

Introduction à la Programmation

Benoit Donnet
Année Académique 2023 - 2024



Agenda

- Introduction
- **Chapitre 1: Bloc, Variable, Instruction Simple**
- Chapitre 2: Structures de Contrôle
- Chapitre 3: Méthodologie de Développement
- Chapitre 4: Structures de Données
- Chapitre 5: Modularité du Code
- Chapitre 6: Pointeurs
- Chapitre 7: Allocation Dynamique

Agenda

- Chapitre 1: Bloc, Variable, Instruction Simple
 - Bloc
 - Variable
 - Expression
 - Instruction Simple

Agenda

- Chapitre 1: Bloc, Variable, Instruction Simple
 - Bloc
 - Variable
 - Expression
 - Instruction Simple

Bloc

- La forme la plus simple d'un programme C

```
int main()
```

```
{  
  
}
```

point d'entrée

bloc

- Un Bloc
 - est délimité par { }
 - contient des instructions exécutables

Agenda

- Chapitre 1: Bloc, Variable, Instruction Simple
 - Bloc
 - Variable
 - ✓ Principe
 - ✓ Type Primitif
 - ✓ Pointeur
 - Expression
 - Instructions Simple

Principe

- **Variable mathématique**

- symbole, parfois indexé, représentant une quantité inconnue appartenant à un ensemble donné

- **Propriétés**

1. *généralisation*

- ✓ permet de traduire qu'une propriété est générale

2. *existence*

- ✓ permet d'affirmer l'existence d'un objet sans l'expliciter

3. *résolution*

- ✓ permet d'exprimer un problème sous forme d'équation

Principe (2)

- **Variable informatique**

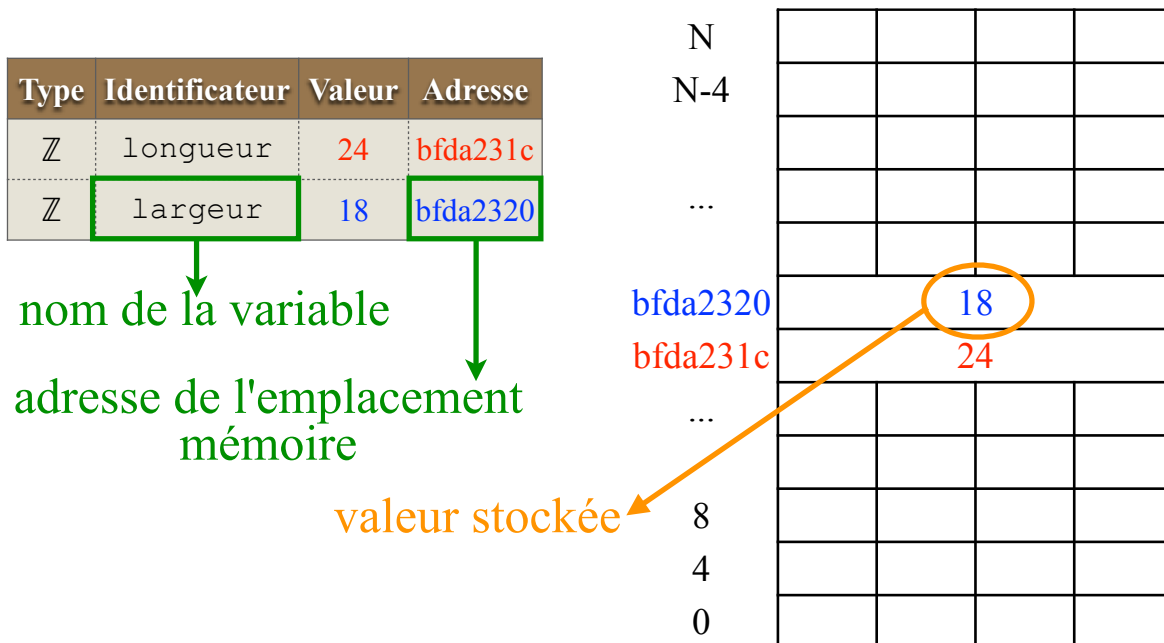
- associe un nom à une valeur appartenant à un ensemble donné
 - ✓ une variable correspond toujours à une valeur
 - ✓ une variable peut subir une opération
 - ✓ la valeur d'une variable peut être modifiée par une opération

- **Caractérisée par**

- un **identificateur** permettant d'y faire référence
 - ✓ composé de lettres (a-z, A-Z, _) et de chiffres
- un **type**
 - ✓ ensemble des valeurs possibles de la variable déclarée
 - ✓ désigne la nature du contenu de la variable
 - ✓ désigne les opérations pouvant être effectuées dessus
 - ✓ lorsqu'une variable est déclarée, la place mémoire correspondant au type est associée à l'identificateur

Principe (3)

- En pratique, une variable correspond à un emplacement mémoire



Type Primitif

- En C, il existe 4 types primitifs

Nom	Domaine	Exemple
char	caractère	'c', 'a', '@', 'A', '1'
int	sous-ensemble des entiers	1, 10, -10
float	sous-ensemble des réels	1.5, -2.24
double	sous-ensemble des réels	9 564

Type Primitif (2)

- Un char
 - est stocké sur 1 byte
- Un caractère est codé/représenté par un chiffre
 - 'A', 'B', ..., 'Z' \Rightarrow 65, 66, .. 90
 - 'a', 'b', ..., 'z' \Rightarrow 97, 98, ..., 122
 - **table ASCII**
 - ✓ cfr. slide suivant
- Peut être signé ou non
 - signed char \Rightarrow [-128, 127]
 - unsigned char \Rightarrow [0, 255]
 - dépendant de l'architecture

Type Primitif (3)

- Table de correspondance ASCII

char	entier
'A'	65
'B'	66
'C'	67
...	
'Z'	90

char	entier
'a'	97
'b'	98
'c'	99
...	
'z'	122

Type Primitif (4)

- un `int`
 - représente un nombre entier
 - stocké sur 32 bits
 - ✓ 4 bytes
- Le type `int` peut être modifié
 - on peut lui donner un signe (ou non)
 - ✓ `unsigned int` $\Rightarrow [0, 2^{32}-1]$ ($\approx \mathbb{N}$)
 - ✓ `signed int` $\Rightarrow [-2^{31}, 2^{31}-1]$ ($\approx \mathbb{Z}$)
 - on peut modifier la taille d'un `int`
 - ✓ entier court (2 bytes)
 - `short int`
 - ✓ entier long (8 bytes)
 - `long int`
 - ✓ combinaison possible avec `signed/unsigned`
 - exemple: `unsigned long int`

Type Primitif (5)

- `float` et `double` permettent de représenter des nombres décimaux ($\approx \mathbb{R}$) avec une certaine précision
 - 4 bytes (`float`)
 - ✓ les 6 premiers chiffres décimaux sont corrects, le 7^{ème} est arrondi
 - 8 bytes (`double`)
 - ✓ les 15 premiers chiffres décimaux sont corrects, le 16^{ème} est arrondi
- Pas de modification possible
 - pas de signe
 - ✓ `float` et `double` sont d'office signés
 - pas de modification de la taille
 - ✓ exception: `long double`

Pointeur

- Il est possible de mémoriser, dans un emplacement mémoire, l'adresse d'une variable

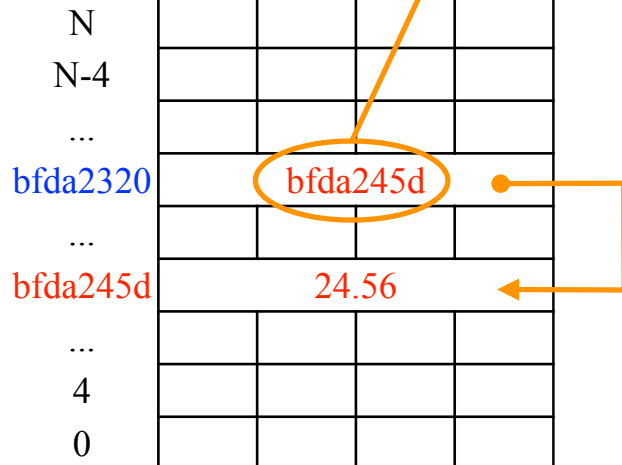
- **Pointeur**

- variable dont la valeur est une adresse

Adresse d'un autre emplacement mémoire

Type	Identificateur	Valeur	Adresse
float *	longueur	bfda245d	bfda2320

pointeur vers
un type float



Agenda

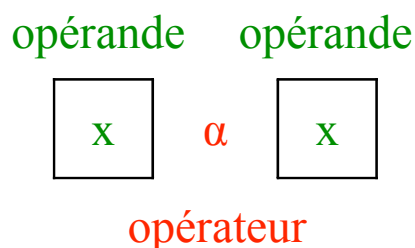
- Chapitre 1: Bloc, Variable, Instruction Simple
 - Bloc
 - Variable
 - Expression
 - ✓ Définition
 - ✓ Opérateurs
 - ✓ Priorité des Opérateurs
 - Instruction Simple

Définition

- Description du calcul d'une valeur
- Le résultat du calcul est une valeur ayant un certain type
- Une **expression** peut être
 1. une variable, dénotée par son identificateur
 - ✓ l'évaluation retourne la valeur courante de la variable
 - ✓ exemple: `moyennePoints`
 2. une constante (ou littéral)
 - ✓ exemple: `'a'`, `3.14159265`
 3. obtenue par l'application d'opérateurs à d'autres expressions
 - ✓ exemple: `(4.0 / 3.0) * pi * r * r * r`

Opérateurs

- Un **opérateur** permet d'évaluer une expression bien définie sur des valeurs (**opérandes**) en produisant un résultat (**valeur** de l'expression)



Opérateurs (2)

- Les opérateurs
 - **unaires** requièrent une unique opérande
 - ✓ exemple: -5
 - **binaires** en requièrent deux
 - ✓ exemple: $3 + x$
 - **ternaires** en requièrent trois
 - ✓ non abordés dans le cours

Opérateurs (3)

- Il existe différents opérateurs
 - opérateurs *arithmétiques*
 - opérateurs de *comparaison*
 - opérateurs *booléens*
 - opérateurs d'*affectation*
 - opérateurs d'*incrémentation/décrémentation*
 - opérateurs *bit-à-bit*
 - opérateurs de *décalage*
 - opérateurs sur les *pointeurs*

Opérateurs (4)

- Opérateurs **arithmétiques**
- La nature de l'opération diffère selon le type des opérandes
 - $2 / 3 = 0$
 - $2.0 / 3.0 = 0.666666\dots$
- modulo applicable seulement aux `int`
- priorité des opérateurs identique à l'algèbre
 - $a * a + b * b$
 - $(a * a) + (b * b)$
- la forme unaire existe
 - -1

Op.	Signification
+	addition
-	soustraction
/	division
*	multiplication
%	modulo

Opérateurs (5)

- Opérateurs de **comparaison**
 - comparaison de deux valeurs
- Retournent des valeurs booléennes
 - vrai ou faux
 - en C standard, il n'existe pas de type booléen
 - ✓ *vrai* \Rightarrow valeur entière non nulle
 - ♦ 1
 - ✓ *faux* \Rightarrow valeur entière nulle
 - ♦ 0

Op.	Signification
<	+ petit que
>	+ grand que
<=	+ petit ou égal
>=	+ grand ou égal
==	égal
!=	différent

Opérateurs (6)

- Opérateurs **booléens**
 - opérations logiques

Op.	Signification
&&	et “lazy”
	ou “lazy”
!	négation

A	B	A && B	A B	!A
V	V	V	V	F
V	F	F	V	F
F	V	F	V	V
F	F	F	F	V

Opérateurs (7)

- Opérateurs booléens (suite)
 - comment est évaluée l'expression suivante?
 - ✓ $n \neq 0 \ \&\& \ m/n > 1$
 - écrire la table de vérité pour
 - ✓ $A \ \&\& \ (B \ || \ C)$
 - ✓ $(A \ \&\& \ !B) \ || \ (!A \ \&\& \ B)$

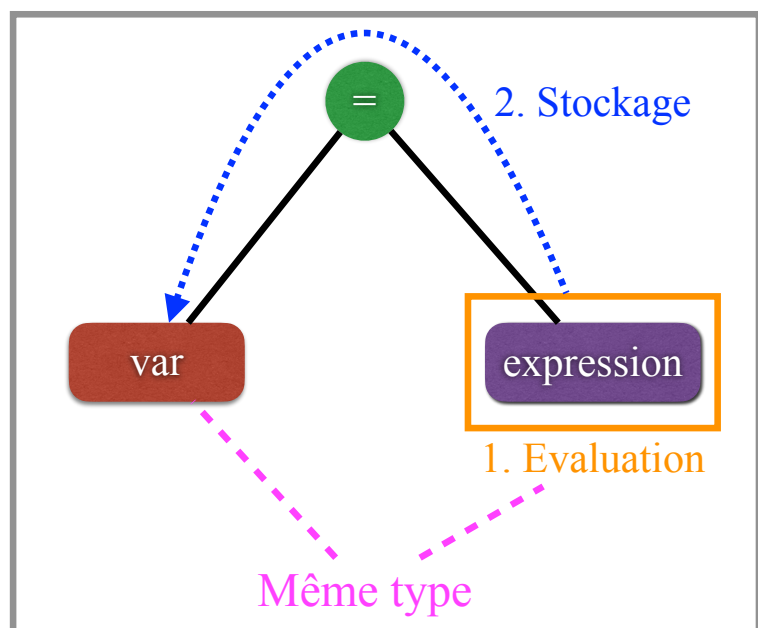
Opérateurs (8)

- Opérateur d'**affectation**
 - =
 - `var = expr`
 - affecter une valeur (`expr`) à une variable (`var`)
 - ✓ l'ancienne valeur de `var` est écrasée/remplacée par `expr`
 - la valeur à affecter peut être le résultat d'une expression
 - le type du résultat de `expr` doit être identique à celui de `var`
- Fonctionnement
 - membre de gauche, `var`, contient l'identificateur de la variable (**valeur à gauche**) qui va accueillir le résultat
 - évaluation du membre de droite, `expr`, fournit la nouvelle valeur à attribuer à `var`
 - après l'affectation, l'expression entière devient égale à la valeur affectée

Opérateurs (9)

- Opérateur d'affectation (suite)
 - illustration du fonctionnement

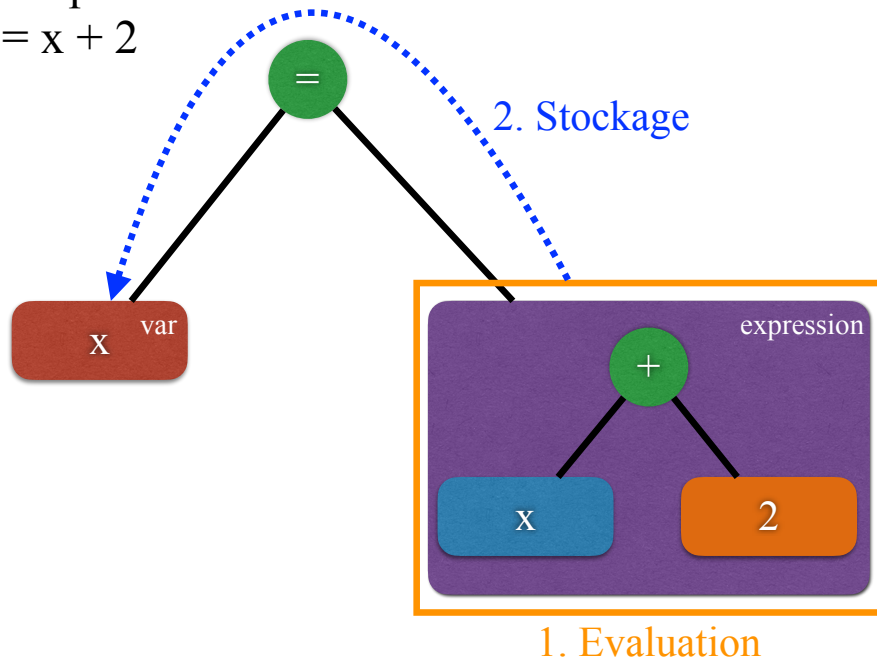
3. Toute l'expression prend la nouvelle valeur de `var`



Opérateurs (10)

- Opérateur d'affectation (suite)

- exemple
- $x = x + 2$



Opérateurs (11)

- Opérateur d'affectation (suite)

- exemple (suite)

$x = x + 2$

Type	Identificateur	Adresse	Valeur
int	x	bfda2320	6

1. Aller lire la valeur de x en mémoire (4)
2. Additionner la valeur de x (4) avec le littéral (2)
3. Placer le résultat de l'addition (6) dans x

N				
N-4				
...				
bfda2320			6	
...				
4				
0				

Opérateurs (12)

- Il est possible de combiner, en un seul opérateur, l'affectation et l'opération arithmétique
 - **sucré syntaxique**
 - ✓ raccourci d'écriture
- Forme
 - `var α = expr`
 - ✓ où $\alpha \in \{+, -, *, /, \%\}$
- Equivalent à `var = var α expr`
- Exemple

`x += 2`

Opérateurs (13)

- Opérateurs d'**incrément/décrément**
 - opérateurs unaires
 - ✓ ++, --
 - opérande est une valeur à gauche
- L'opérateur peut être placé
 - à *droite* de l'opérande
 - ✓ variable incrémentée (++) ou décrémentée (--) d'une unité
 - ✓ la valeur de l'expression correspond à celle de la variable avant l'opération
 - à *gauche* de l'opérande
 - ✓ idem mais la valeur de l'expression correspond à celle après l'opération

Opérateurs (14)

- Exemples
 - Soient x et y , deux variables entières, initialisées à la valeur 0, évaluer
 - ✓ $x = y++$
 - $x?$
 - $y?$
 - ✓ $x = ++y$
 - $x?$
 - $y?$
- Attention
 - lisibilité!
 - ✓ $x = --x + x++$
 - ✓ l'évaluation de cette expression n'est pas garantie par le standard
 - combinaison avec des opérateurs booléens
 - ✓ $x > 0 \ \&\& \ x--$

Opérateurs (15)

- Opérateurs **bit-à-bit**
 - effectue une opération (logique) sur chacun des bits des deux opérandes
- Exemples

Op.	Signification
&	et
	ou
^	ou exclusif

$9 \ \& \ 12$	$9 \ \ 12$	$9 \ ^ \ 12$
$\begin{array}{r} 1001 \\ \& \ 1100 \\ \hline 1000 \end{array}$	$\begin{array}{r} 1001 \\ \ 1100 \\ \hline 1101 \end{array}$	$\begin{array}{r} 1001 \\ ^ \ 1100 \\ \hline 0101 \end{array}$

Opérateurs (16)

- Opérateurs de **décalage**
 - décalage de x bits vers la gauche/droite
 - ce n'est pas une rotation!
- Fonctionnement
 - $\text{var} \propto x$
 - ✓ var , la valeur initiale
 - ✓ $\propto \in \{<<, >>\}$
 - ✓ x , nombre de pas de décalage (> 0)
- Exemples

Op.	Signification
$<<$	décalage à gauche
$>>$	décalage à droite avec conservation du signe

$$\begin{array}{rcl}
 6 << 1 & & 6 >> 1 \\
 \hline
 00000110 << 1 & & 00000110 >> 1 \\
 00001100 & & 00000011
 \end{array}$$

Opérateurs (17)

- Opérateurs de décalage (suite)
- Exemples (suite)
 - en signé

$$\begin{array}{rcl}
 -6 << 1 & & -6 >> 1 \\
 \hline
 \textcolor{red}{1}1111010 << 1 & & 1111101\textcolor{red}{0} >> 1 \\
 1111010\textcolor{red}{0} & & \textcolor{red}{1}1111101
 \end{array}$$

Opérateurs (18)

- On peut faire 2 opérations sur les pointeurs

- **référencement**

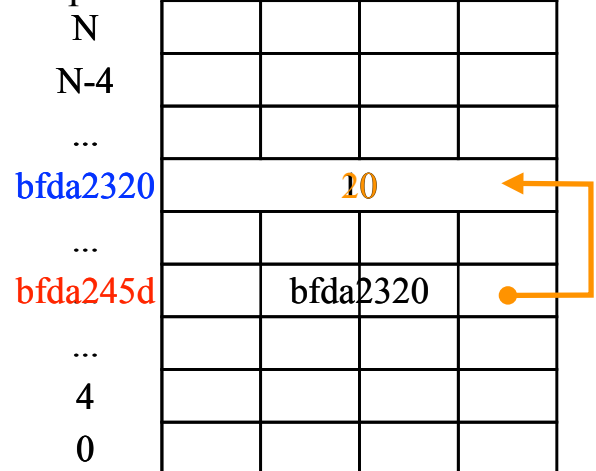
- ✓ obtenir l'adresse d'une variable
- ✓ opérateur: &

- **déréférencement**


- ✓ obtenir la valeur vers laquelle on pointe
- ✓ opérateur: *

```
var = 10
ptr = &var
*ptr = 20
```

Type	Identificateur	Adresse	Valeur
int	var	bfd2320	20
int *	ptr	bfd245d	bfd2320



Priorité des Opérateurs

Priorité	Opérateurs	Sens
+++++	() [] . ->	→
	! -- ++ - * &	←
	* / %	→
	+ -	→
	<< >>	→
	< <= > >=	→
	== !=	→
	&	→
	^	→
		→
	&&	→
	= += -= ...	←

Priorité des Opérateurs (2)

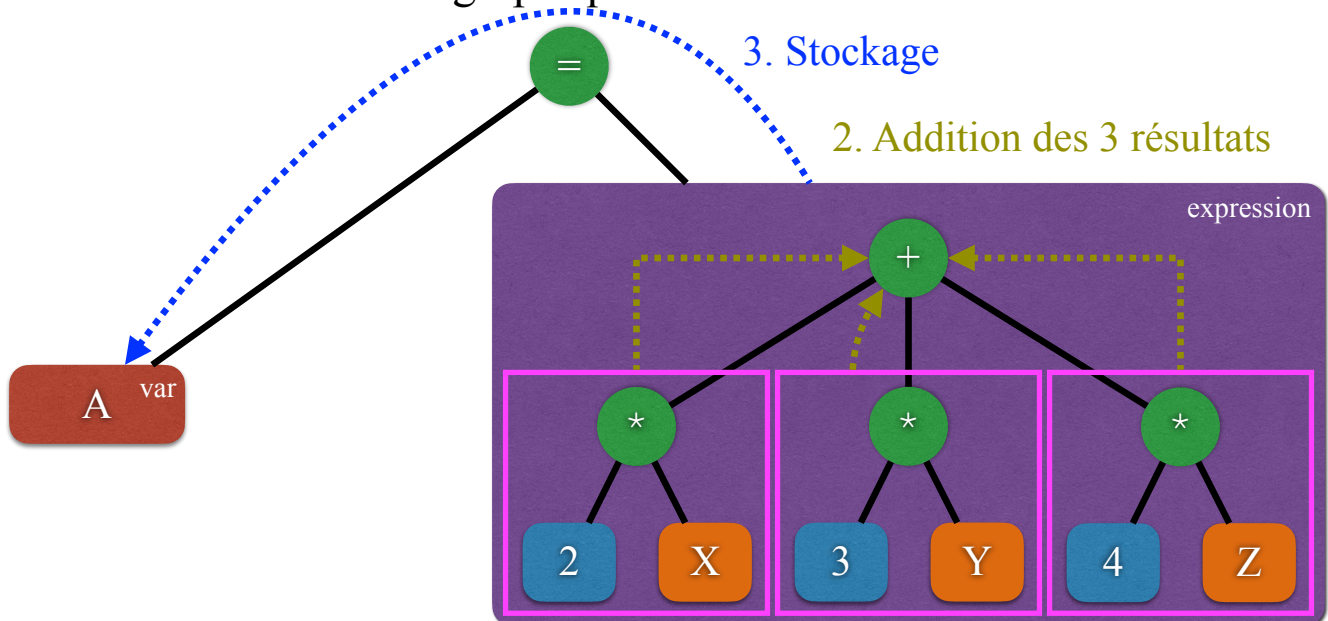
- Soient
 - $X=5$
 - $Y=10$
 - $Z=1$
- Exemple 1

$A = 2 * X + 3 * Y + 4 * Z$

1. Opérateur $*$ prioritaire sur $+$
2. Ordinateur effectue $2 * X$, $3 * Y$ et $4 * Z$
3. Ordinateur effectue l'addition des 3 résultats
4. affecte le résultat à la variable A

Priorité des Opérateurs (3)

- Exemple 1 (suite)
 - raisonnement graphique



Priorité des Opérateurs (4)

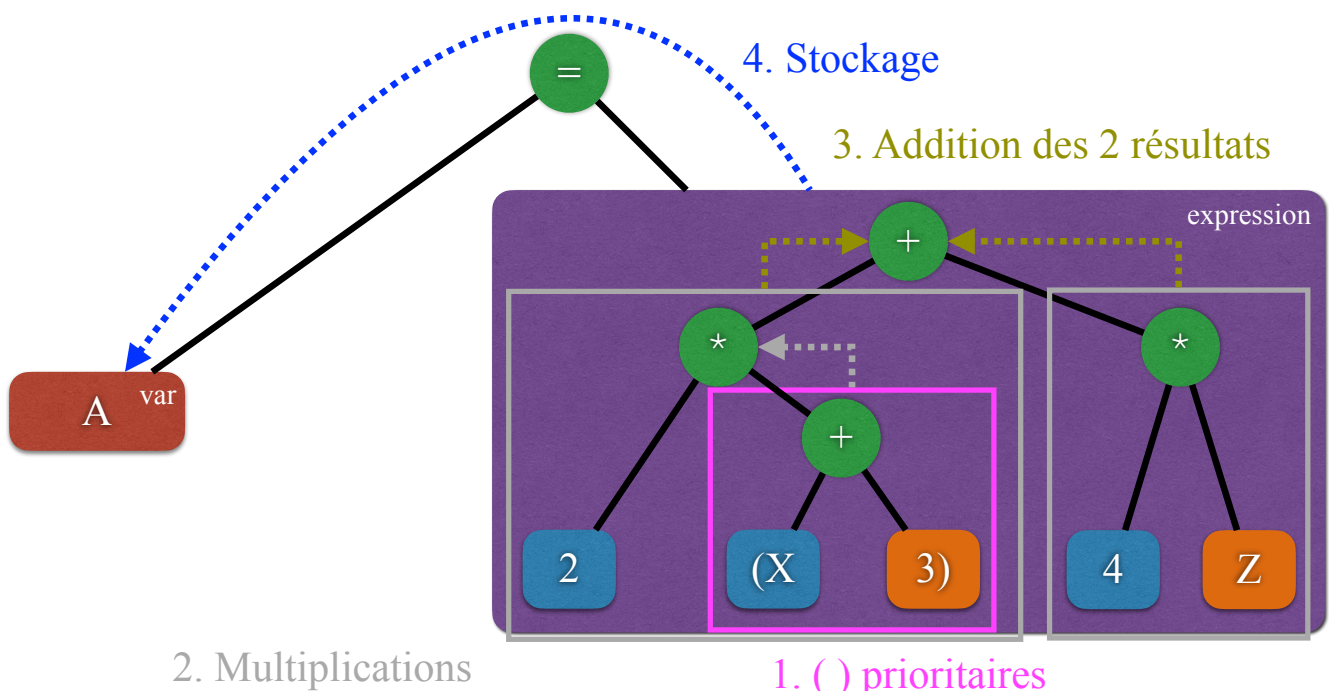
- Exemple 2

$A = 2 * (X + 3) * Y + 4 * Z$

1. () prioritaires
2. l'ordinateur effectue $X + 3$
3. l'ordinateur effectue $2 * 8 * Y$ et $4 * Z$
4. l'ordinateur effectue l'addition des deux résultats
5. l'ordinateur affecte le résultat à A

Priorité des Opérateurs (5)

- Exemple 2 (suite)



Priorité des Opérateurs (6)

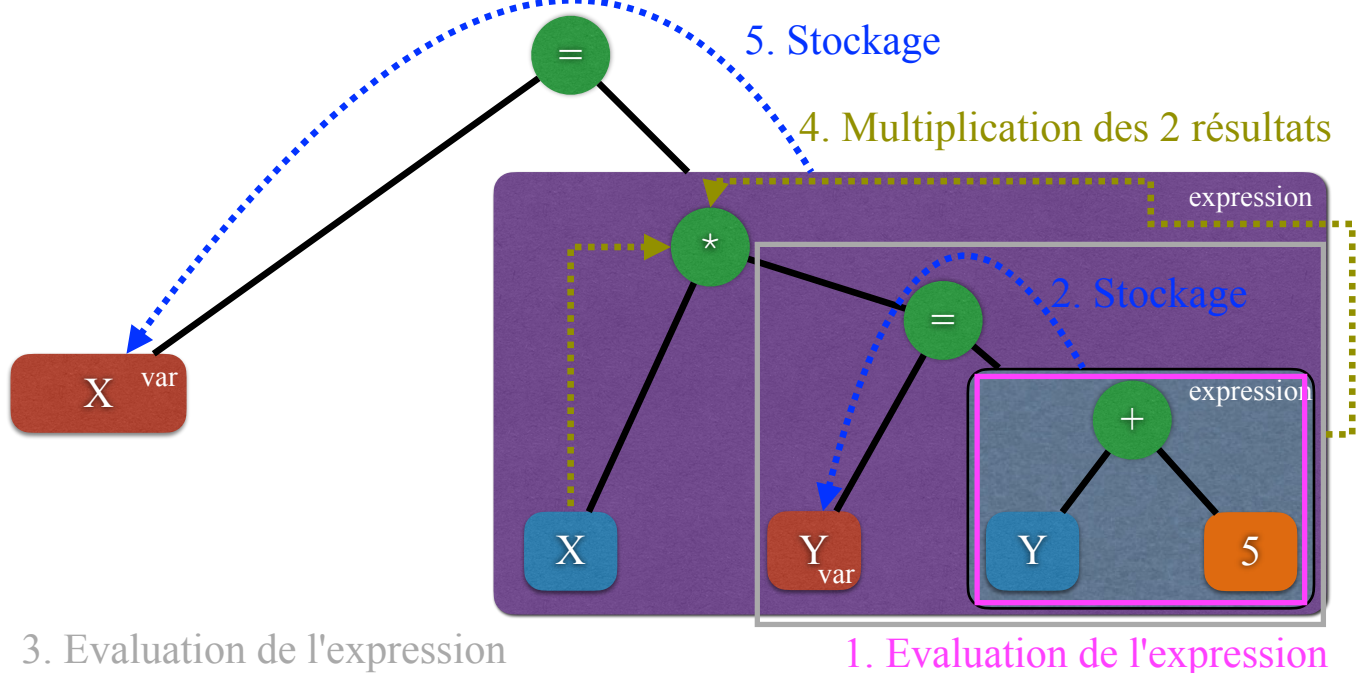
- Soient
 - $X = 3$
 - $Y = 4$
- Exemple 3

$X *= Y += 5$

1. $*=$ a la priorité la plus faible
2. l'ordinateur effectue $Y += 5$
3. l'ordinateur effectue $X *= 9$
4. l'ordinateur affecte le résultat à X

Opérateurs (7)

- Exemple 3 (suite)
 - $X = X * (Y = Y + 5)$



Priorité des Opérateurs (8)

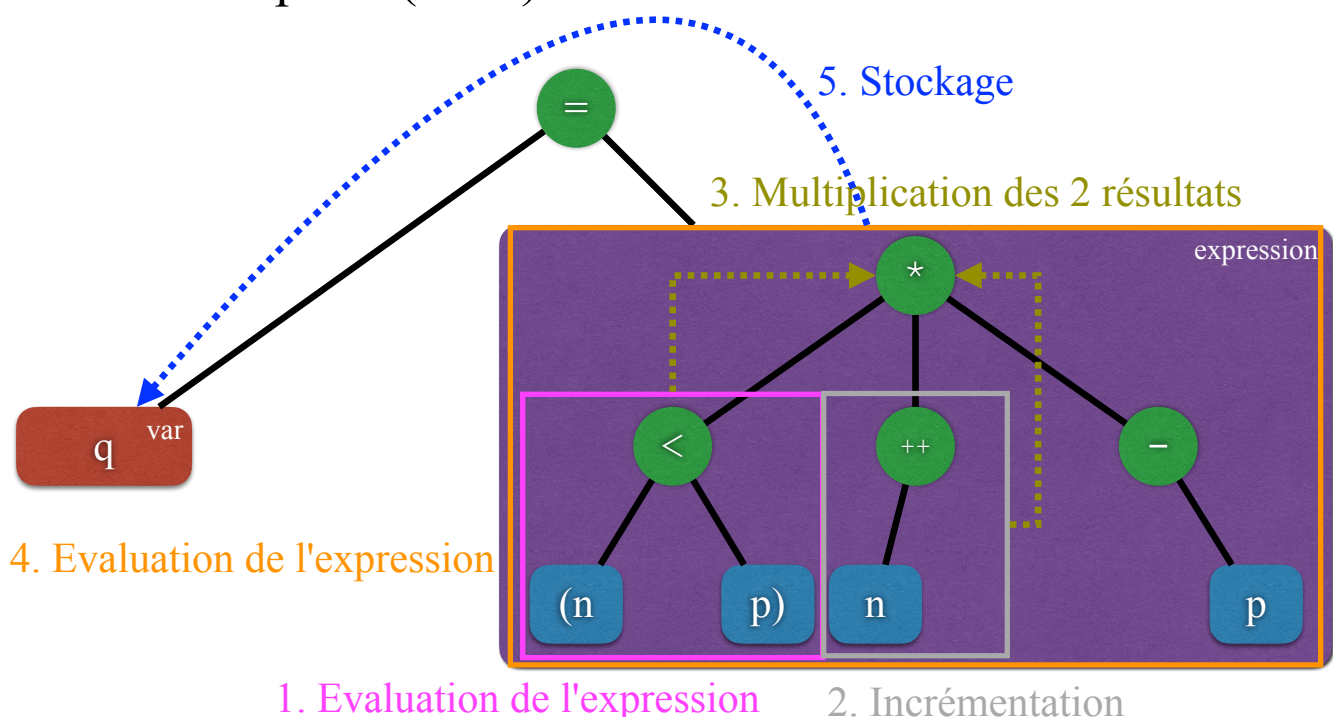
- Soient
 - $n = 5$
 - $p = 5$
- Exemple 4

$q = (n < p) * n++ - p$

1. $(n < p)$ a la plus grande priorité
2. l'ordinateur $n++$
3. l'ordinateur effectue $0 * 5 - 5$
4. l'ordinateur affecte le résultat à q

Priorité des Opérateurs (9)

- Exemple 4 (suite)



Priorité des Opérateurs (8)

- Attention

- $X *= Y+1$ équivaut à $X = X * (Y+1)$
- $X *= Y+1$ n'équivaut pas à $X = X * Y + 1$

Agenda

- Chapitre 1: Bloc, Variable, Instruction Simple
 - Bloc
 - Variable
 - Expression
 - Instruction Simple
 - ✓ Principe
 - ✓ Déclaration
 - ✓ Manipulation des Variables
 - ✓ Entrées/Sorties
 - ✓ Type Cast
 - ✓ Commentaires

Principe

- **Instruction?**
 - étape dans un programme informatique
- Dicte à l'ordinateur l'action nécessaire à effectuer avant de passer à l'instruction suivante
- Un programme est constitué d'une suite d'instructions
- Les instructions sont exécutées les unes après les autres
 - *exécution séquentielle*
- Comment séparer 2 instructions?
 - ;

Principe (2)

- Il existe différents types d'instructions
 - déclaration
 - calcul/manipulation de variables
 - entrées/sorties
 - contrôle
 - ✓ cfr. Chap. 2
 - procédure/fonction
 - ✓ cfr. Chap. 6

Déclaration

- Comment savoir de quel type est une variable?
 - il faut déclarer la variable et son type
- Une **déclaration de variables**
 - est une instruction
 - qui définit l'identificateur et le type d'une variable donnée
- Les variables doivent toujours être déclarées avant d'être utilisées
 - ne peut se faire que dans le bloc d'instructions
 - ✓ pas toujours vrai (cfr. Chap. 6)

Déclaration (2)

- Format d'une déclaration

Optionnel

```
type de la variable  nom de la variable  initialisation
[const] type  identificateur [= valeur]
constante      [, identificateur [= valeur]]
               ...
               [, identificateur [= valeur]];
               indicateur de la fin d'une instruction
```

Déclaration (3)

- Exemple

```
int main(){
    int i;
    int j = 5, k = 2, z;
    int *p;
    unsigned long codeBarre;
    char symbole = 'a', eol='\n';
    const double MOYENNE = 15.56;
} //fin programme
```

- Attention à l'identificateur
 - il doit décrire clairement l'usage de la variable
 - cohérence

Déclaration (3)

- Ce qu'il ne faut pas faire avec les noms de variables

Mauvaise Déclaration	Raison
<code>int Quantité;</code>	pas d'accents
<code>int Prix Hors Taxe;</code>	pas d'espace entre les mots
<code>int a, b, c, 1P45;</code>	signification?

Déclaration (4)

- Un programme aura donc la forme suivante

```
int main(){  
    //Déclaration des variables  
  
    //Instructions de calcul/manipulation des variables  
} //fin programme
```

Manipulation Variables

- Déclarer et initialiser une variable, c'est bien
- Savoir l'utiliser dans le programme pour des calculs, c'est mieux
- On va utiliser des expressions dans des instructions pour effectuer des calculs sur les variables déclarées
 - l'affectation nous permet de conserver les résultats intermédiaires

Manipulation Variables (2)

- Exemple
 - calcul de l'aire d'un cercle
 - ✓ $\text{aire} = \pi \times r^2$

```
int main(){
    //Déclaration des variables
    const double PI = 3.1415;
    double rayon = 4.0;
    double aire = 0.0;

    //Instructions de calcul/manipulation des variables
    aire = PI * (rayon * rayon);
} //fin programme
```

instruction

expression

expression

Entrées/Sorties

- Rappel: un programme prend des données en entrée et produit un résultat en sortie
- les données peuvent être saisies sur l'entrée standard
 - clavier
- les résultats peuvent être écrits sur la sortie standard
 - écran

Entrées/Sorties (2)

- Comment écrire un message à l'écran?
- 2 étapes
 1. inclure la librairie `stdio.h`
 2. utiliser l'instruction `printf(message) ;`
 - ✓ `message` est une chaîne de caractères entre guillemets
- Exemple

```
#include <stdio.h>

int main(){
    printf("I'm on the highway to hell!");
} //fin programme
```

Entrées/Sorties (3)

- On peut rajouter au message à afficher des caractères spéciaux de formatage
 - `\n` => retour à la ligne
 - `\t` => tabulation horizontale
 - `\v` => tabulation verticale
- Exemple

```
#include <stdio.h>

int main(){
    printf("I'm on the highway to hell!\n");
} //fin programme
```

Entrées/Sorties (4)

- Comment écrire un message à l'écran qui affiche le contenu d'une variable?
- 2 étapes
 1. inclure la librairie `stdio.h`
 2. utiliser l'instruction `printf(message, expression);`
 - ✓ `message` est une chaîne de caractères entre guillemets avec un **formatage**
 - ✓ `expression` est la liste des expressions qu'on veut afficher
 - chaque expression est séparée par une virgule

Entrées/Sorties (5)

- Exemple 1

```
#include <stdio.h>

int main(){
    int x = 5;
    float y = 5.4;
    char c = 'a';

    printf("La variable x vaut %d\n", x);
    printf("La variable y vaut %f\n", y);
    printf("La variable c vaut %c\n", c);
    printf("Les 3 variables valent: %d %f %c\n", x, y, c);
} //fin programme
```

Entrées/Sorties (6)

- Les Entrées/Sorties nécessitent un formatage en fonction du type primitif

Type Primitif	Formatage	Exemple
int	%d	int i=-10; printf("%d", i);
float	%f	float x=2.0/3.0; printf("%f", x);
double	%lf	double x=2.0/3.0; printf("%lf", x);
char	%c	char car='a'; printf("%c", car);
long	%ld	long int li=1000; printf("%ld",li);
short	%hd	short si=1; printf("%hd", si);
unsigned int	%u	unsigned int ui = 10; printf("%u", ui);
unsigned long	%lu	unsigned long ul = 10; printf("%lu", ul);
unsigned short	%hu	unsigned short us = 10; printf("%hu", us);

Entrées/Sorties (7)

- Exemple 2
 - calcul de l'aire d'un cercle

```
#include <stdio.h>

int main(){
    //Déclaration des variables
    const double PI = 3.1415;
    double rayon = 4.0;
    double aire = 0.0;

    //Instructions de calcul/manipulation des variables
    aire = PI * (rayon * rayon);

    printf("L'aire du cercle est: %lf\n", aire);
} //fin programme
```

Entrées/Sorties (8)

- Comment faire pour lire une donnée au clavier?
- 2 étapes
 1. inclure la librairie `stdio.h`
 2. utiliser l'instruction `scanf(format, variable);`
 - ✓ `format` est une chaîne de caractères, entre guillemets, qui indique le formatage (cfr. slide 61)
 - ✓ `variable` est la variable, précédée du caractère `&`, qui va contenir ce qui est lu au clavier
 - possibilité de lire plusieurs variables d'un seul coup

Entrées/Sorties (9)

- Exemple 1

```
#include <stdio.h>

int main(){
    int x;
    float y;

    scanf("%d", &x);
    scanf("%f", &y);
    printf("Les 2 variables valent: %d %f\n", x, y);
    scanf("%d %f", &x, &y);
    printf("Les 2 variables valent: %d %f\n", x, y);
} //fin programme
```


Entrées/Sorties (10)

- Exemple 2
 - calcul de l'aire d'un cercle

```
#include <stdio.h>

int main(){
    //Déclaration des variables
    const double PI = 3.1415;
    double rayon;
    double aire = 0.0;

    //Instructions de calcul/manipulation des variables
    printf("Entrez une valeur pour le rayon: ");
    scanf("%lf", &rayon);

    aire = PI * (rayon * rayon);

    printf("L'aire du cercle est: %lf\n", aire);
} //fin programme
```

Type Cast

- Possibilité d'affecter à une variable de type t_1 une valeur d'un autre type t_2
 - une conversion de la valeur vers le type t_1 est alors effectuée
 - **type cast**

Type Cast (2)

- Il existe deux types de type cast

1. casting implicite

```
int main(){  
    double p = 3.1416;  
    int x;
```

```
    x = p;  
} //fin programme
```

x prend la valeur 3

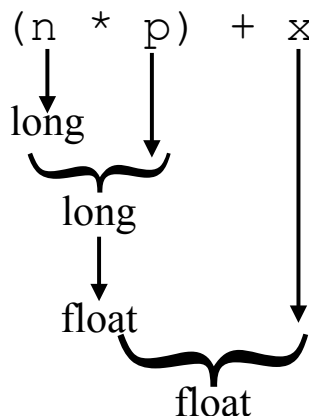
- Règles de conversion
 - ✓ *promotion entière*
 - les types plus petits ou égaux à int (i.e., char et short) sont automatiquement convertis en int avant toute opération
 - ✓ opérandes de types différents?
 - le type faible est converti dans le type fort
 - hiérarchie de type (int < long < float < double < long double)
 - ✓ perte de précision

Type Cast (3)

- Casting implicite

- exemple

```
int n;  
long p;  
float x;
```



Type Cast (4)

- Il existe deux types de type cast
 1. casting implicite
 2. casting explicite
 - format

(type) expression

- la valeur est celle de expression convertie en type

```
int main(){  
    double a = 2.0;  
    double b = 3.0, x;  
    int y;  
    x = a/b;  
    y = (int)a/(int)b;  
} //fin programme
```

x prend la valeur 0.66666...
y prend la valeur 0

Type Cast (5)

- Casting explicite
 - exemples

```
int n=7, p=2;
```

type et valeur des expressions suivantes?

(double) n

type: ???

valeur: ???

(double) (n/p)

type: ???

valeur: ???

(double) n/p

type: ???

valeur: ???

Commentaires

- Un programme n'est pas uniquement destiné à être exécuté
 - il doit être tenu à jour par les programmeurs
 - il peut être modifié au cours du temps, pour répondre à de nouvelles exigences
- Les commentaires permettent de rendre un programme lisible
 - `//`
 - ✓ commente une ligne
 - `/* */`
 - ✓ commente tout ce qui se trouve entre `/*` et `*/`

Commentaires (2)

- Exemple

```
int main(){  
    //je commente une seule ligne  
  
    /*  
    je commente plusieurs lignes  
    en une seule fois  
    */  
} //fin programme
```

Exercices

- Transformer une température en degré Fahrenheit (saisie au clavier) en degré Celsius et afficher la transformation à l'écran
 - $C \sim 0.55556 \times (F - 32)$
- Soit la fonction f définie par $f(x) = (2x+3)(3x^2+2)$
 - calculer et afficher l'image par f d'un nombre saisi au clavier
 - calcul et afficher une approximation de la dérivée f' de f en un point x (entré au clavier), pour un h assez petit (saisi au clavier)

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$