CI - mémoire

- Les 3 modes d'allocations :
  - l'allocation statique : variables globales
  - l'allocation automatique : variables de contexte
  - l'allocation dynamique : réservation et libération par le programmeur, accessible via des pointeurs.

les variables

- Quel que que soit le point d'exécution du programme, lorsqu'une variable *v* existe, elle correspond :
  - à un espace mémoire réservé
  - la taille de cet espace dépend du type de la variable : sizeof(v)
  - l'interprétation de cet espace dépend du type de la variable
  - la valeur (i.e. l'interprétation) est accessible via *v*
  - l'adresse de début de cet espace est *&v*
  - ainsi les octets d'adresses &*v*, &*v*+1, … &v+sizeof(v)-1 sont réservés pour la variable *v*

les variables

- Quel que que soit le point d'exécution du programme, lorsqu'une variable *v* existe, elle correspond :
  - à un espace mémoire réservé
  - la taille de cet espace dépend du type de la variable : sizeof(v)
  - l'interprétation de cet espace dépend du type de la variable
  - la valeur (i.e. l'interprétation) est accessible via *v*
  - l'adresse de début de cet espace est *&v*
  - ainsi les octets d'adresse &*v*, &*v*+1, … &v+sizeof(v)-1 sont réservés pour la variable *v*
- Lorsque la variable n'est plus accessible, la mémoire correspondant n'est plus réservée et est susceptible d'être utilisée par ailleurs.

Allocation statique

- Correspond aux variables globales et aux chaines de caractères.
- Disponible dès le chargement du programme et ce jusqu'à la fin du processus.
- Exemple :

```
int c=0;
char str[]="hello" ;
```



```
$ gcc -c  a.c
$ nm a.o
0000000000000004 C c
0000000000000000 D str
```

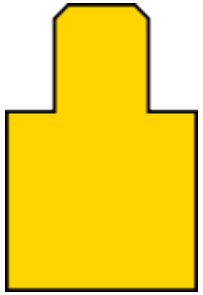
Allocation automatique

- Ce sont les variables de contexte : déclarées dans un bloc {…}
- Paramètres de fonction
- Variables « dites » locales
- Ces variables n'existent que dans le bloc
- La mémoire correspondante est automatiquement prise dans la pile en début de bloc
- Cette mémoire est automatiquement libérée en sortie de bloc.

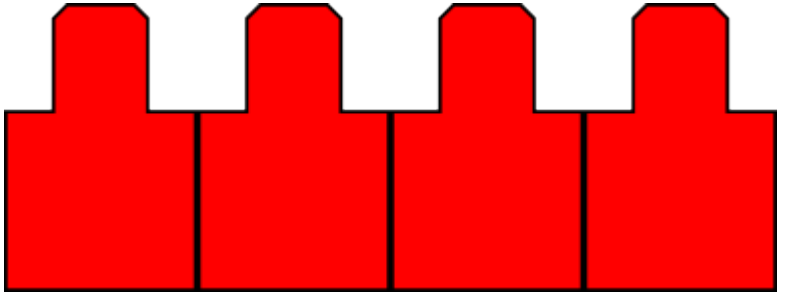
Simulation mémoire

Comment comprendre la mémoire avec des MegaBlock(R)

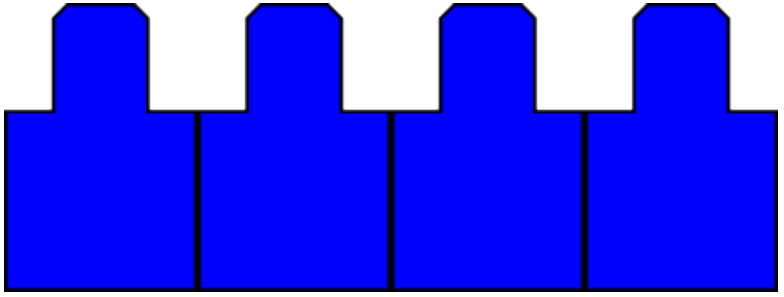
Présentation du kit:



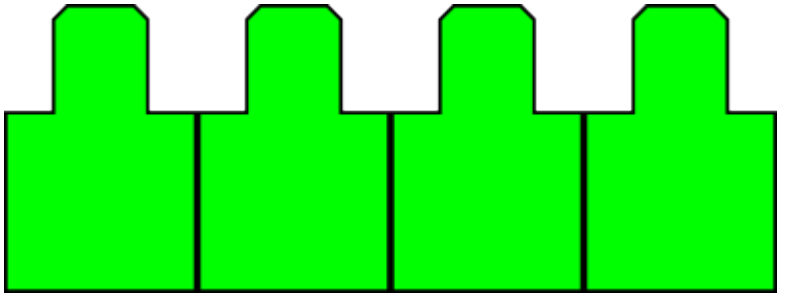
char



int



pointeur



appel de fonction

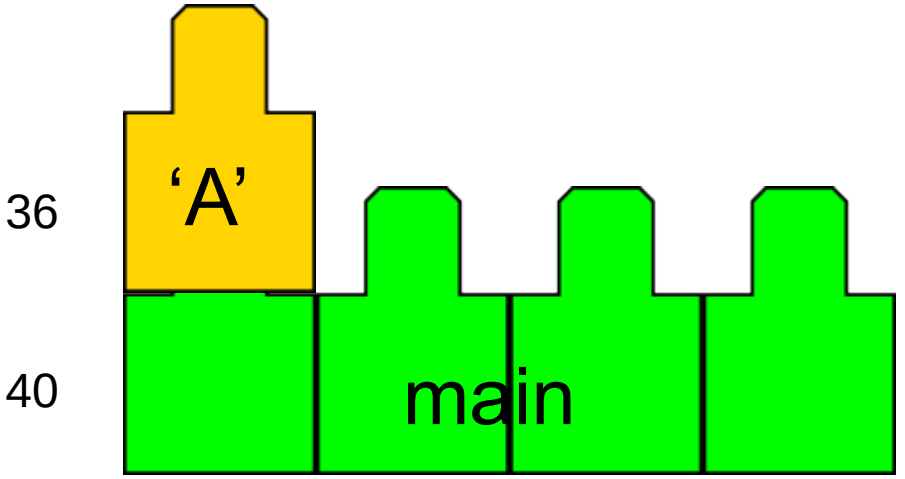
Exemple

```
int main(){
        char c='A' ;
}
```

les adresses de pile commencent à 40.

Exemple

```
int main(){
        char c='A' ;
}
```

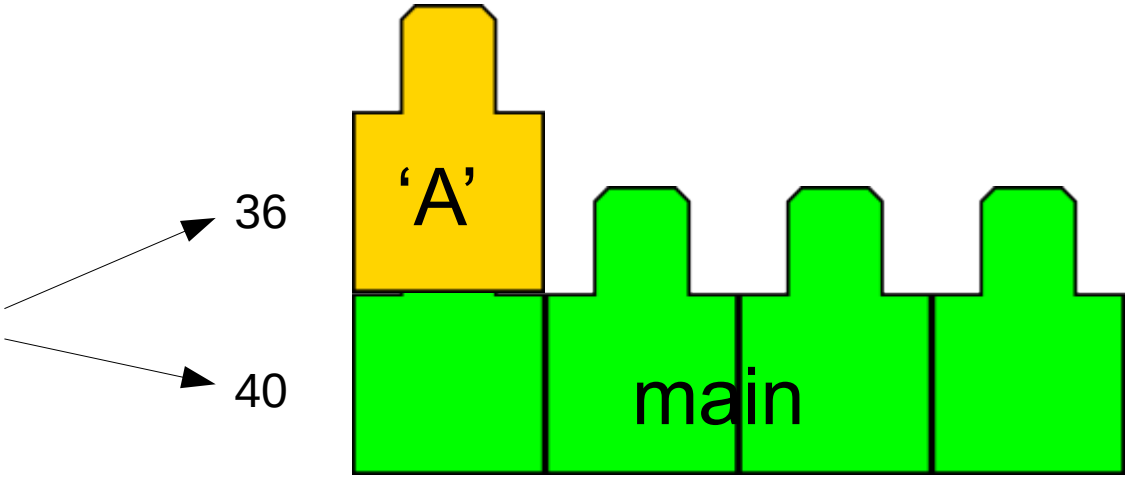


les adresses de pile commencent à 40.

Exemple

```
int main(){
        char c='A' ;
}
```

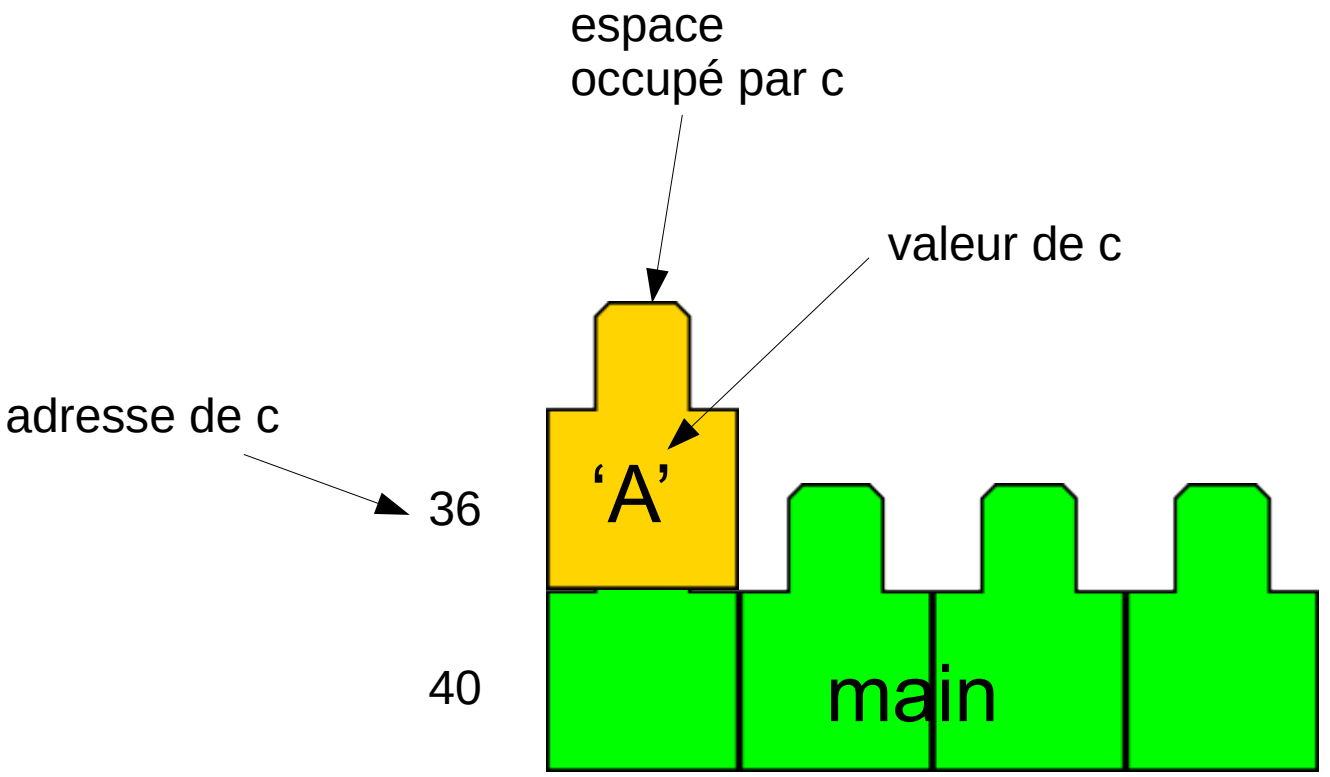
Les adresses de piles sont décroissantes. Si 0 ⇒ segfault



les adresses de pile commencent à 40.

Exemple

```
int main(){
        char c='A' ;
}
```

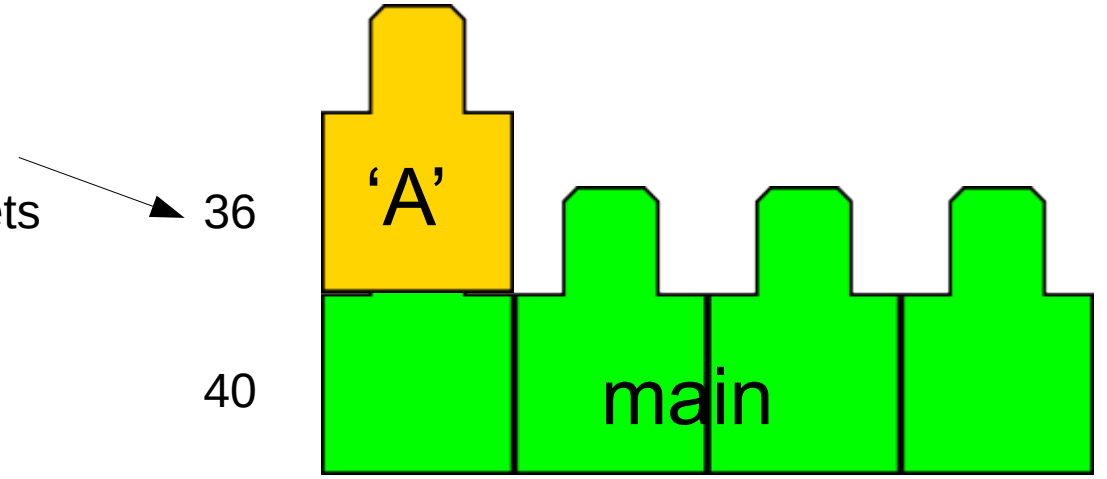


les adresses de pile commencent à 40.

Exemple

```
int main(){
        char c='A' ;
}
```

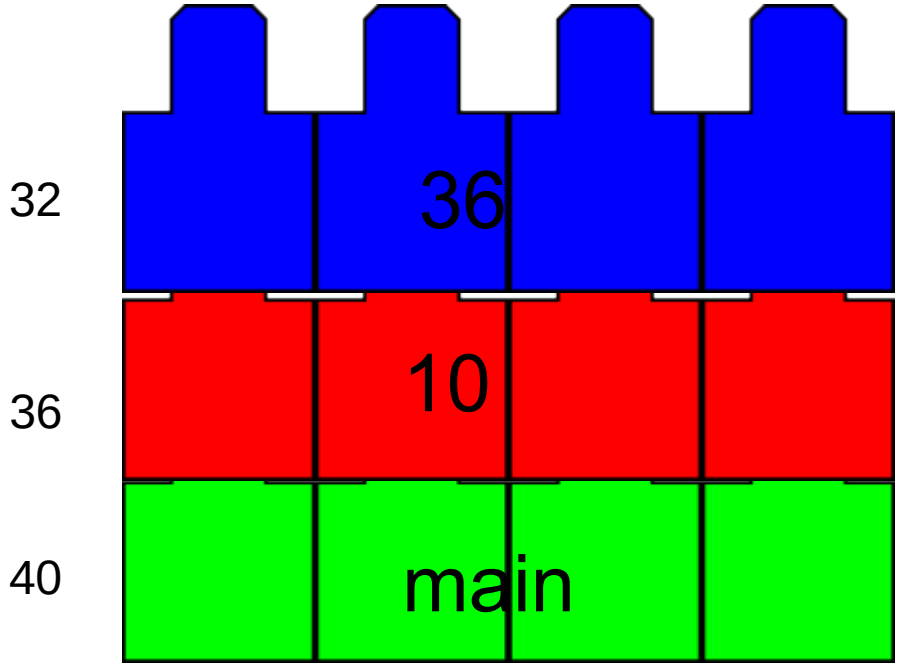
Les adresses de variables sont « alignées » sur 4 octets



les adresses de pile commencent à 40.

Les pointeurs

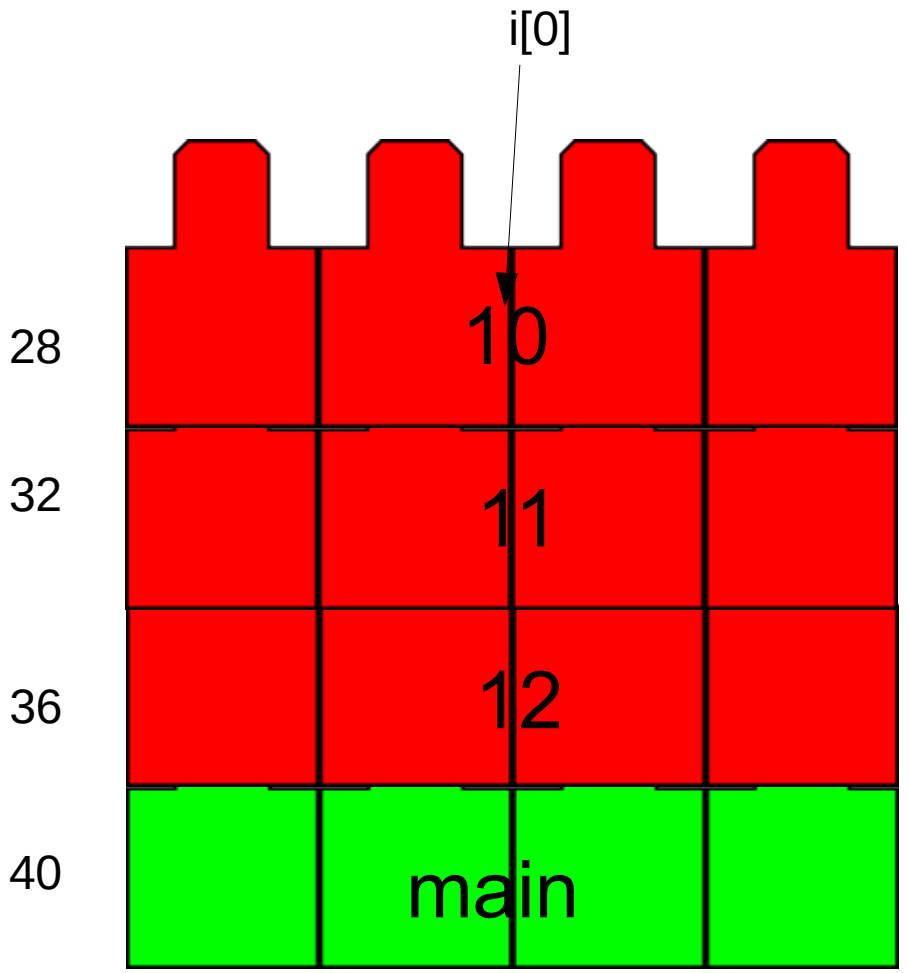
```
int main(){
      int i=10 ;
      int *p=&i ;
}
```



Les tableaux automatiques

```
int main(){
    int i[]={10,11,12} ;
}
```

valeur de i
et adresse de i !
⇒ i == &i



TP

Faire les 9 premiers exercices

Les appels de fonctions

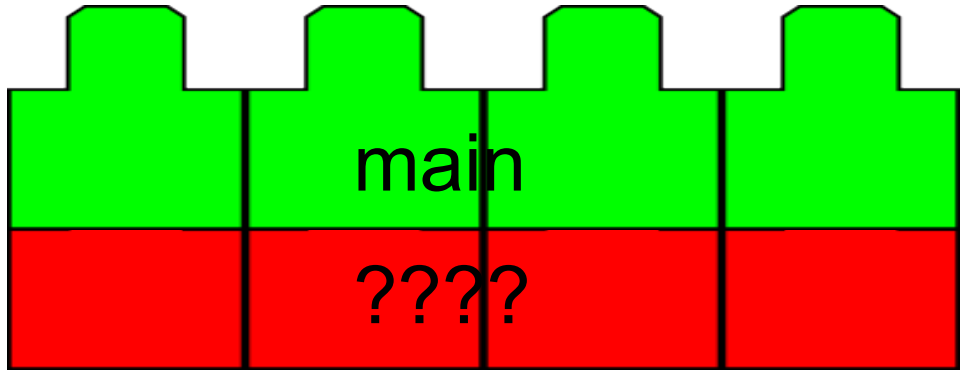
- On simplifie l'appel de fonction comme suit :
  - Lors de l'appel de fonction, on réserve dans la pile l'espace nécessaire pour stocker la valeur de retour de la fonction
  - On empile l'appel : bloc vert
  - On empile alors les paramètres de la fonction
  - On exécute le code de la fonction

exemple

```
int f(char c){
   int x = c+1;
   return x ;
}

int main(){
   char p='A' ;
   int r ;
   r=f(p) ;
   return r ;
}
```

réservation de l'espace
pour la valeur de retour

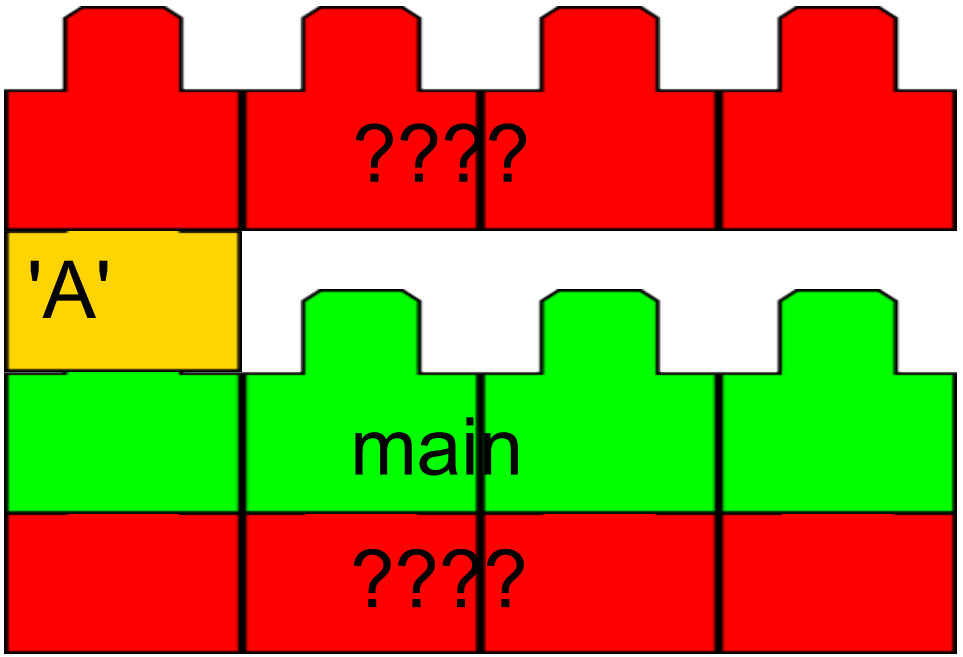


exemple

```
int f(char c){
   int x = c+1;
   return x ;
}

int main(){
   char p='A' ;
   int r ;
   r=f(p) ;
   return r ;
}
```

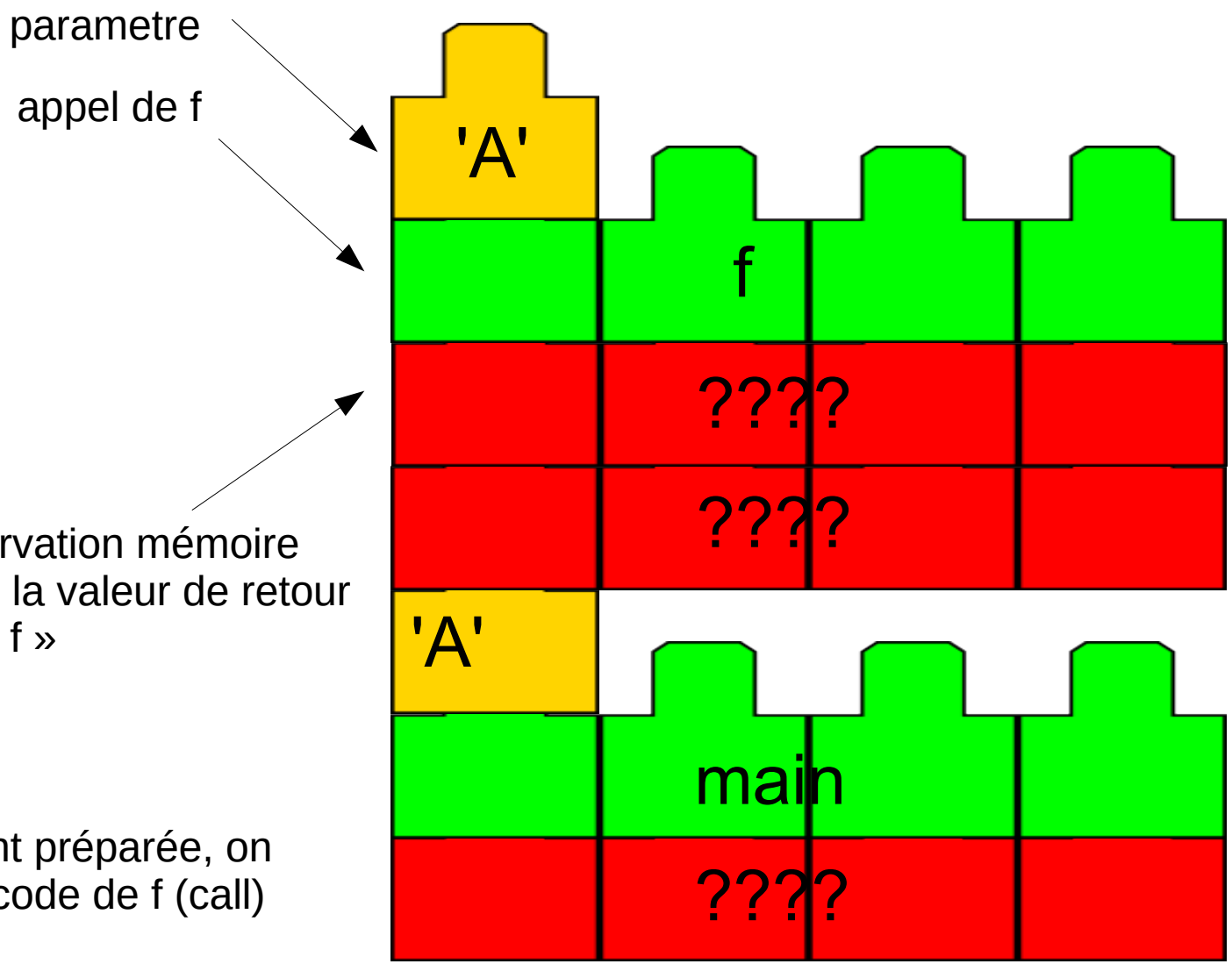
variables
locales de
la fonction « main »



exemple

```
int f(char c){
   int x = c+1;
   return x ;
}

int main(){
   char p='A' ;
   int r ;
   r=f(p) ;
   return r ;
}
```



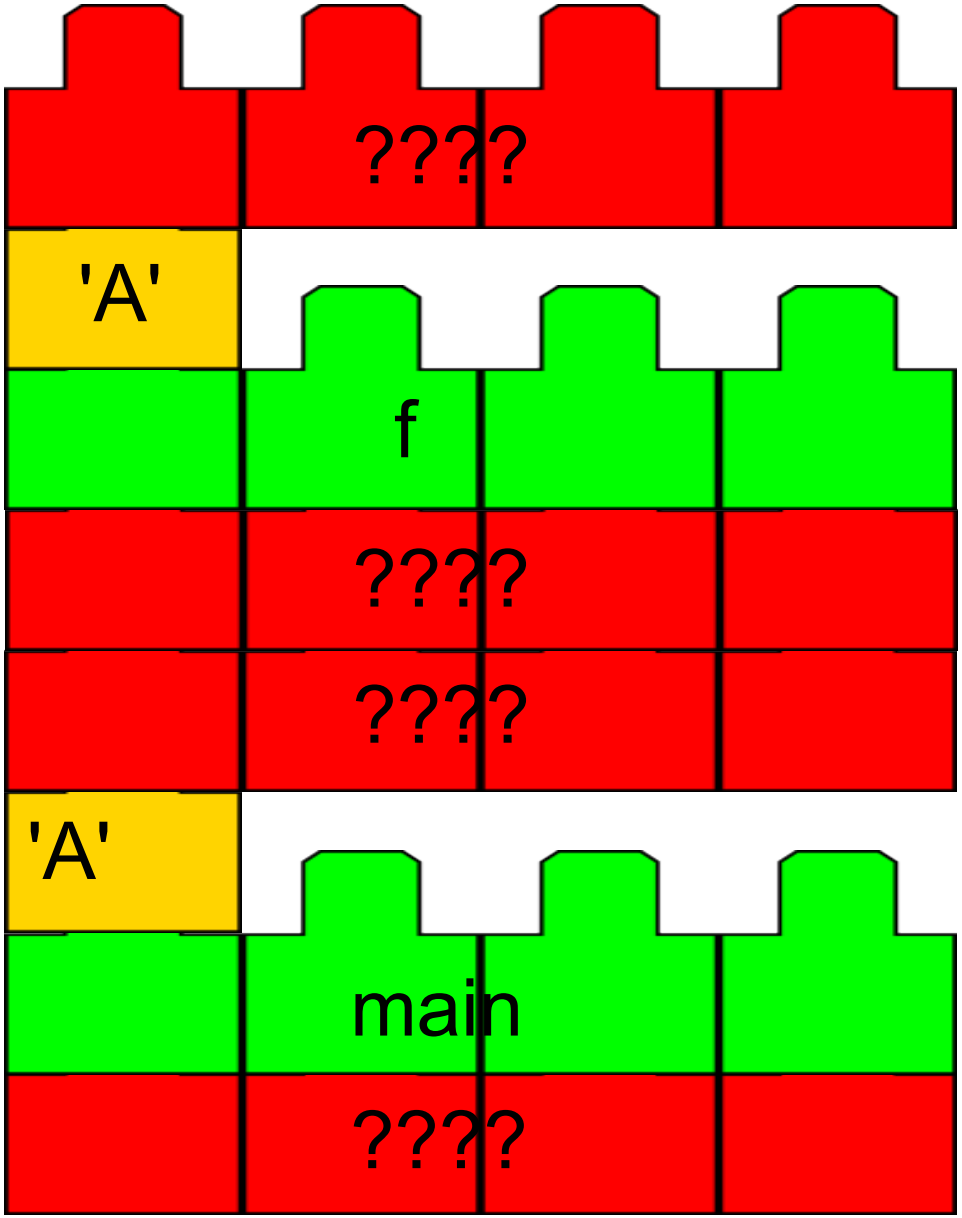
La pile est maintenant préparée, on peut passer dans le code de f (call)

exemple

```
int f(char c){
   int x = c+1;
   return x ;
}

int main(){
   char p='A' ;
   int r ;
   r=f(p) ;
   return r ;
}
```

variables
locales de
la fonction « f »

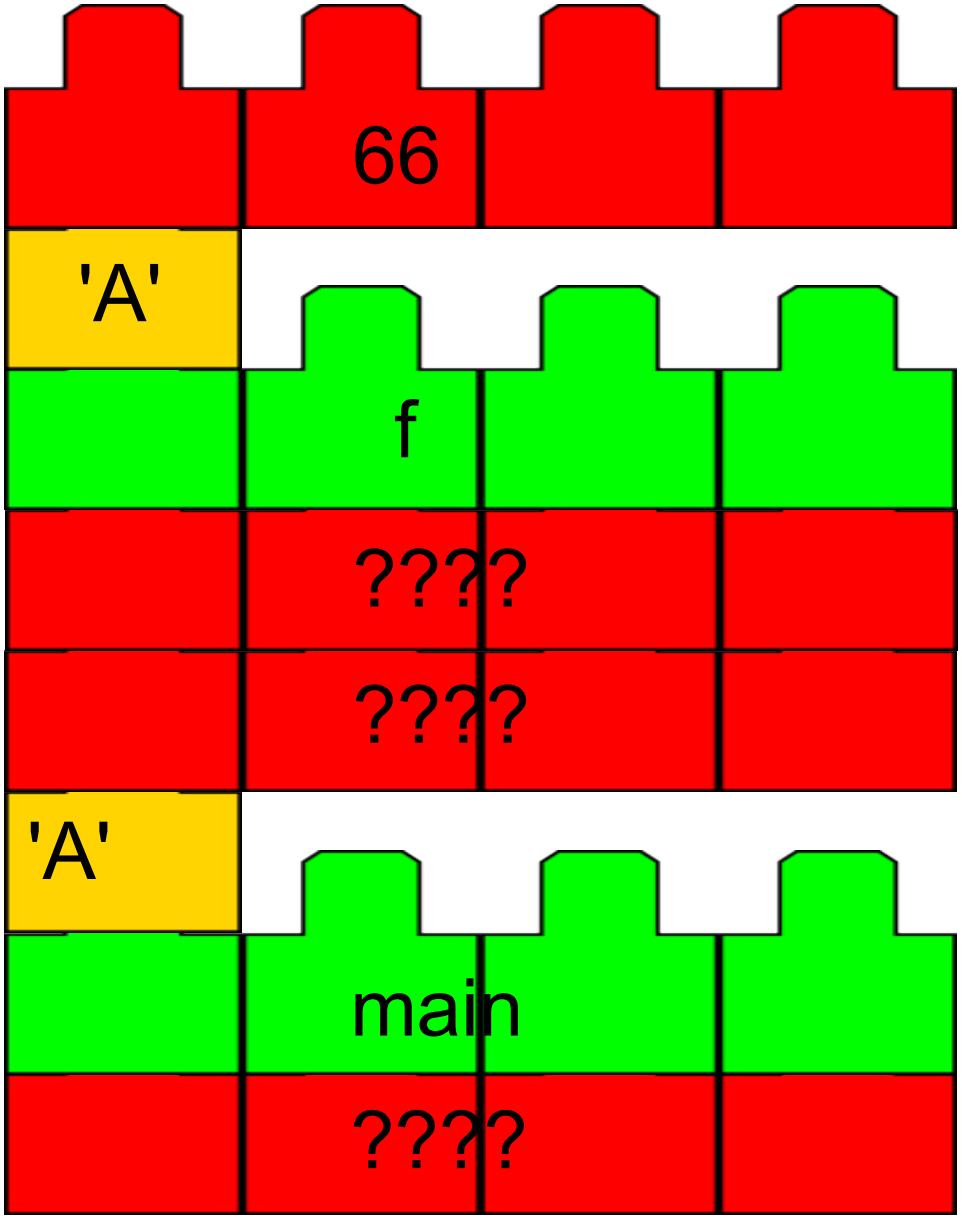


exemple

```
int f(char c){
   int x = c+1;
   return x ;
}

int main(){
   char p='A' ;
   int r ;
   r=f(p) ;
   return r ;
}
```

affectation dans x

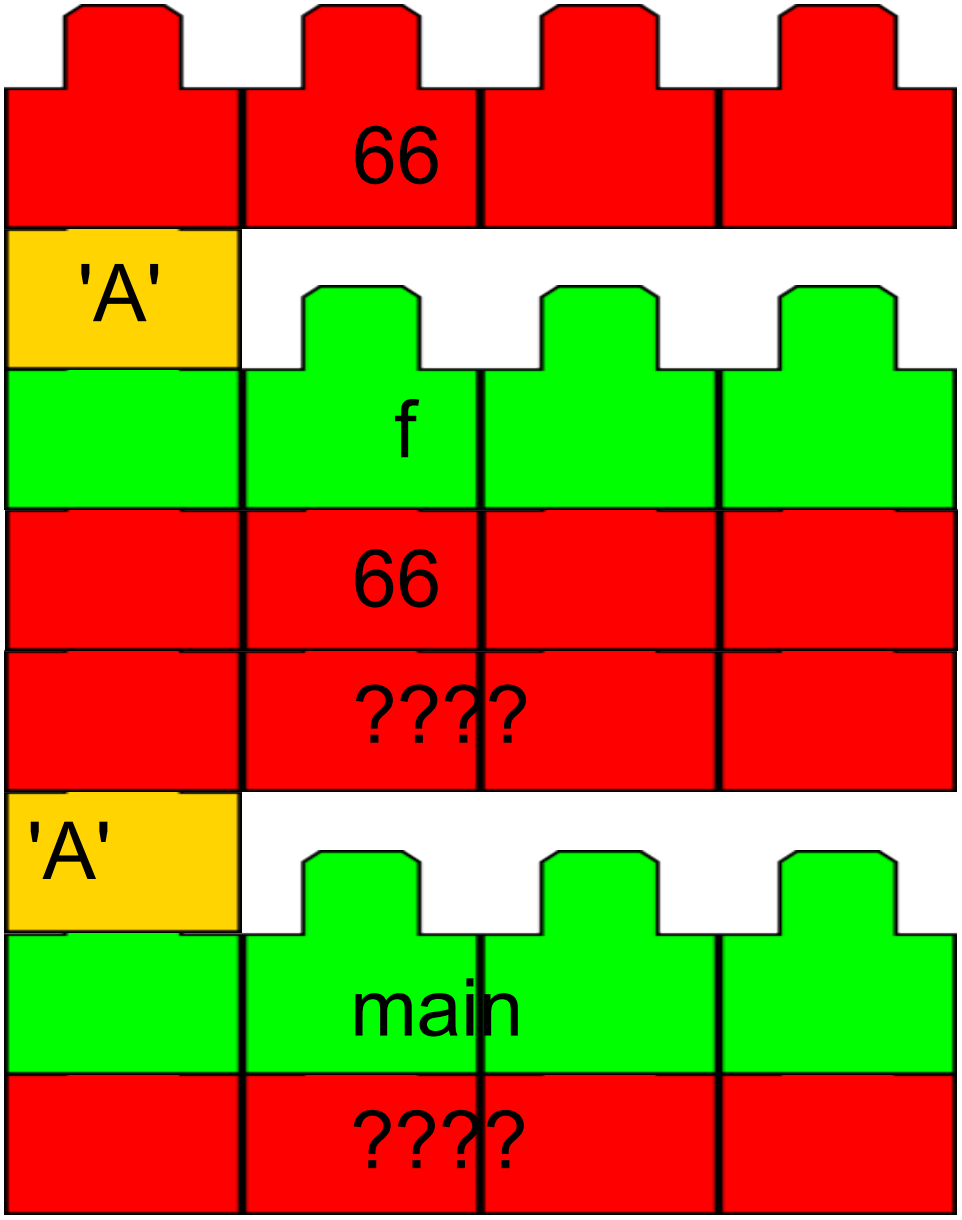


exemple

```
int f(char c){
   int x = c+1;
   return x ;
}

int main(){
   char p='A' ;
   int r ;
   r=f(p) ;
   return r ;
}
```

copie de x dans
la valeur de retour

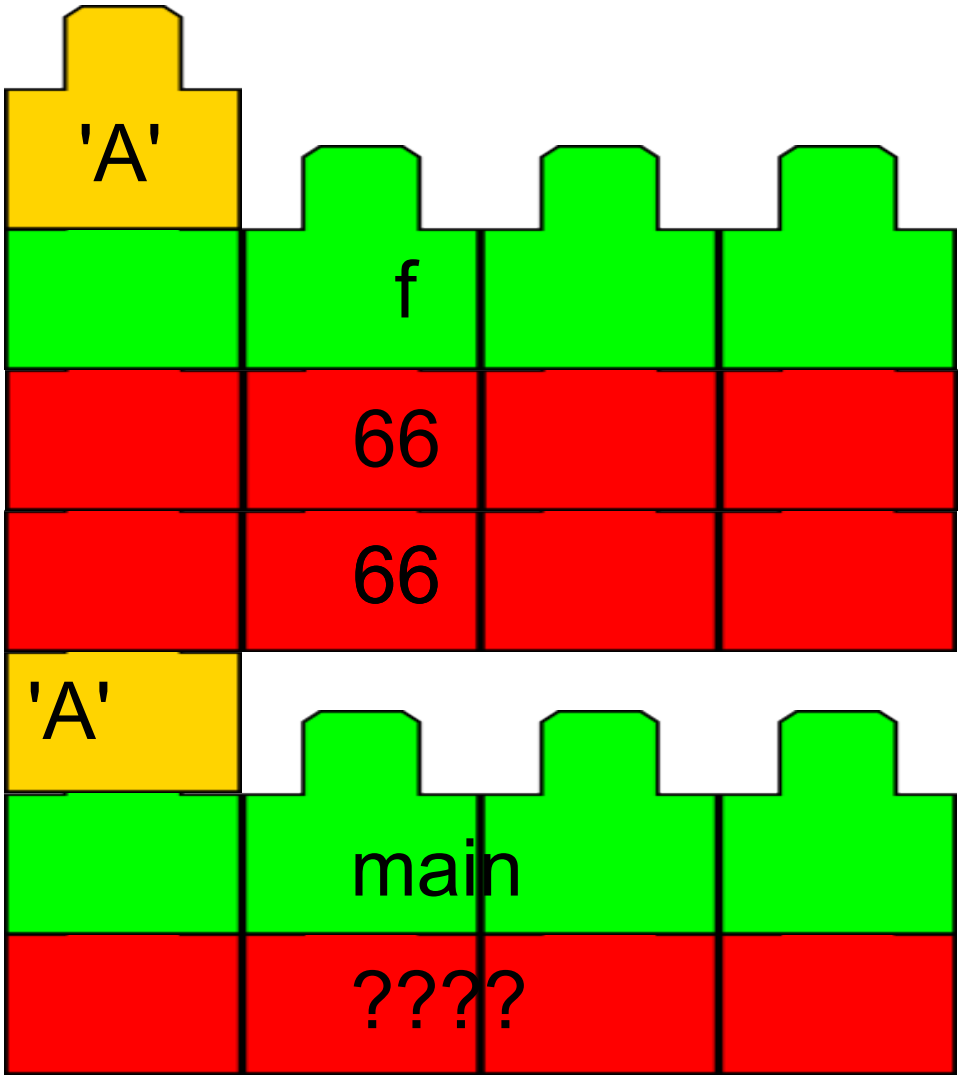


exemple

```
int f(char c){
   int x = c+1;
   return x ;
}

int main(){
   char p='A' ;
   int r ;
   r=f(p) ;
   return r ;
}
```

affectation de la valeur de retour dans r

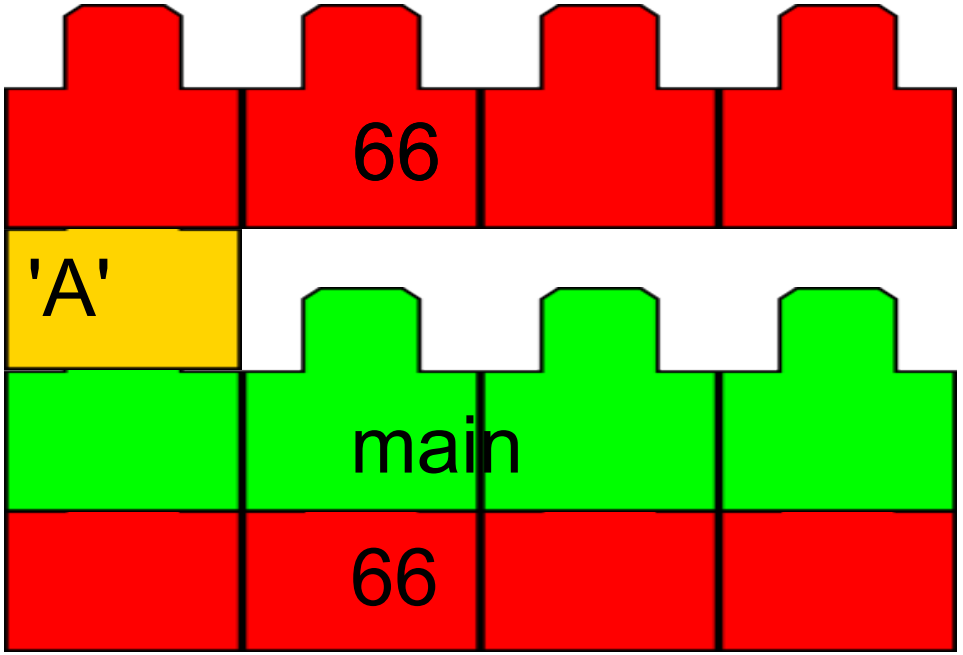


exemple

```
int f(char c){
   int x = c+1;
   return x ;
}

int main(){
   char p='A' ;
   int r ;
   r=f(p) ;
   return r ;
}
```

copie de la valeur de r
dans la valeur de retour



TP

Faire les exercices de

« Appel de fonction : fonction sans paramètre »

à

« Appel de fonction : récursivité »

L'allocation dynamique

- L'allocation dynamique correspond à l'utilisation de mémoire dans le tas.
- On demande de la mémoire avec *malloc*
- On restitue la mémoire avec *free*
- Important :
  - il n'y a **aucune vérification** que vous demandez la bonne quantité de mémoire par rapport à l'interprétation que vous allez faire de celle-ci
  - malloc renvoie des adresses :

    ⇒ son utilisation repose **obligatoirement** sur l'utilisation de pointeur
  - Le type du pointeur : *foo* * détermine comment seront utilisés et interprétés les octets réservés dans le tas.

L'allocation dynamique

- exemple:

  int *p = malloc(1) ;

  *p=10 ;

- On demande 1 octet, l'adresse est fournie par malloc et conservée dans p. Lorsque l'on fait **p,* on interprète l'adresse contenue dans *p* comme le début d'une zone de *sizeof(int)* octets !

- Pour un pointeur :

  &p : l'adresse mémoire du début d'une zone de *sizeof(int *)* octets

  p : correspond à la valeur du pointeur, c'est à dire une adresse

  *p : correspond à la valeur pointée par *p*, ici un entier donc on **suppose** que l'adresse de *p* est le début d'une zone de 4 octets

free

- Pour rendre de la mémoire prise dans le tas, vous devez faire appel à **free**

  free(p)

- free prend comme unique paramètre l'adresse de début de la zone précédemment obtenue par malloc.
- Une fois une zone mémoire restituée, celle-ci pourra être à nouveau utilisée par un malloc

TP

Terminer les exercices