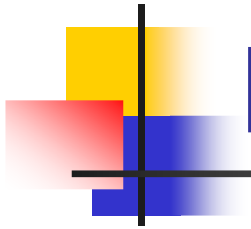
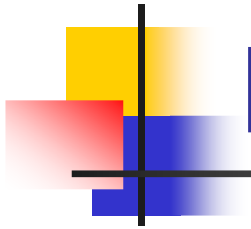
Environnements d'exécution

ESI – École nationale Supérieure en Informatique



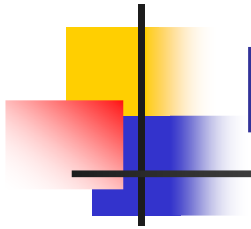
Environnement d'exécution

- Comment va s'exécuter un programme
- Comment est implémenté l'appel de procédure
- Relations qui existent entre noms et données en mémoire



Noms

- Les noms utilisés dans un programme ne servent qu'à identifier les objets associés à l'exécution (variables, tableaux, procédures, ...).
- Les noms sont arbitraires dans le sens ou chaque programmeur peut utiliser les noms qu'il veut (club de foot, noms de fleurs, …) pour nommer ses objets.



Noms

- Le programme objet, généré à l'issue de la phase de compilation :
  - Les noms ne persistent pas dans les exécutables machines dépendants.
  - Les noms persistent dans les programmes objets dits machine-indépendants

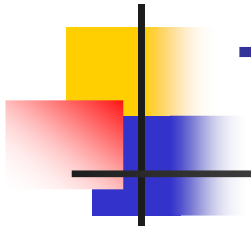


Table des symboles

- Pendant l'analyse du programme source le compilateur collecte et utilise les informations liées aux noms qui y apparaissent (*bindings*).
- Ces informations sont stockées dans une structure de données appelée Tables des Symboles ou Environnement.
- Cette table sera "vivante" durant toute la phase de compilation.

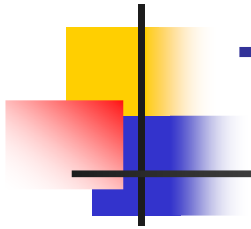


Table des symboles

- Les informations concernant un nom apparaissant dans un programme source sont typiquement :
  - Chaîne de caractères composant le nom
  - Son type (entier, réel, …)
  - Sa forme (variable simple, un tableau, une structure, ...)
  - Son adresse et sa taille mémoire à l'exécution

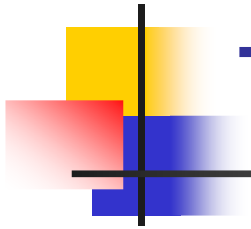


Table des symboles

- La table des symboles associe également :
  - Si la variable est un tableau, la taille du tableau est également sauvegardée.
  - Pour chaque nom de constante son type et sa valeur.
  - Pour chaque fonction et procédure, la liste des paramètres formels et leurs définitions et son type de retour.

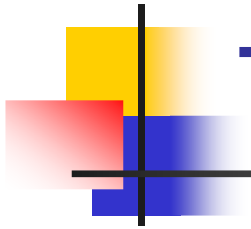


Table des symboles

- Chaque entrée dans la table des symboles est référencée par le couple : **(nom, informations).** La table des symboles peut être structurée de différentes manières.

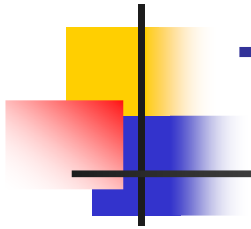


Table des symboles

- Les opérations typiques sur la table des symboles sont :
  - Déterminer si un nom est dans la table des symboles (*lookup*).
  - Ajouter un nom est dans la table des symboles.
  - Accéder aux informations relatives à un nom.
  - Ajouter de nouvelles informations relatives à un nom.
  - Entrer dans une portée.
  - Sortir d'une portée.

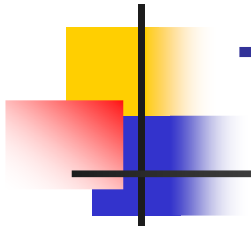


Table des symboles

- **Exemple :**
- Considérer le programme suivant :

```
program
real x; integer ajout;
begin
x=3.14; ajout=5;
x:=x+ajout;
write(x) ;
End.
```

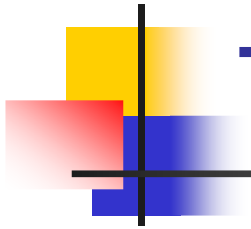
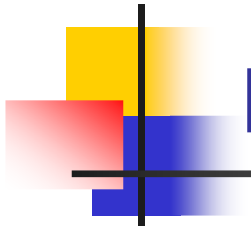


Table des symboles

- La table des symboles contiendra les informations suivantes :

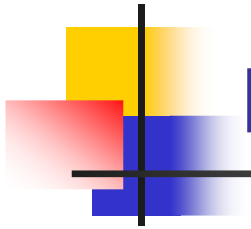
nom	descripteurs
x	variable, real, 4 octets, adresse #1
ajout	variable, integer, 2 octets, adresse #2
write	Procédure, adresse obtenue par le chargeur

Taille d'un exécutable sur mémoire secondaire



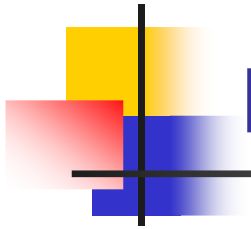
- Un exécutable en mémoire secondaire est constitué
  - des instructions machines spécifiques à ce programme
  - et des données statiques allouées dans ce programme.

Taille d'un exécutable sur mémoire secondaire



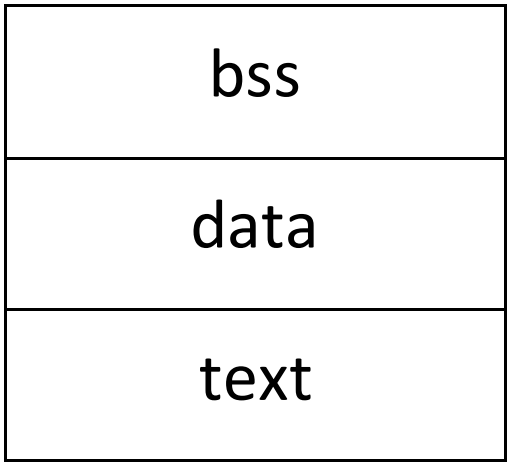
- La partie instructions est souvent désignée sous le vocable **text**.
- Les données statiques initialisées sont stockées dans la partie **data** et les données non initialisées sont stockées dans la partie **bss** pour block started from symbol dénomination historique provenant de UA-SAP (United Aircraft Symbolic Assembly Program).

Taille d'un exécutable sur mémoire secondaire

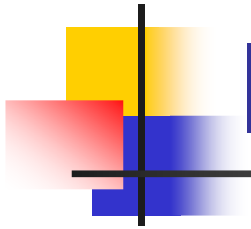


- Souvent les parties data et bss sont désignées uniquement par le terme data.
- La taille des parties text, data et bss est connue à la fin du processus de compilation.

Taille d'un exécutable sur mémoire secondaire

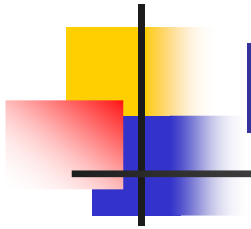


Répartition de la mémoire à l'exécution



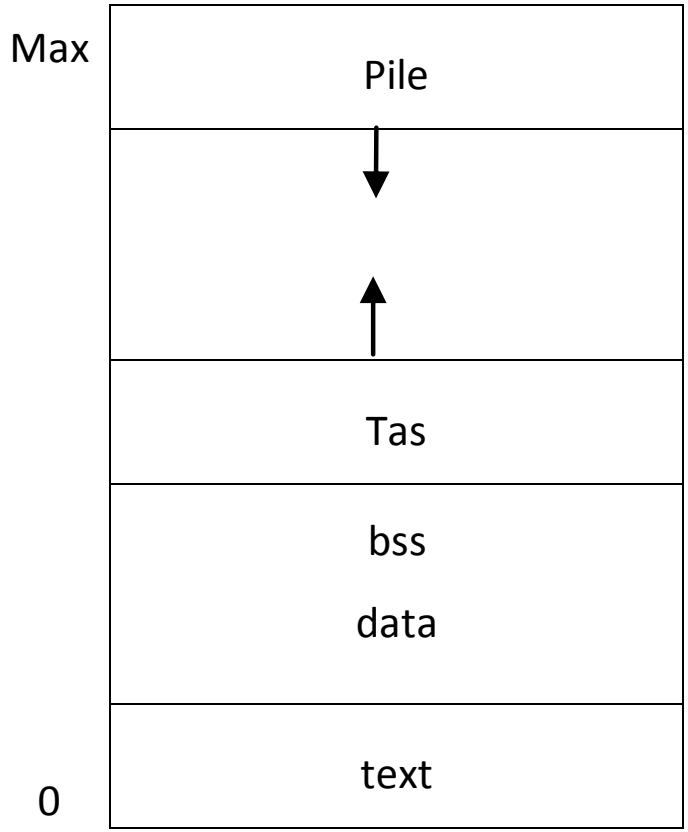
- Lors du lancement d'un exécutable, une mémoire virtuelle est allouée à ce programme
- L'espace d'adressage de cette mémoire virtuelle va de 0 à MAX (MAX est égal à 512GO avec le système d'exploitation Linux).

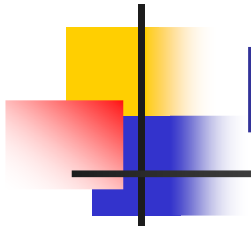
Répartition de la mémoire à l'exécution



- Par convention, la partie basse de cette zone mémoire est occupé par la partie text (instructions machine) suivie des parties data (données statiques initialisées) et bss (expansion des données statiques non initialisées).

Répartition de la mémoire à l'exécution



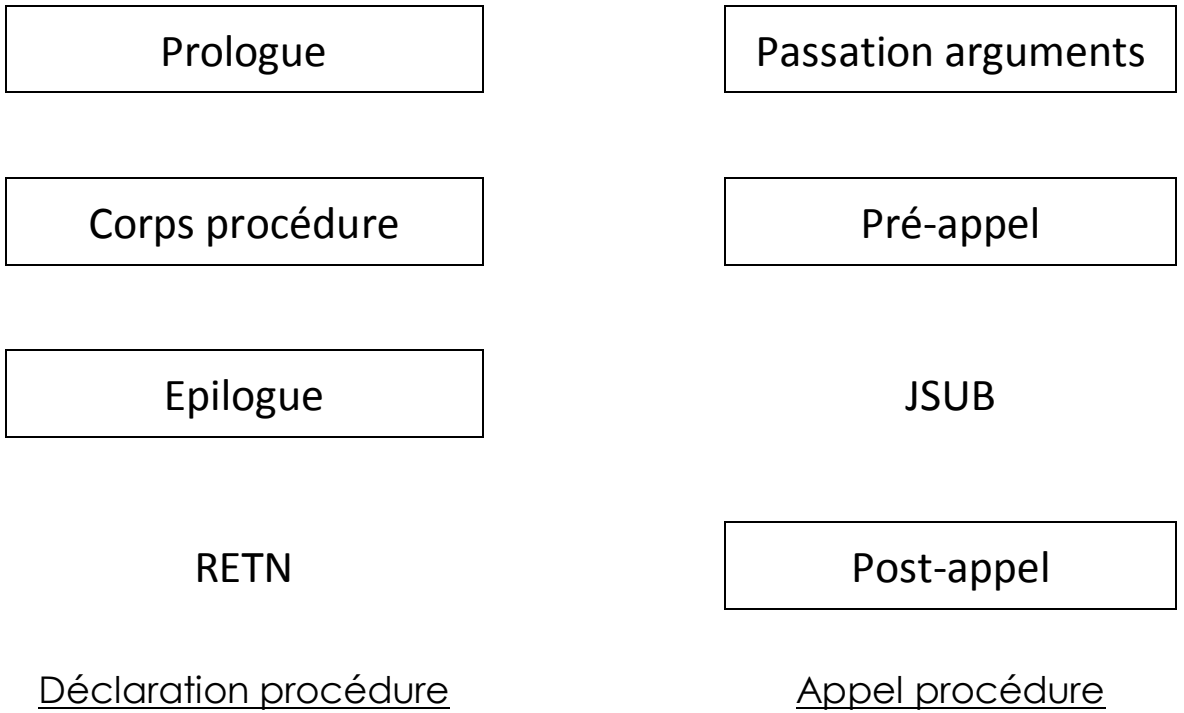


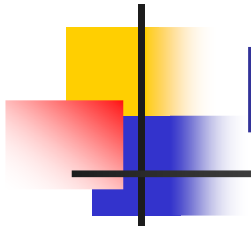
Procédures et activations

- Procédure :
  - Une procédure est définie, statiquement, par une identification et un corps (on parle de corps de procédure).
  - L'identificateur est le nom de la procédure et le corps une suite d'instructions.

Procédures et activations

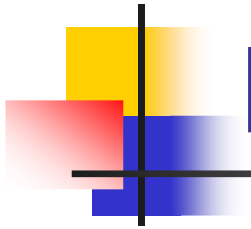
- Nous distinguerons le corps de la procédure et un appel de procédure.





Procédures et activations

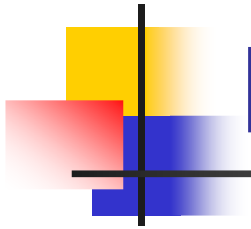
- A la définition d'une procédure correspondra dans la zone de code les instructions : d'un prologue, du corps de la procédure, d'un épilogue et de l'instruction RETN.
- A un appel de procédure correspondra dans la zone de code les instructions relatives à : la passation d'arguments, pré-appel, l'instruction JSUB et post-appel de la procédure.



Procédures et activations

- A chaque appel de procédure, une structure appelée bloc d'activation est crée en pile.
- Ce bloc d'activation sera structuré pour contenir différents paramètres (qui seront détaillés dans la section suivante) nécessaires au bon fonctionnement du programme.
- Au retour de cet appel de procédure, un traitement devra être effectué pour libérer ce bloc d'activation et restaurer l'environnement d'appel.

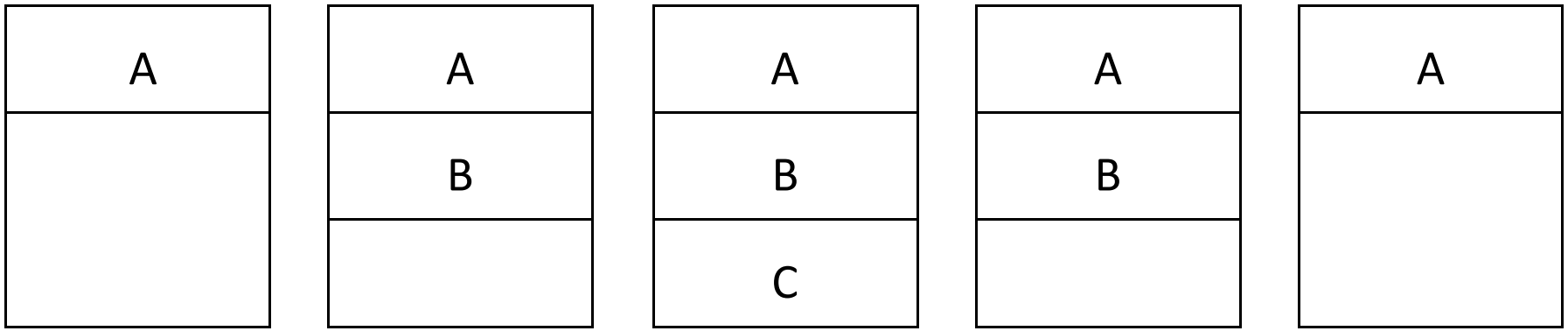
Bloc d’activation



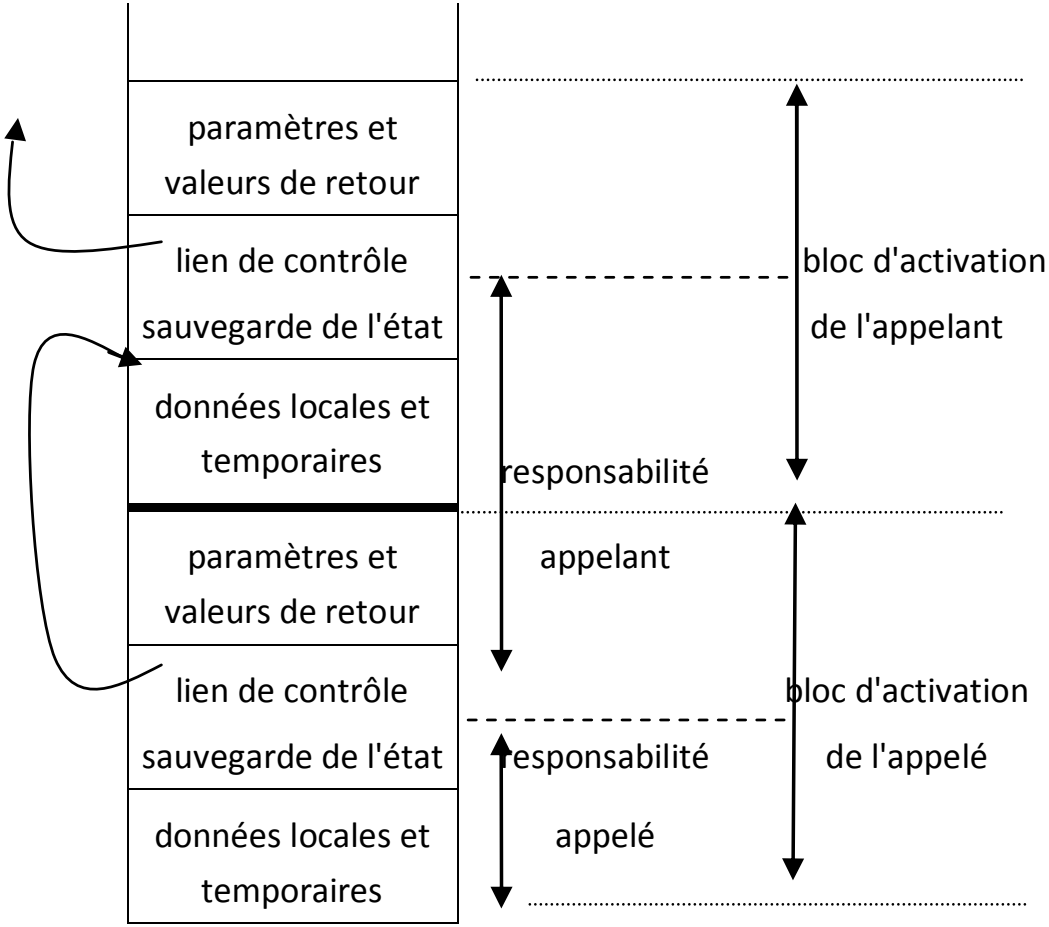
Adresse de retour
Paramètres effectifs
Lien de contrôle
Lien d'accès
Etat machine sauvegardé
Données locales
Temporaires

Bloc d'activation

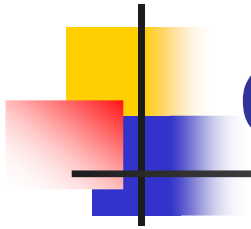
- Considérons une procédure A appelant la procédure B qui elle-même appelle la procédure C.



Protocoles d'appel

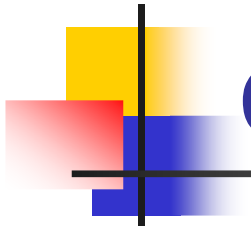


Noms et adresses mémoires dans le langage C

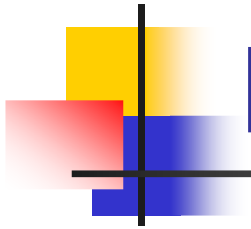


- Associations des noms aux adresses mémoires correspondantes
- Taille et la durée d'existence de ces références utilisées dans le langage C.

Noms et adresses mémoires dans le langage C

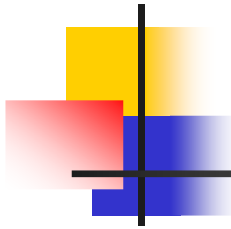


- Les noms utilisés dans le programme source pour désigner les variables, procédures, … n'ont plus d'existence dans un programme exécutable machine.
- Le compilateur devra donc s'assurer du bon usage des noms (déclaration, type, portée, …) et faire l'association avec les emplacements mémoires respectifs.
- Cette association est sauvegardée durant la compilation dans la table des symboles.



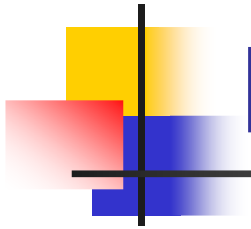
Règles de portée

- Le langage C a adopté une règle courante, appelée règle de portée statique
- C utilise également la règle de "l'englobant le plus imbriqué".
- Si un nom est utilisé à un certain niveau, la déclaration qui s'y applique est la plus imbriquée.



Structure de blocs

```
{
        déclarations
        instructions
}
```

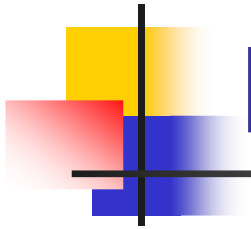


Portée d'une déclaration

- Si un nom X est déclaré dans un bloc B alors cette déclaration s'applique à toutes les références à X dans B.
- Si un nom X n'est pas déclaré dans un bloc B, une occurrence de X dans B est dans la portée d'une déclaration de X dans un bloc englobant B' tel que :
  - **i)** B' comporte une déclaration de X et,
  - **ii)** B' est le plus imbriqué des blocs contenant B et ayant une déclaration de X.

Portée - Exemple

```
main ( )
{   int a = 0;
    int b = 0;
    {           int  b = 1;
                {           int a = 2;
                            printf("a=%d b= %d\n", a , b);
                }
                {           int b = 3;
                            printf("a=%d  b=%d\n", a , b);
                }
                printf("a=%d  b=%d\n", a , b);
    }
    printf("a=%d  b=%d\n", a , b);
}
```

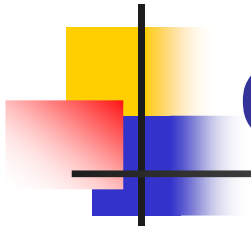


Portée - Exemple

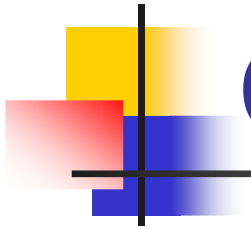
- La sortie du programme précédent donne les valeurs suivantes pour les variables a et b. La règle de l'englobant le plus imbriqué est illustrée par cet exemple.

```
a=2 b=1

a=0 b=3

a=0 b=1

a=0 b=0
```

Implantation de la structure de blocs

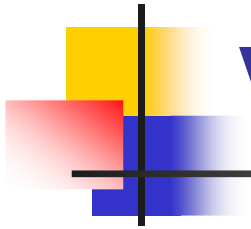


- La structure de blocs est implantée en utilisant une allocation en pile.
- Puisque la portée d'une déclaration ne s'étend pas à l'extérieur du bloc dans lequel elle apparaît, l'emplacement réservé au nom déclaré peut être alloué à l'entré du bloc et libérée à la sortie



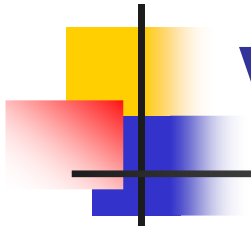
Classes de stockage en C

- La classe de stockage détermine la partie de la mémoire où le stockage est alloué pour les variables et les procédures et la durée de vie de cette allocation durant l'exécution d'un programme.
- Dans le langage C, quatre classes de stockage peuvent être définis par les mots clés : **automatic**, **register**, **external**, and **static**.



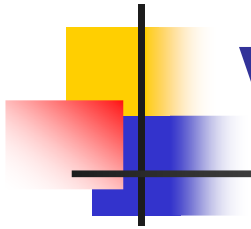
Variables automatiques

- Ces variables sont déclarés au début de bloc d'un programme.
- La mémoire est allouée automatiquement lors de l'entrée à un bloc et libérée automatiquement à la sortie de ce bloc.
- Ces variables sont également appelées variables locales.



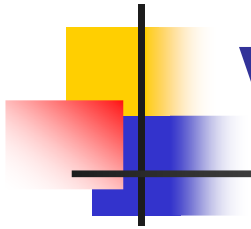
Variables automatiques

- Les variables automatiques peuvent être spécifiés par l'utilisation du mot-clé "auto" devant le nom d'une variable e.g. "auto int nom_variable".
- Toutefois, il n'est pas nécessaire d'utiliser le mot-clé auto car par défaut, la classe de stockage au sein d'un bloc est automatique.



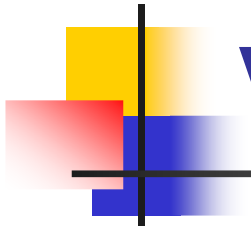
Variables automatiques

- La portée des variables automatiques est locale au bloc dans lequel elles sont déclarées.
- Les variables automatiques déclarées avec des valeurs sont initialisés chaque fois que le bloc dans lequel elles sont déclarées est exécuté.



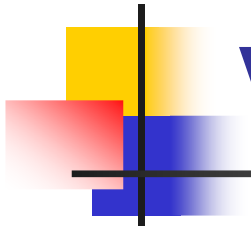
Variables registres

- En langage C, l'utilisation du mot clé "register" devant une variable (e.g. "register int nom_variable") suggère au compilateur de stocker cette variable dans un registre lors de la production de code.
- Cette déclaration n'est pas obligatoire pour le compilateur mais satisfaite tant que possible.



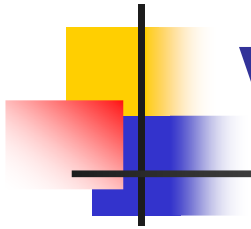
Variables externes

- Les variables externes sont des variables globales introduites par le mot clé "extern" e.g. "extern int nom_variable".
- Elles sont déclarées en dehors de tout bloc ou procédure.



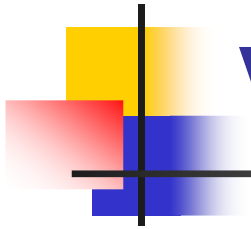
Variables externes

- Ces variables sont de portée globale i.e. accessibles à partir de n'importe quel bloc ou procédure.
- Toutefois, si une variable locale ayant le même nom est déclarée dans un bloc ou procédure, les références au nom auront accès à la cellule de variable locale.



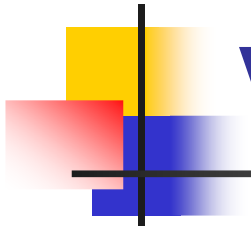
Variables externes

- A la différence des variables locales, les variables externes persistent en mémoire jusqu'à la fin de l'exécution d'un programme.
- Les variables externes sont stockées dans la zone data (variables initialisées) ou bss (variables non initialisées) d'un exécutable.



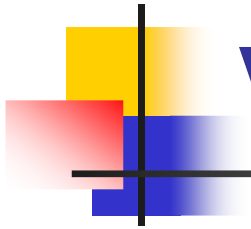
Variables externes

- Si une variable est définie au début d'un fichier source, le mot clé extern peut être omis.
- Si une variable est définie dans un fichier_1 et utilisée dans un fichier_2 alors le mot clé extern doit être utilisé dans le fichier_2.



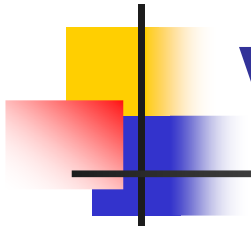
Variables externes

- Si le nombre de variables et procédures externes devient grand, l'utilisation courante consiste à les regrouper dans un fichier en-tête séparé ("fichier.h"), puis inclus en utilisant la directive "#include fichier.h".



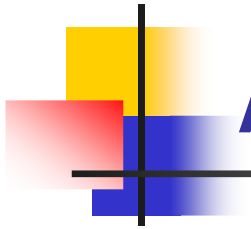
Variables statiques

- Les variables statiques sont des variables globales introduites par le mot clé "static" e.g. "static int nom_variable".
- A l'instar des variables externes, les variables statiques persistent en mémoire jusqu'à la fin de l'exécution d'un programme.
- La différence est que les variables statiques ont des portées limitées.



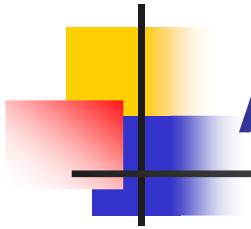
Variables statiques

- Les variables statiques définies dans un bloc continuent d'exister même après la fin de l'exécution du bloc ou procédure dans lesquels elles sont définies.
- La portée de variables statiques automatiques est identique à celle de variables automatiques.
- La valeur d'une variable statique dans une procédure est retenue entre les appels successifs de cette procédure.



Allocation dynamique

- Les données allouées dynamiquement dans un programme n'ont pas d'existence dans un exécutable sur mémoire secondaire.
- Ces données vont exister pendant l'exécution du programme dans la zone tas ou heap en anglais.



Allocation dynamique

- Le langage C fournit une librairie de fonctions qui permettent d'allouer ou de libérer des espaces mémoires dans le tas. Pour cela le fichier en-tête "stdlib.h" doit être inclus.
- Les principales fonctions et constantes utilisées sont données ci-après : malloc(), calloc(), realloc(), free(), NULL.