    {
        int z; /* portée de z*/
        /*
         */
    }
    /*
     */
}
int t;
void g(void)
{
/*
*/
}
```

AlgoProg 2015-16 132

Variables : portée

```
int x;
void f(int a) {
    int y;
    /*
     */
    {
        int z;
        /*
         */
    }
    /*
     */
}
int t; /* portée de t*/
void g(void)
{
/*
*/
}
```

AlgoProg 2015-16 133

Variables : portée

```
void f(int a) {
    int i; /* portée de i*/
    /*
     */
    for(i=1;i<10;i++)
    {
        /*
         */
    }
    /*
     */
}
```

AlgoProg 2015-16 134

Variables : portée

```
void f(int a) {
    /*
     */
    for(int i=1;i<10;i++)
    {
        /*
         */
    }
    /*
     */
}
```



AlgoProg 2015-16 135

Variables : masquage

- ▶ On dit qu'une variable est **visible** dans les blocs de sa portée.
- ▶ Si 2 variables de même nom sont visibles dans le même bloc, le nom fait référence à la variable définie dans le bloc le plus interne.
- ▶ En cas de conflit de nom, la variable définie dans le bloc le plus interne **masque** l'autre.

AlgoProg 2015-16 136

Variables : masquage

```
#include <stdio.h>
int x = 7;
int main(void) {
    printf("x = %d\n", x); /* affiche 7 */
    {
        int x = 10;
        printf("x = %d\n", x); /* affiche 10 */
    }
    int x = 3;
    printf("x = %d\n", x); /* affiche 3 */
}
printf("x = %d\n", x); /* affiche 10 */
printf("x = %d\n", x); /* affiche 7 */
```

AlgoProg 2015-16 137

Variables : définition et déclaration

- ▶ Une variable doit être définie ou déclarée avant d'être utilisée.
- ▶ la déclaration associe un type avec un nom de variable.
- ▶ la définition, **en plus**
 - demande l'allocation mémoire pour la variable,
 - donne une valeur initiale.

AlgoProg 2015-16 138

Variables : définition et déclaration

- ▶ Une variable doit être définie ou déclarée avant d'être utilisée.
- ▶ la déclaration associe un type avec un nom de variable.
- ▶ la définition, **en plus**
 - demande l'allocation mémoire pour la variable,
 - donne une valeur initiale.

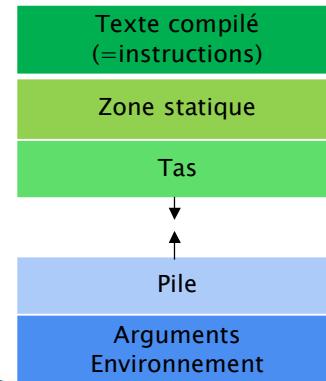
AlgoProg 2015-16 139

Déclaration de variables globales

- ▶ Une déclaration « **promet** » au compilateur qu'il y a une variable ayant ce type et ce nom, définie
 - soit dans un autre fichier source,
 - soit dans une bibliothèque.
- ▶ Les déclarations permettent d'utiliser les variables globales dans plusieurs fichiers.
- ▶ Pour chaque variable, il y a **une** définition et éventuellement plusieurs déclarations.
- ▶ Une déclaration fait référence à une définition dans une autre partie du programme.

AlgoProg 2015-16 140

Schéma conceptuel de la mémoire



- La zone statique et le texte sont **fixés avant le lancement**
- Les arguments sont initialisés au lancement du programme
- La pile et le tas évoluent pendant l'exécution

AlgoProg 2015-16 141

Allocation

- ▶ Allouer une variable, c'est lui **réservé** un emplacement en mémoire.
- ▶ Une définition est en particulier une demande d'allocation.
- ▶ La place d'une variable peut être, selon les cas, réservée dans :
 - la zone statique,
 - la pile,
 - le tas, ...
- ▶ Les **arguments** d'une fonction et les **variables locales non statiques** sont alloués sur la **pile**.
- ▶ Les variables **globales** et celles déclarées **static** sont allouées en **zone statique**.

AlgoProg 2015-16 142

Classe d'allocation

- ▶ La classe d'allocation détermine
 - où en mémoire est conservée la variable,
 - si l'allocation est permanente ou temporaire.
- ▶ une **variable automatique** (locale non statique) est
 - allouée sur la pile d'exécution,
 - seulement pendant que son bloc est actif
- ▶ une **variable globale ou statique**
 - est allouée en zone statique
 - a une existence permanente en mémoire.

AlgoProg 2015-16 143