# Langage C

## Apprendre à lire...

François Cuvelier

Institut Galilée
Université Paris XIII.

19 octobre 2019

# Plan

Listing 1 – Exemple de déclarations

```
1    TRUC a; /* ESSAI */
2    TRUC *pT;
3    TRUC t[3];
```

Listing 1 – Exemple de déclarations

```
1    TRUC a; /* ESSAI */
2    TRUC *pT;
3    TRUC t[3];
```

- a est un TRUC,

Listing 1 – Exemple de déclarations

```
1    TRUC a; /* ESSAI */
2    TRUC *pT;
3    TRUC t[3];
```

- a est un TRUC,
- **pT** est un pointeur de TRUC.

Listing 1 – Exemple de déclarations

```
1    TRUC a; /* ESSAI */
2    TRUC *pT;
3    TRUC t[3];
```

- a est un TRUC,
- **pT** est un pointeur de TRUC.
- **t** est un tableau de 3 TRUC.

Lorsque le nom du tableau t est utilisé dans une expression il est **converti** en un pointeur valant l'adresse du premier élément du tableau (i.e. &t[0]).
On note

$$\boxed{t \rightsquigarrow \&t[0]}$$

cette convertion.
Attention, la convertion n'a pas lieu dans les deux cas suivants :

1. utilisation de la fonction sizeof. sizeof(t) correspond à la taille globale du tableau.
2. utilisation de l'opérateur &. &t[i] est l'adresse du i-ème élément du tableau t.

Pour acceder aux éléments d'un tableau, on peux utiliser l'opérateur d'indiçage [] . En fait, t[i] est remplacé par le compilateur en $*(t+i)$. on note

$$\boxed{t[i] \hookrightarrow *(t+i)}$$

cette opération.
Pour effectuer la somme d'une adresse de TRUC avec un entier, le compilateur utilise la relation suivante :

$$t+i \rightsquigarrow \&t[0]+i*\text{sizeof}(\text{TRUC})$$

et donc

$$*(t+i) \rightsquigarrow *(\&t[0]+i*\text{sizeof}(\text{TRUC}))$$

Pour affecter une valeur correcte à pT, il faut lui donner un pointeur (i.e. une adresse) de TRUC existant. Par exemple :

Listing 2 – Exemples simples d'affectation

```
1    pT=&a ;
2    pT=&t [0] ;
3    pT=t ;
```

Examinons les instructions suivantes

Listing 3 – Exemples complexes d'affectation

```
1    pT=t+1;
2    *pT=a;
3    *++pT=a;
```

Examinons les instructions suivantes

Listing 3 – Exemples complexes d'affectation

```
1    pT=t+1;
2    *pT=a;
3    *++pT=a;
```

- On a t+1 $\rightsquigarrow$ &t[0]+1*sizeof(TRUC). On affecte donc à pT la valeur de &t[1].

Examinons les instructions suivantes

Listing 3 – Exemples complexes d'affectation

```
1    pT=t+1;
2    *pT=a;
3    *++pT=a;
```

- On a t+1 $\rightsquigarrow$ &t[0]+1*sizeof(TRUC). On affecte donc à pT la valeur de &t[1].
- Pour la seconde intruction, *pT correspond alors à *(&t[1]) c'est à dire à t[1]. En d'autres termes, *pT est l'objet pointé par pT. Donc, on affecte à t[1] la valeur de a.

Examinons les instructions suivantes

Listing 3 – Exemples complexes d'affectation

```
1    pT=t+1;
2    *pT=a;
3    *++pT=a;
```

- On a t+1 $\rightsquigarrow$ &t[0]+1*sizeof(TRUC). On affecte donc à pT la valeur de &t[1].
- Pour la seconde intruction, *pT correspond alors à *(&t[1]) c'est à dire à t[1]. En d'autres termes, *pT est l'objet pointé par pT. Donc, on affecte à t[1] la valeur de a.
- La dernière instruction est équivalente à ++pT; *pT=a; ou encore pT=pT+1; *pT=a;, c'est à dire pT prend la valeur &t[2] et, ensuite, t[2] la valeur de a.

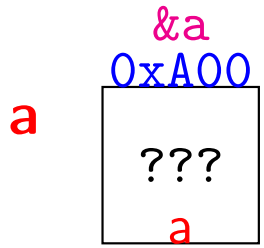
Listing 4 – pointeur_ex01.c

```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6      int t[3]={1,2,3};
7
8      a=5;
9      pT=t+2;
10     *pT=a;
11     *--pT=a-1;
12     return 0;
13 }
```

```
int a;
```

Listing 4 – pointeur_ex01.c

```c
#include <stdio.h>

int main(){
    int a;
    int *pT;
    int t[3]={1,2,3};

    a=5;
    pT=t+2;
    *pT=a;
    *--pT=a-1;
    return 0;
}
```

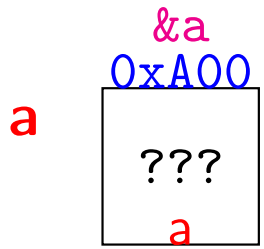



**int a;**

a est de type **int** (non initialisé)

Listing 4 – pointeur_ex01.c

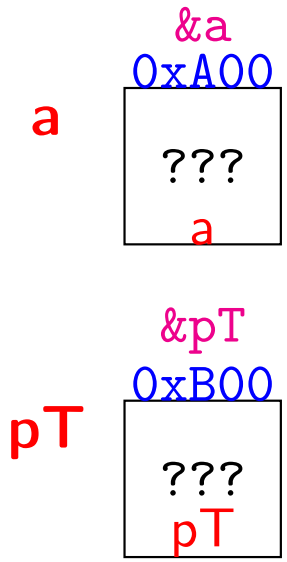
```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6      int t[3]={1,2,3};
7
8      a=5;
9      pT=t+2;
10     *pT=a;
11     *--pT=a-1;
12     return 0;
13 }
```




```
int *pT;
```

Listing 4 – pointeur_ex01.c

```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6      int t[3]={1,2,3};
7
8      a=5;
9      pT=t+2;
10     *pT=a;
11     *--pT=a-1;
12     return 0;
13 }
```

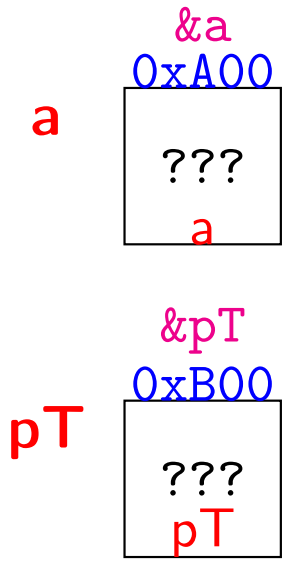



**int *pT;**

pT est un pointeur de **int** (non initialisé)

Listing 4 – pointeur_ex01.c

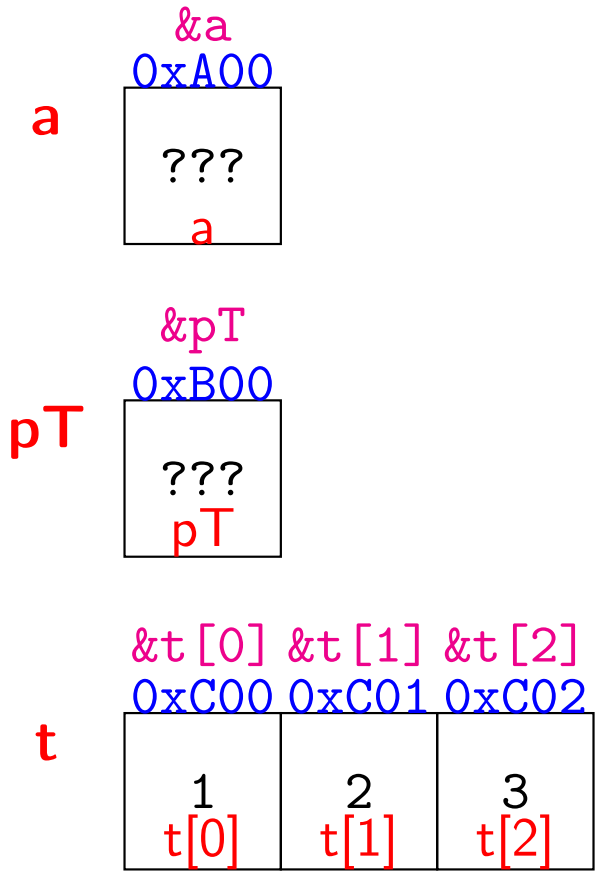
```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6  ▶   int t[3]={1,2,3};
7
8      a=5;
9      pT=t+2;
10     *pT=a;
11     *--pT=a-1;
12     return 0;
13 }
```




**int** t[3]={1, 2, 3};

Listing 4 – pointeur_ex01.c

```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6      int t[3]={1,2,3};
7
8      a=5;
9      pT=t+2;
10     *pT=a;
11     *--pT=a-1;
12     return 0;
13 }
```

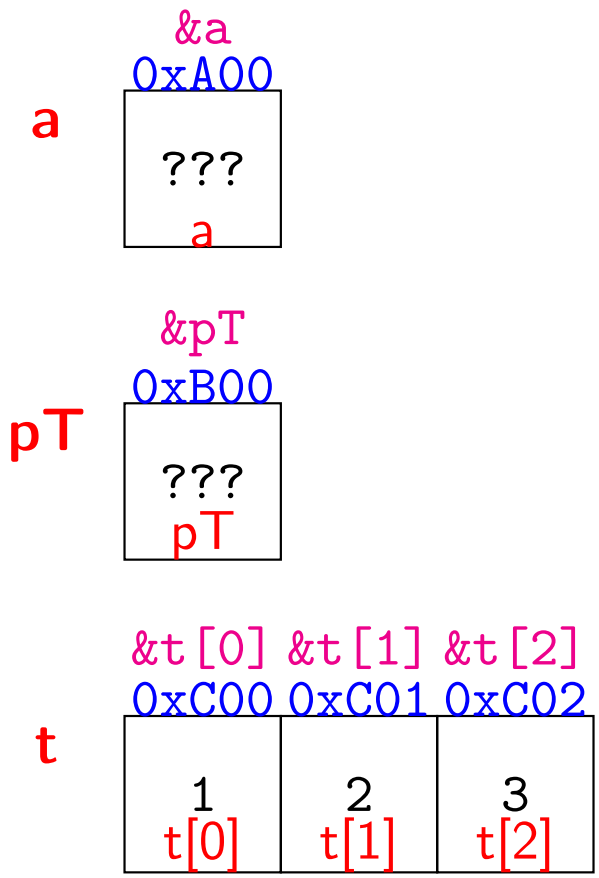



**int** t[3]={1, 2, 3};

t est un tableau de 3 **int** initialisé par {1, 2, 3}

Listing 4 – pointeur_ex01.c

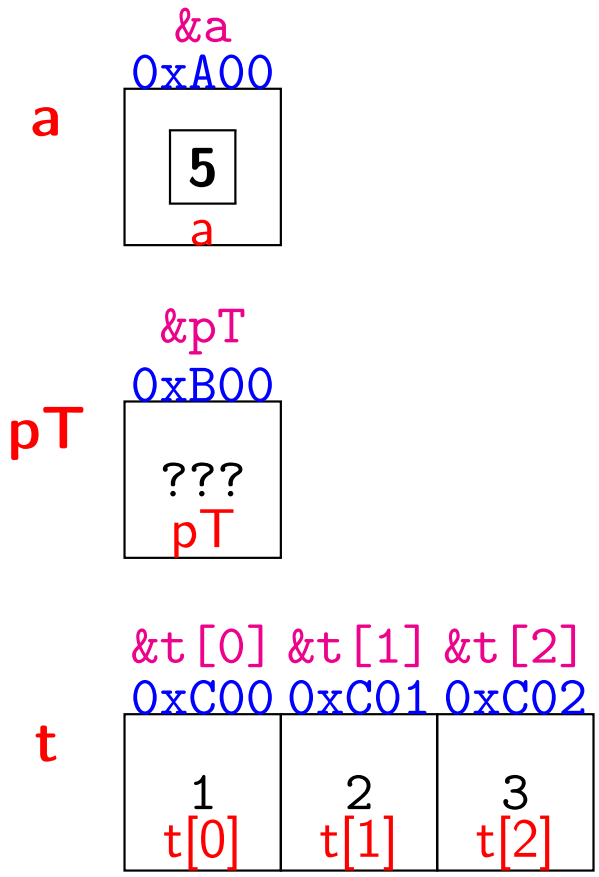
```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6      int t[3]={1,2,3};
7
8      a=5;
9      pT=t+2;
10     *pT=a;
11     *--pT=a-1;
12     return 0;
13 }
```




a=5;

Listing 4 – pointeur_ex01.c

```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6      int t[3]={1,2,3};
7
8      a=5;
9      pT=t+2;
10     *pT=a;
11     *--pT=a-1;
12     return 0;
13 }
```

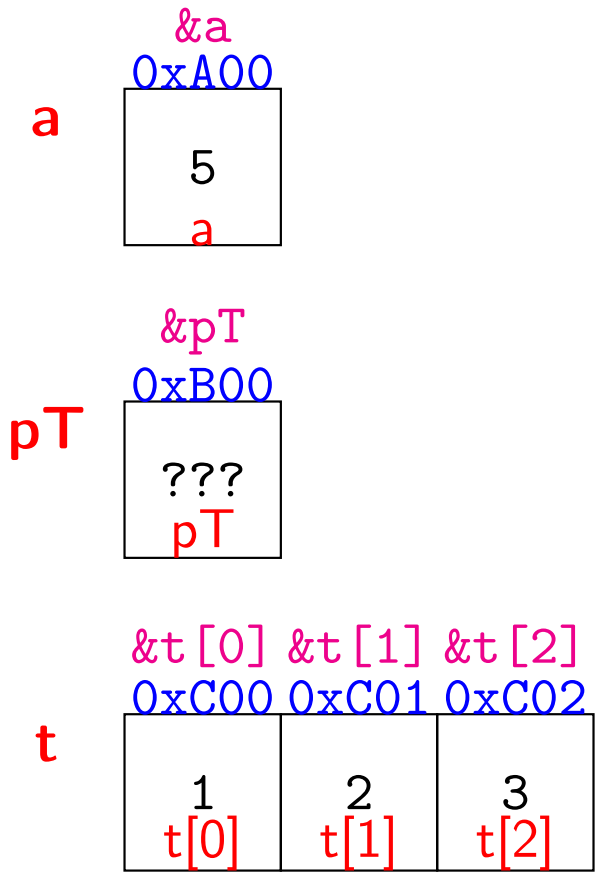



a=5;

On affecte 5 à a

Listing 4 – pointeur_ex01.c

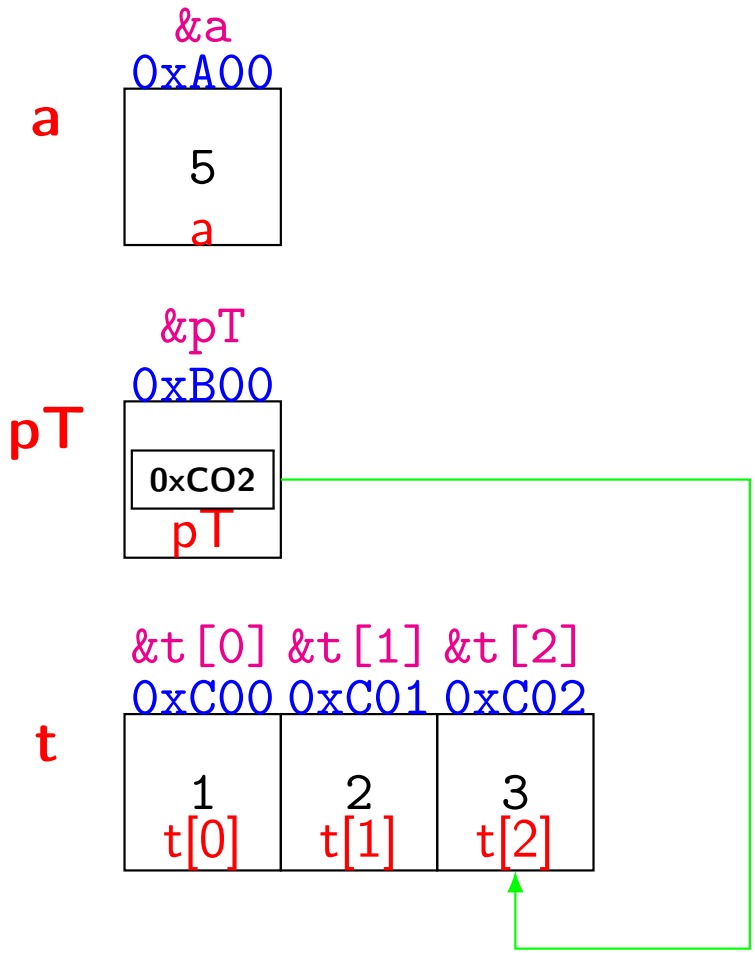
```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6      int t[3]={1,2,3};
7
8      a=5;
9  ▶   pT=t+2;
10     *pT=a;
11     *--pT=a-1;
12     return 0;
13 }
```




pT=t+2;

Listing 4 – pointeur_ex01.c

```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6      int t[3]={1,2,3};
7
8      a=5;
9  ▶   pT=t+2;
10     *pT=a;
11     *--pT=a-1;
12     return 0;
13 }
```

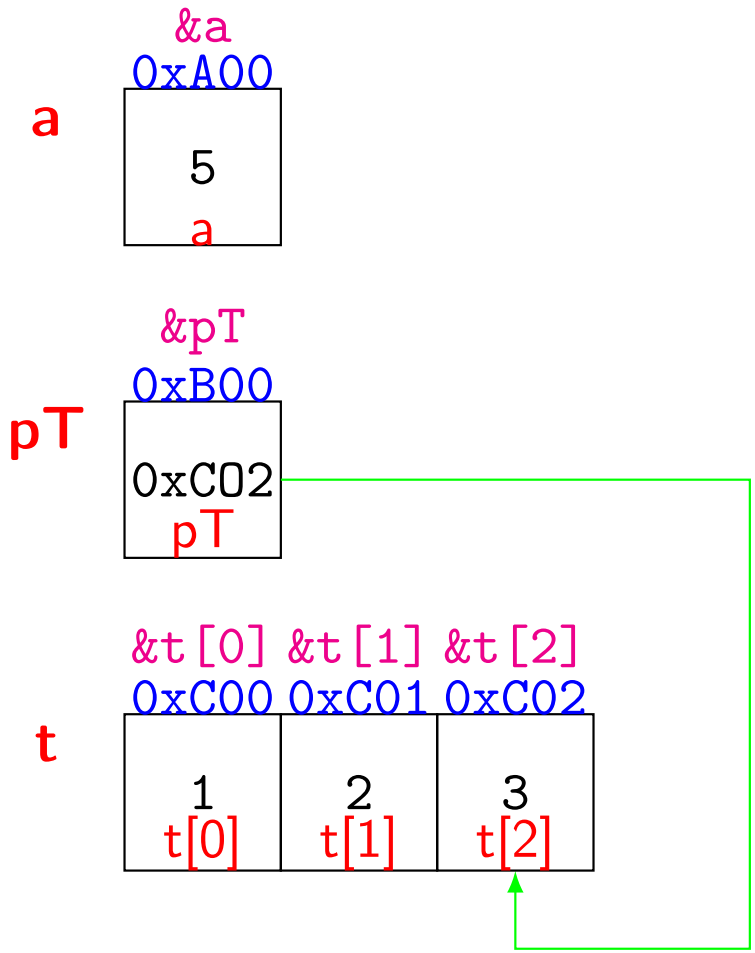



pT=t+2;

t+2 $\rightsquigarrow$ &t[0]+2*sizeof(TRUC) $\Rightarrow$ pT $\leftarrow$ &t[2]

Listing 4 – pointeur_ex01.c

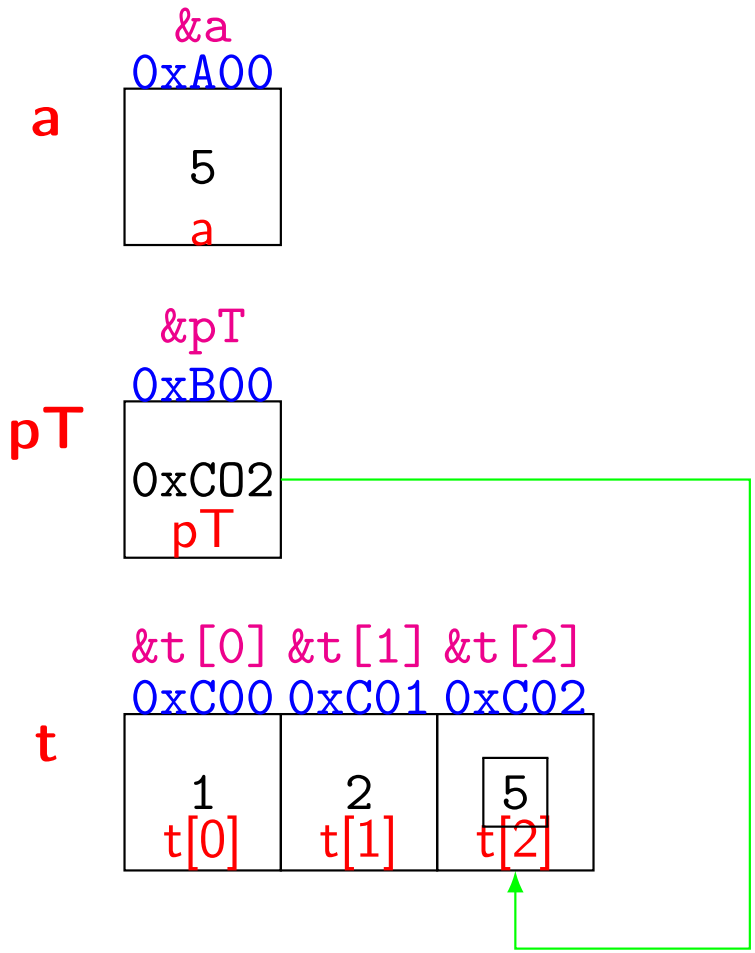
```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6      int t[3]={1,2,3};
7
8      a=5;
9      pT=t+2;
10 ▶   *pT=a;
11     *--pT=a-1;
12     return 0;
13 }
```




*pT=a;

Listing 4 – pointeur_ex01.c

```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6      int t[3]={1,2,3};
7
8      a=5;
9      pT=t+2;
10 ▶   *pT=a;
11     *--pT=a-1;
12     return 0;
13 }
```

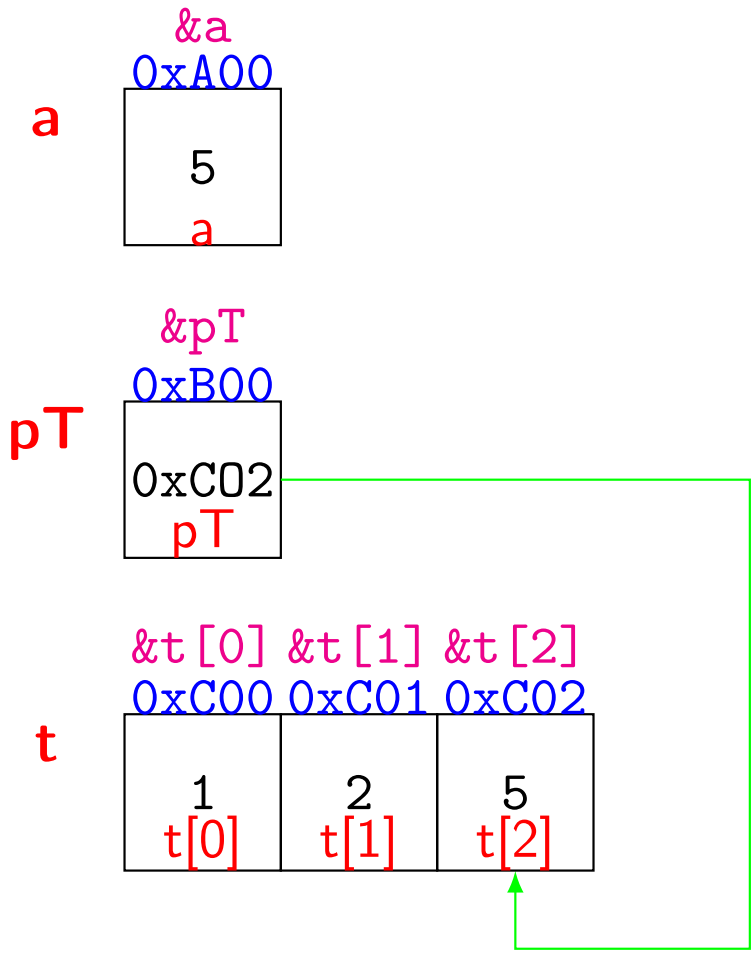



*pT=a;

*pT $\iff$ t[2] $\Rightarrow$ t[2] $\leftarrow$ a

Listing 4 – pointeur_ex01.c

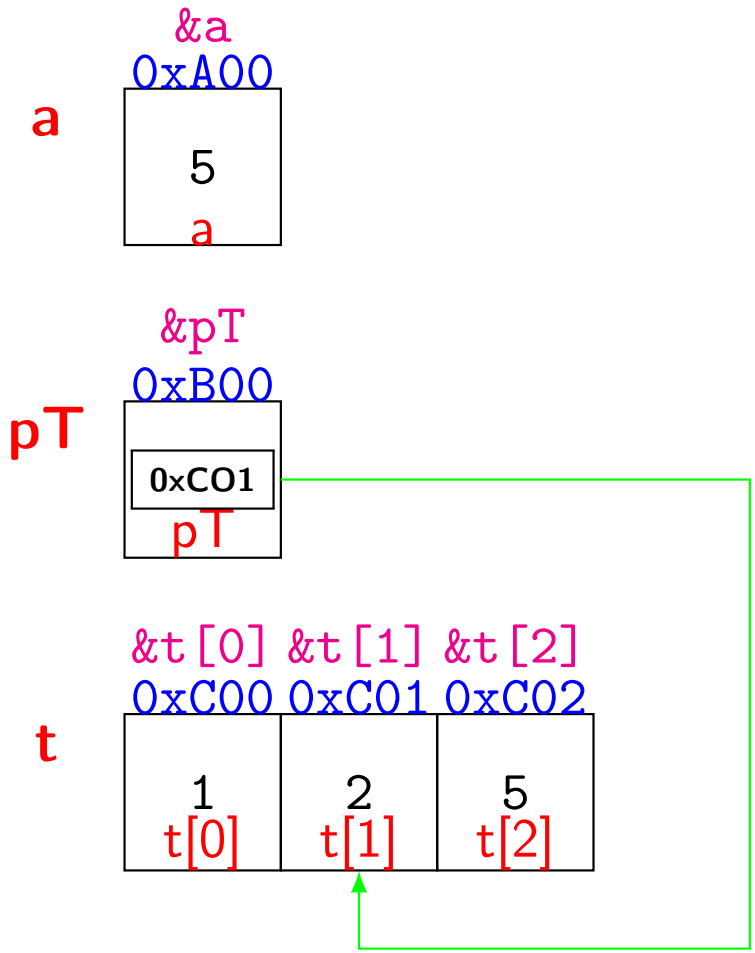
```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6      int t[3]={1,2,3};
7
8      a=5;
9      pT=t+2;
10     *pT=a;
11 ▶   *--pT=a-1;
12     return 0;
13 }
```




*--pT=a-1;

Listing 4 – pointeur_ex01.c

```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6      int t[3]={1,2,3};
7
8      a=5;
9      pT=t+2;
10     *pT=a;
11 ▶   *--pT=a-1;
12     return 0;
13 }
```

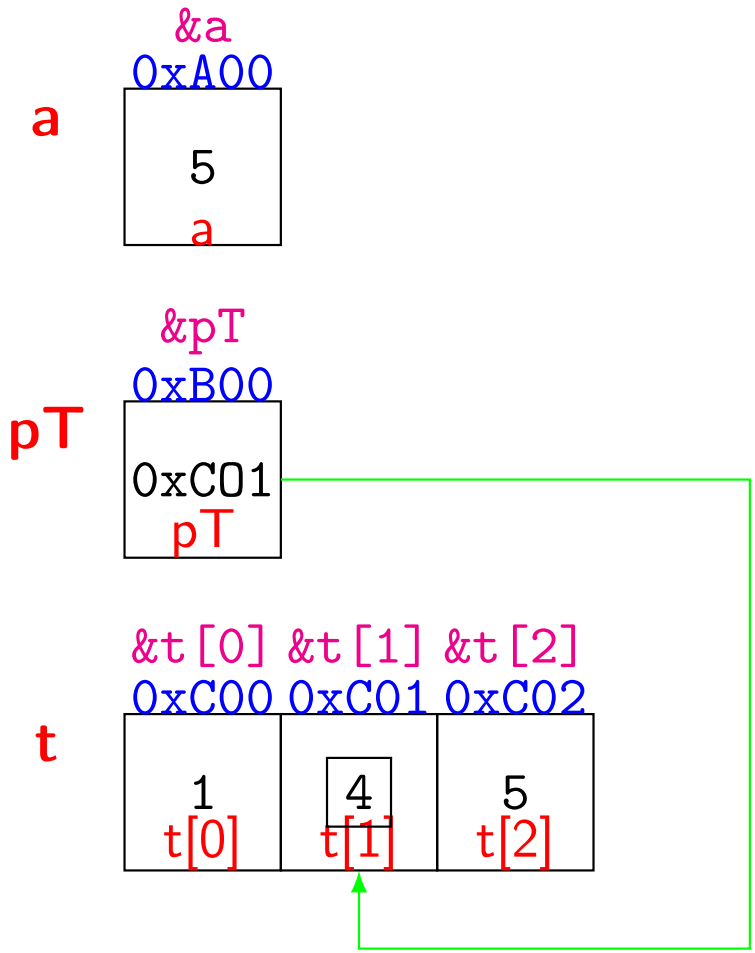



*--pT=a-1;

*--pT=a-1; $\iff$ --pT; *pt=a-1; $\Rightarrow$ pT $\leftarrow$ &t[1]

Listing 4 – pointeur_ex01.c

```
1  #include <stdio.h>
2
3  int main(){
4      int a;
5      int *pT;
6      int t[3]={1,2,3};
7
8      a=5;
9      pT=t+2;
10     *pT=a;
11 ▶   *--pT=a-1;
12     return 0;
13 }
```

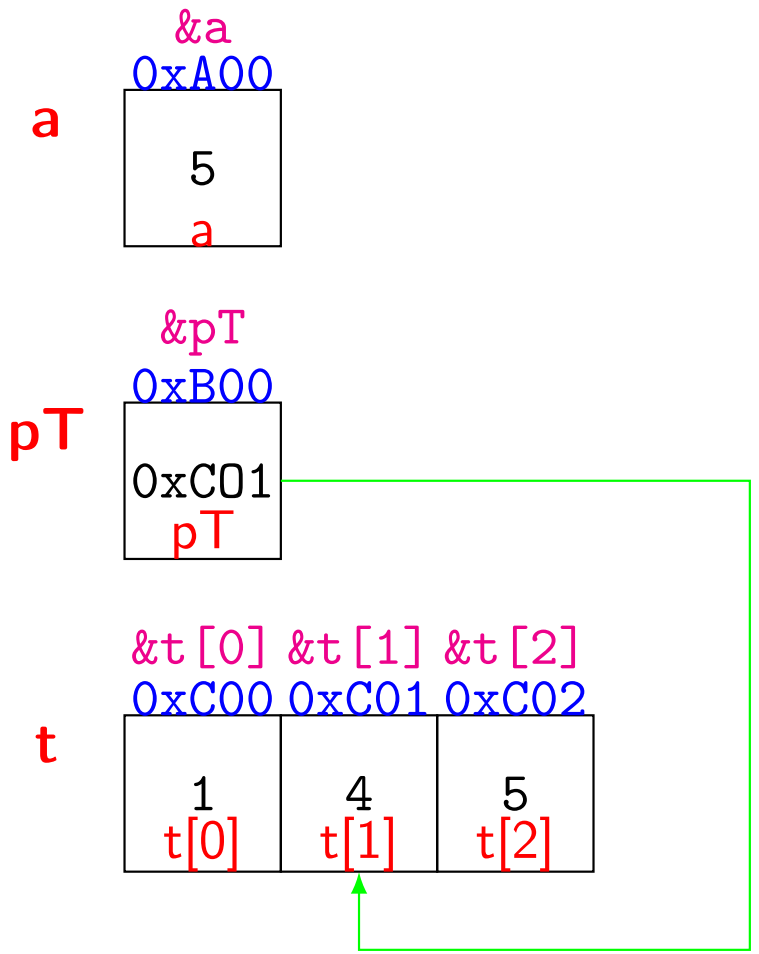



*--pT=a-1;

*--pT=a-1; $\iff$ --pT; *pt=a-1; $\Rightarrow$ pT $\leftarrow$ &t[1] puis t[1] $\leftarrow$ a-1

Listing 4 – pointeur_ex01.c

```c
#include <stdio.h>

int main(){
   int a;
   int *pT;
   int t[3]={1,2,3};

   a=5;
   pT=t+2;
   *pT=a;
   *--pT=a-1;
   return 0;
}
```



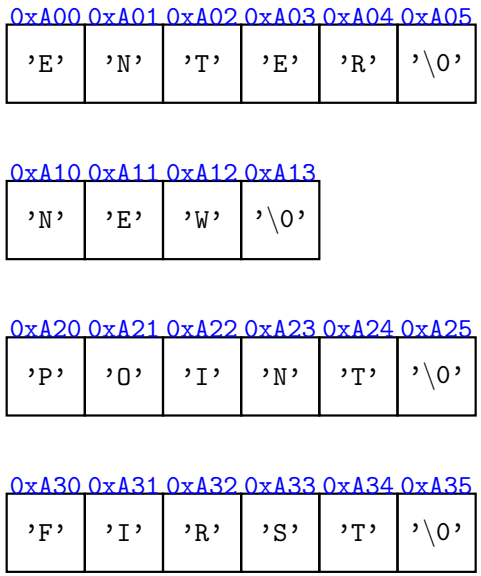
# Plan

## Question

*Expliquer (à l'aide de schémas de représentation mémoire,...) le déroulement du programme suivant. Qu'affiche-t-il ? Décrire le contenu des variables* c, cp *et* cpp *en fin d'exécution.*

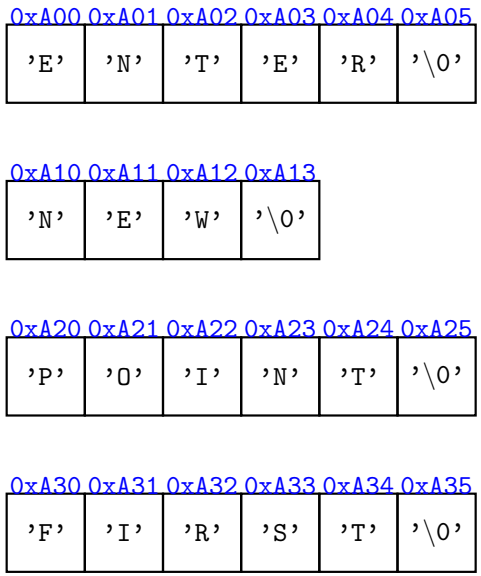
Listing 5 – Exo1.c

```c
#include <stdio.h>

void main(){
   char *c[] = {"ENTER","NEW","POINT","FIRST"};
   char **cp[] = {c+3,c+2,c+1,c};
   char ***cpp = cp;

   printf("%s",**++cpp);
   printf("%s",*--*++cpp+3);
   printf("%s",*cpp[-2]+3);
   printf("%s\n",cpp[-1][-1]+1);
}
```



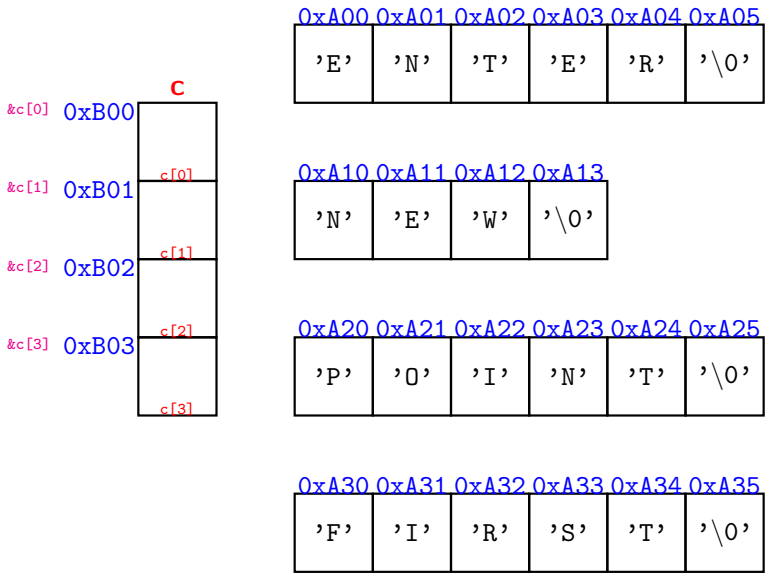

**char** *c[] = {"ENTER","NEW","POINT","FIRST"};

{"ENTER","NEW","POINT","FIRST"} ?



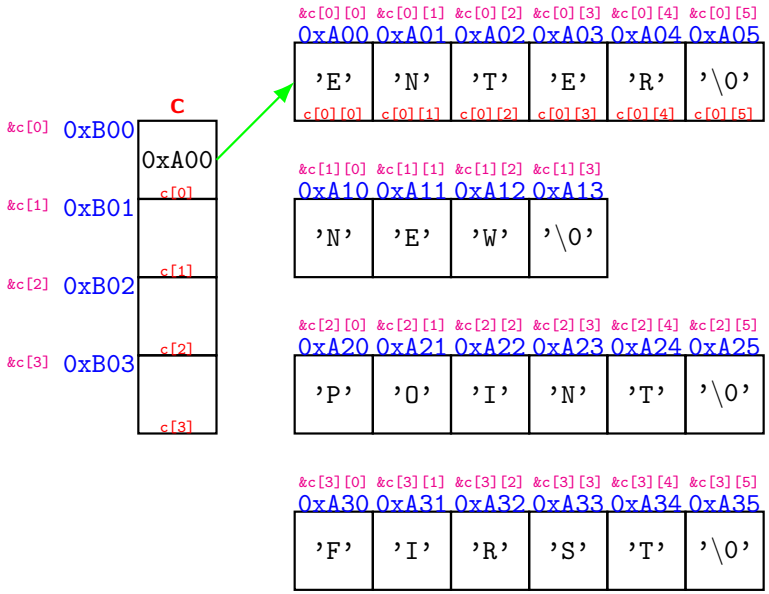

**char** *c[] = {"ENTER","NEW","POINT","FIRST"};

- c est un ?



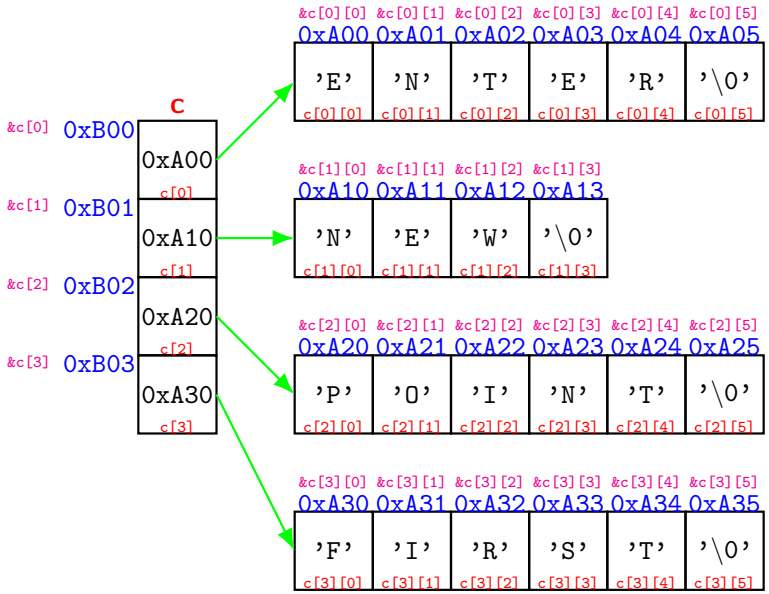

**char** *c[] = {"ENTER","NEW","POINT","FIRST"};

- c est un tableau de 4 (nombre d'éléments de la liste) pointeurs de char.
- c[0] contient ?



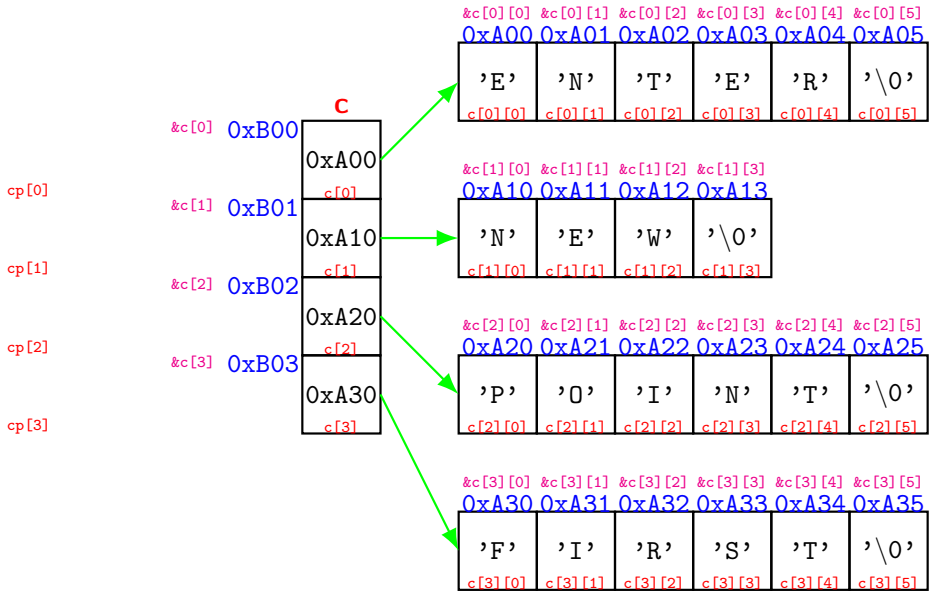

**char** *c[] = {"ENTER","NEW","POINT","FIRST"};

- c est un tableau de 4 (nombre d'éléments de la liste) pointeurs de char.
- c[0] contient l'adresse du premier élément de la chaine "ENTER",
- c[1] contient ?



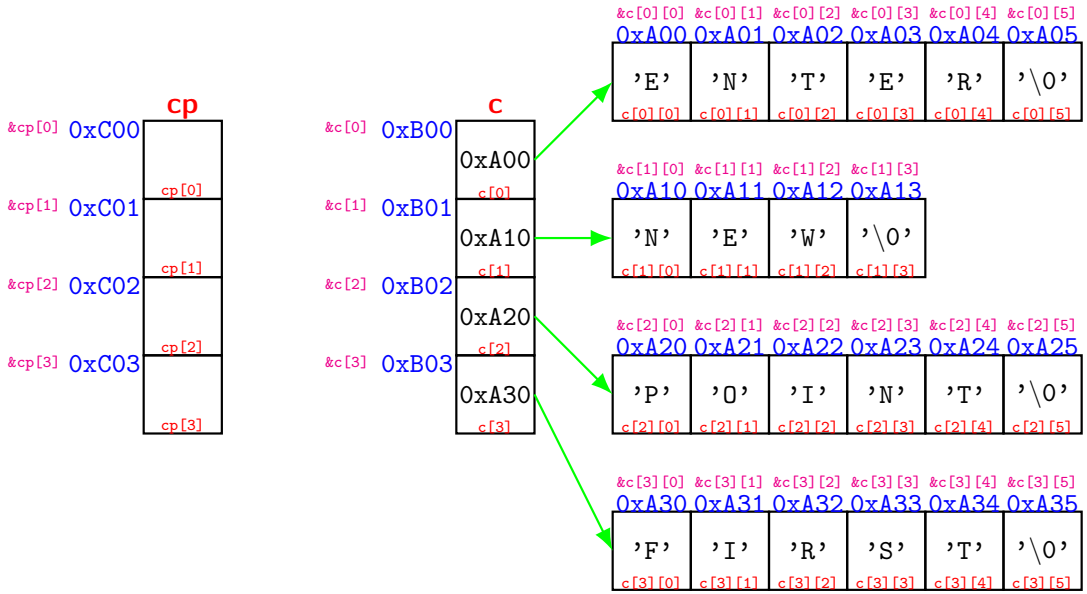

**char** *c[] = {"ENTER","NEW","POINT","FIRST"};

- c est un tableau de 4 (nombre d'éléments de la liste) pointeurs de char.
- c[0] contient l'adresse du premier élément de la chaine "ENTER",
- c[1] contient l'adresse du premier élément de la chaine "NEW",
- c[2] contient l'adresse du premier élément de la chaine "POINT",
- c[3] contient l'adresse du premier élément de la chaine "FIRST",



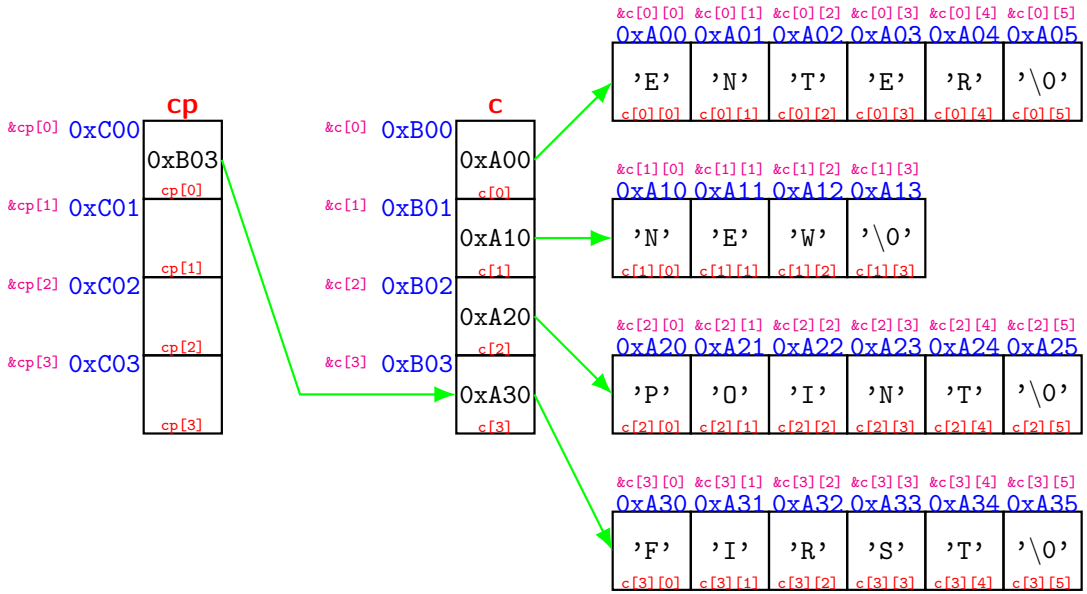

**char** **cp[] = {c+3,c+2,c+1,c};

- cp est un ?



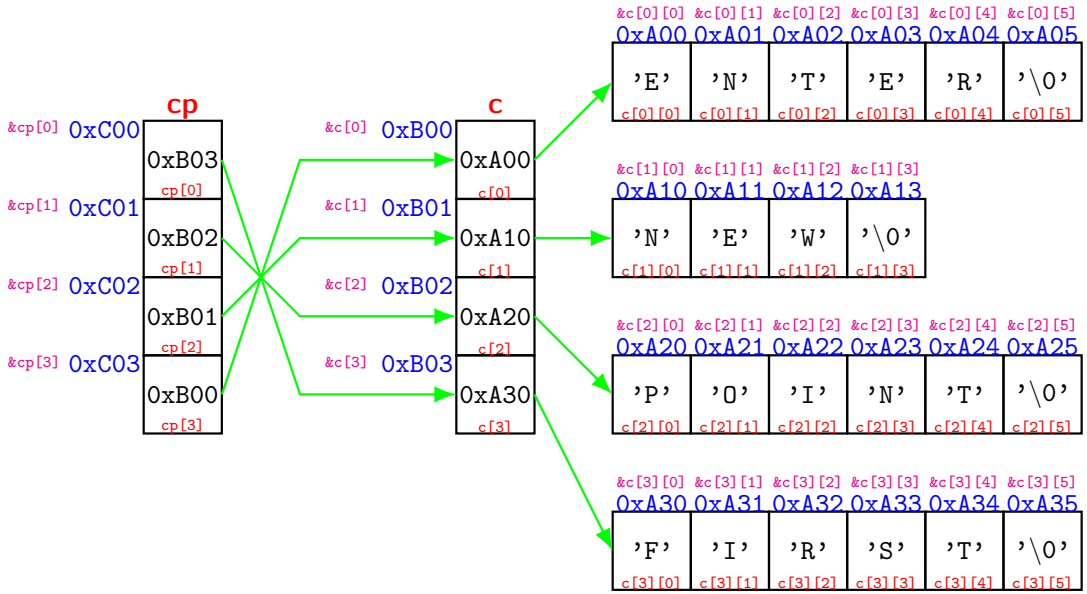

**char** $**cp[] = \{c+3,c+2,c+1,c\};$

- cp est un tableau de 4 pointeurs de pointeurs de char.
- cp[0] contient ?



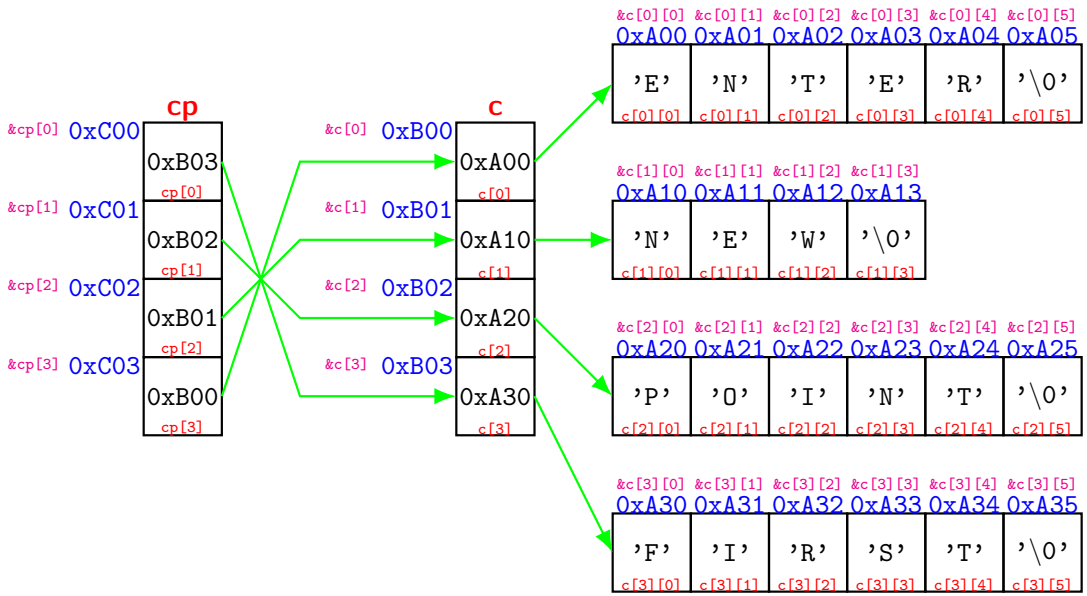

**char** $**cp[] = \{c+3,c+2,c+1,c\};$

- cp est un tableau de 4 pointeurs de pointeurs de char.
- cp[0] contient c+3 $\Longleftrightarrow$ &c[0]+3*sizeof(char **) $\Longleftrightarrow$ &c[3].
- cp[1] contient ?



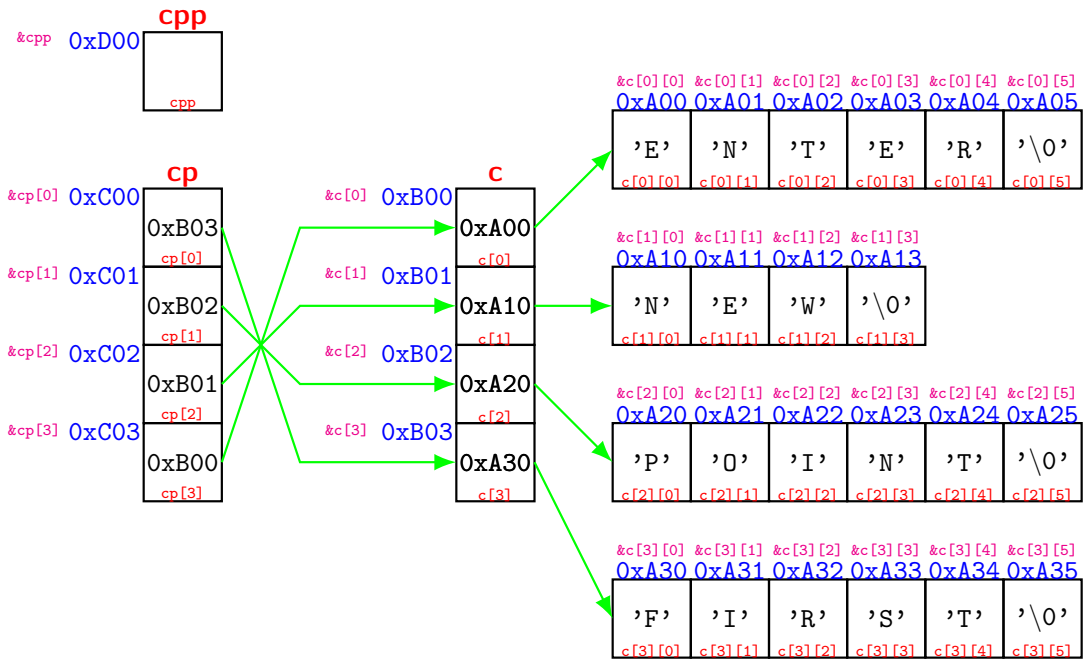

**char** $**cp[] = \{c+3,c+2,c+1,c\};$

- cp est un tableau de 4 pointeurs de pointeurs de char.
- cp[0] contient c+3 $\Longleftrightarrow$ &c[0]+3∗sizeof(char **) $\Longleftrightarrow$ &c[3].
- cp[1] contient c+2 $\Longleftrightarrow$ &c[0]+2∗sizeof(char **) $\Longleftrightarrow$ &c[2].
- cp[2] contient c+1 $\Longleftrightarrow$ &c[0]+sizeof(char **) $\Longleftrightarrow$ &c[1].
- cp[3] contient c $\Longleftrightarrow$ &c[0].



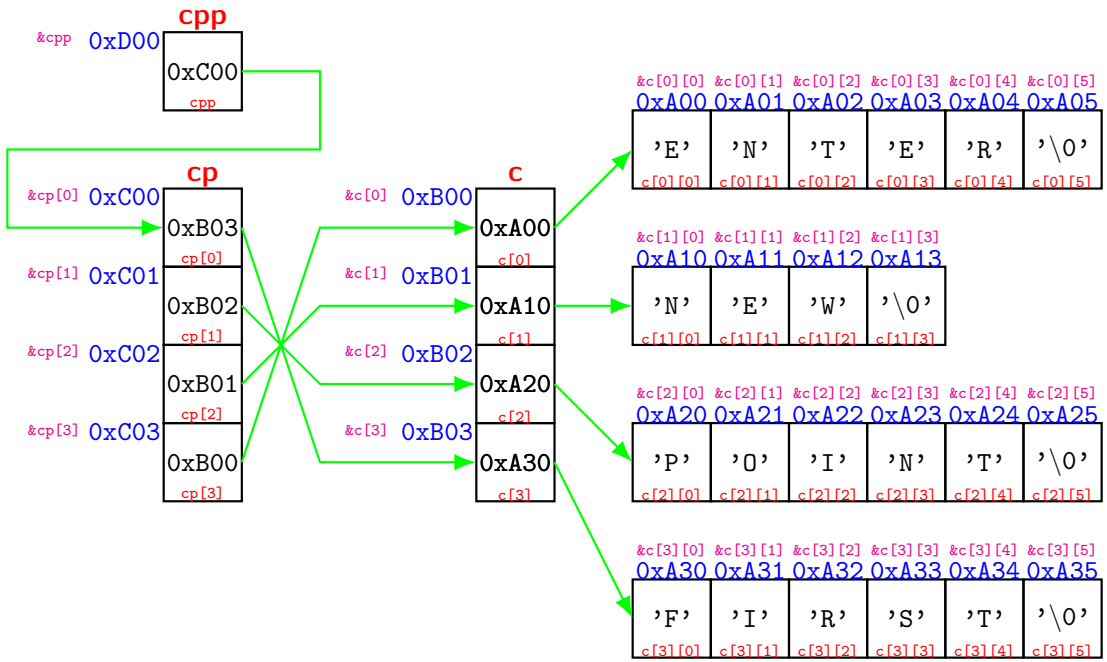

## char ***cpp = cp;

- cpp est un ?



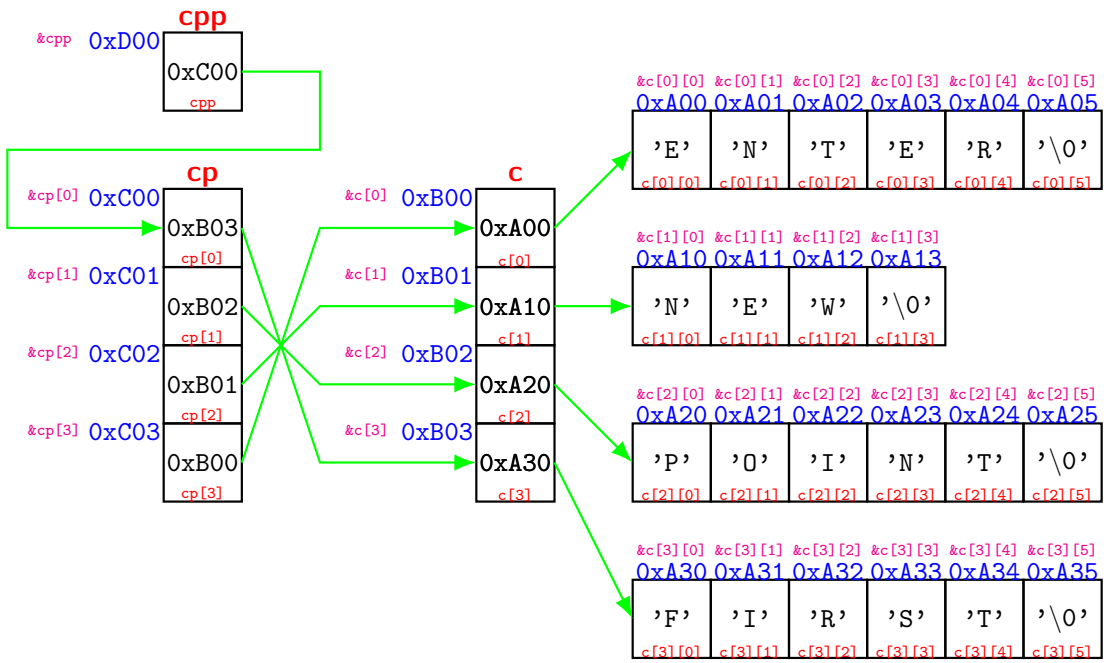

## char ***cpp = cp;

- cpp est un pointeur de pointeur de pointeur de char.
- cpp contient ?

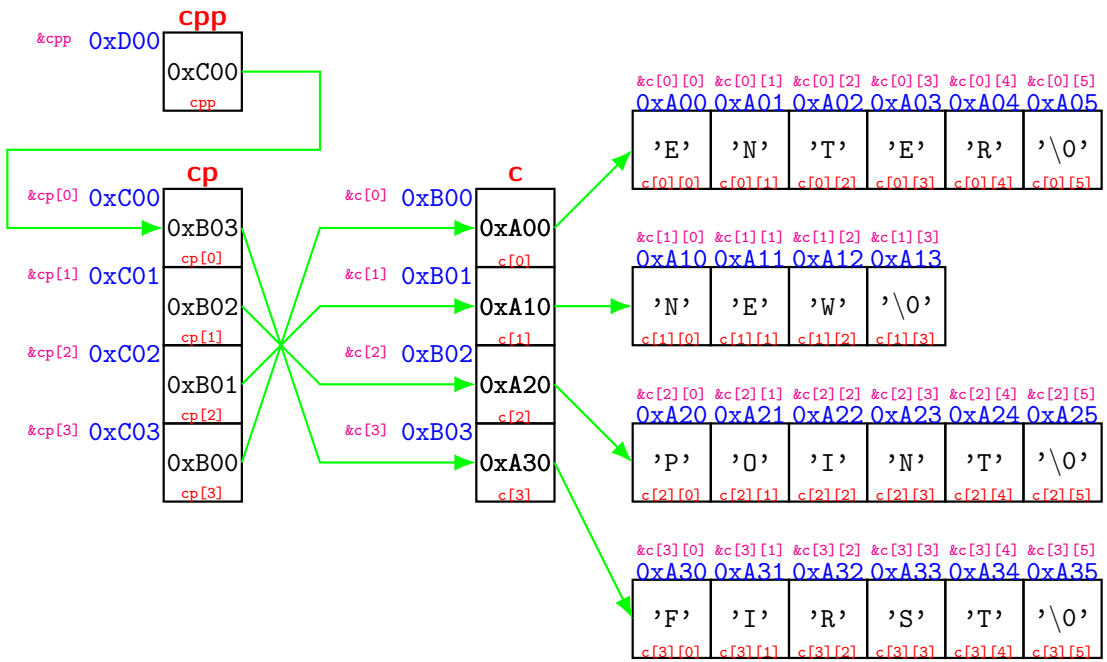



## char ***cpp = cp;

- cpp est un pointeur de pointeur de pointeur de char.
- cpp contient cp $\Longleftrightarrow$ &cp[0].

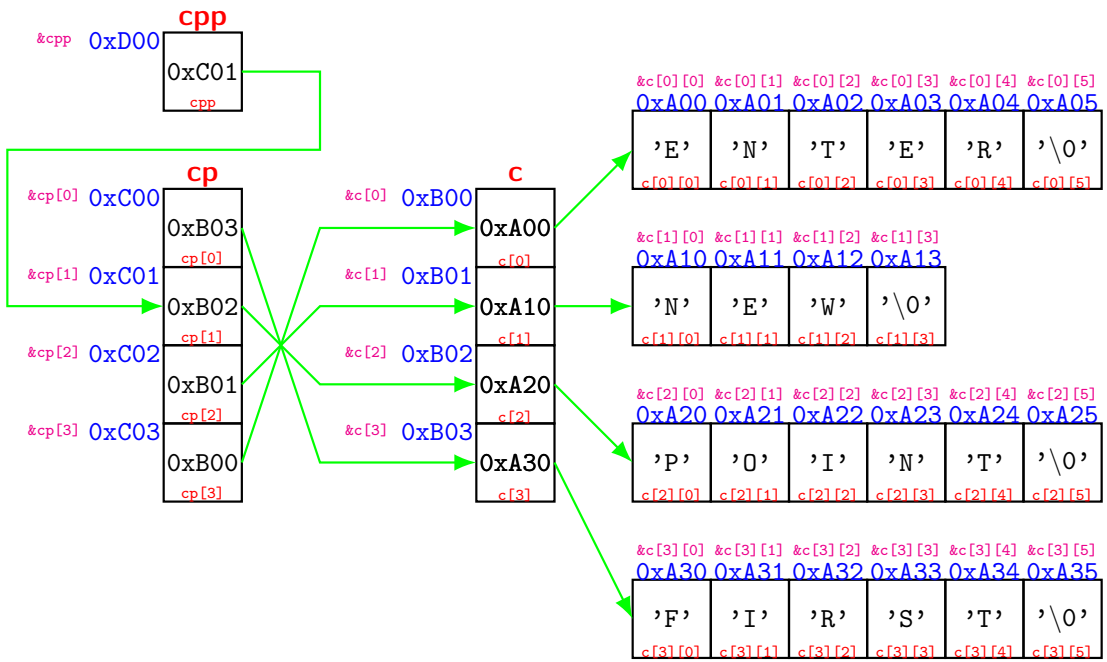



printf ("%s",**++cpp); ⇑ *mémoire avant instruction* ⇑



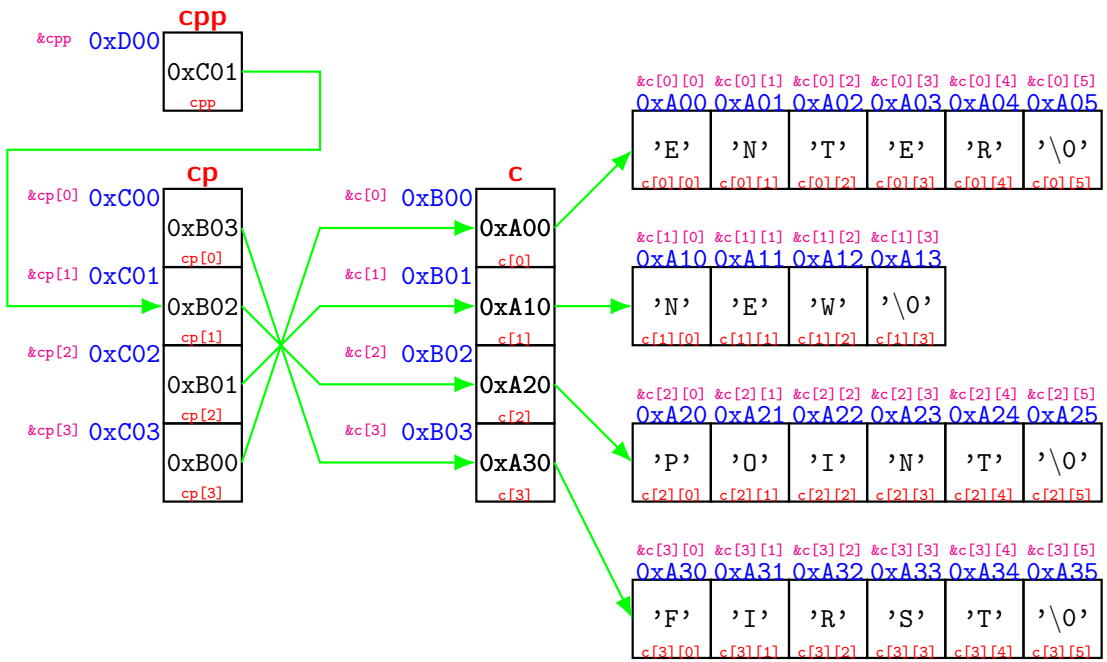

printf ("%s",**++cpp); ⇑ *mémoire avant instruction* ⇑

On évalue tout d'abord l'expression **++cpp qui est équivalente à **(++cpp). ++cpp est évalué avant le calcul de l'expression totale et correspond à cpp=cpp+1, c'est à dire cpp=&cp[0]+1=&cp[1].
Ensuite, *++cpp correspond à *(&cp[1])=cp[1]=&c[2]. Puis **++cpp correspond à *(&c[2])=c[2] qui est l'adresse de la chaine pointée par c[2] c'est à dire "POINT".
En résumé, cpp prend la valeur &cp[1] et on affiche la chaine "POINT".

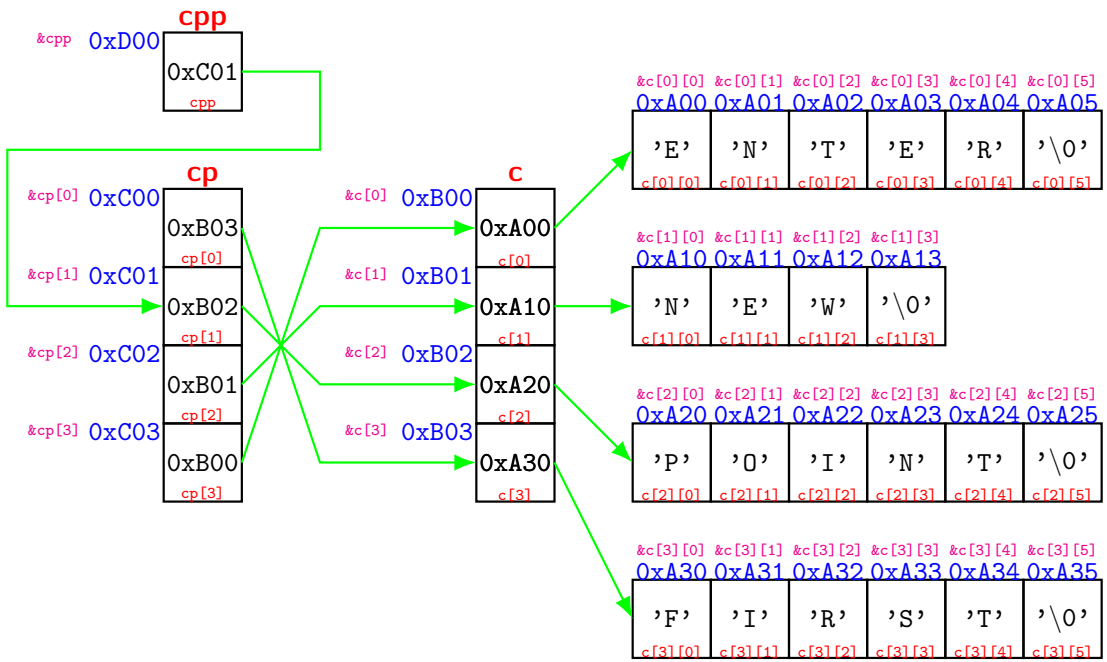



printf ("%s",**++cpp); ⇑ *mémoire après instruction* ⇑

On évalue tout d'abord l'expression **++cpp qui est équivalente à **(++cpp). ++cpp est évalué avant le calcul de l'expression totale et correspond à cpp=cpp+1, c'est à dire cpp=&cp[0]+1=&cp[1].
Ensuite, *++cpp correspond à *(&cp[1])=cp[1]=&c[2]. Puis **++cpp correspond à *(&c[2])=c[2] qui est l'adresse de la chaine pointée par c[2] c'est à dire "POINT".
En résumé, cpp prend la valeur &cp[1] et on affiche la chaine "POINT".



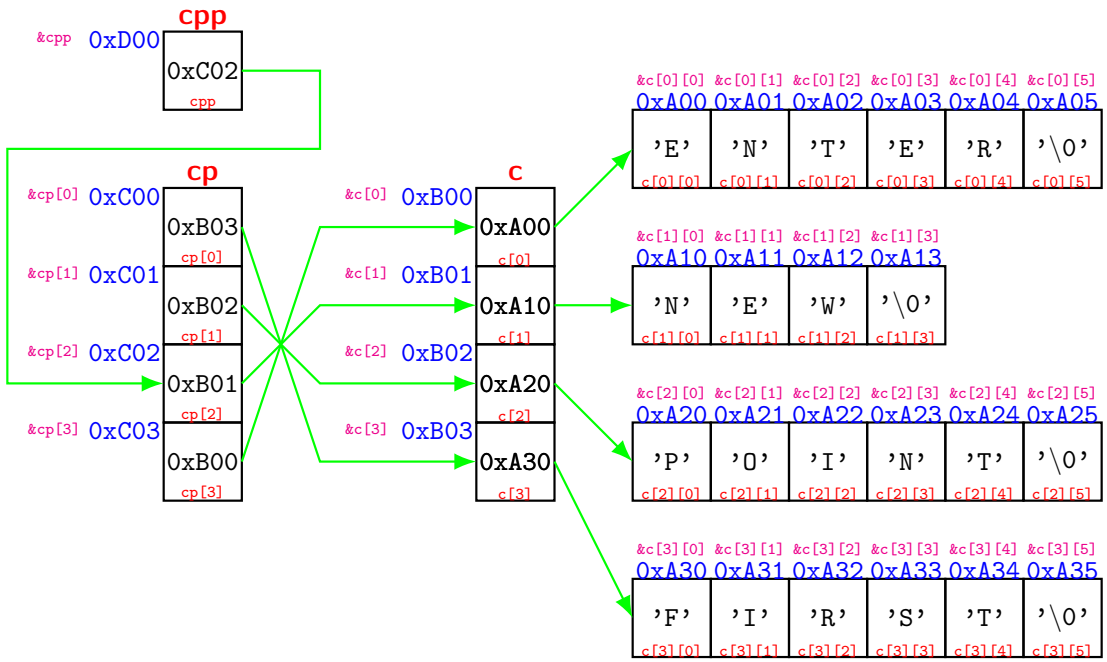

printf ("%s",*--*++cpp+3); ⇑ *mémoire avant instruction* ⇑




printf ("%s",*--*++cpp+3); ⇑ *mémoire avant instruction* ⇑

$$*--*++cpp+3 \iff (*(--(*(++cpp))))+3$$

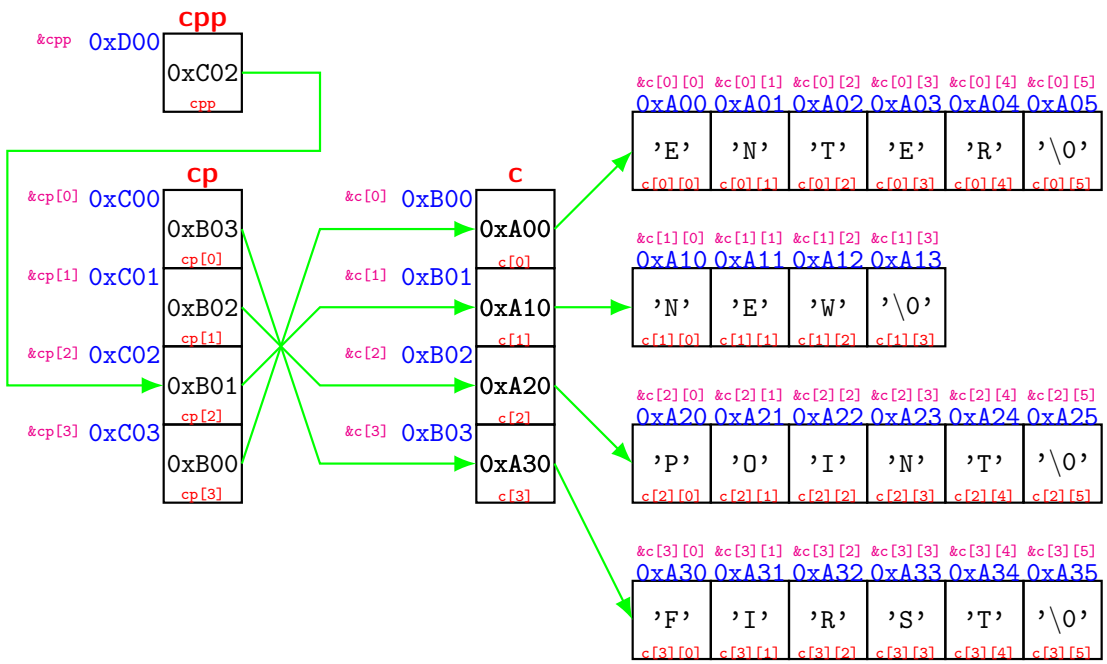
++cpp : on incrémente cpp, puis on l'évalue. cpp=&cp[1]+1=&cp[2].




printf ("%s",*--*++cpp+3); ⇑ *Après évaluation de* *++cpp* ⇑

$$*--*++cpp+3 \iff (*(--(*(++cpp))))+3$$

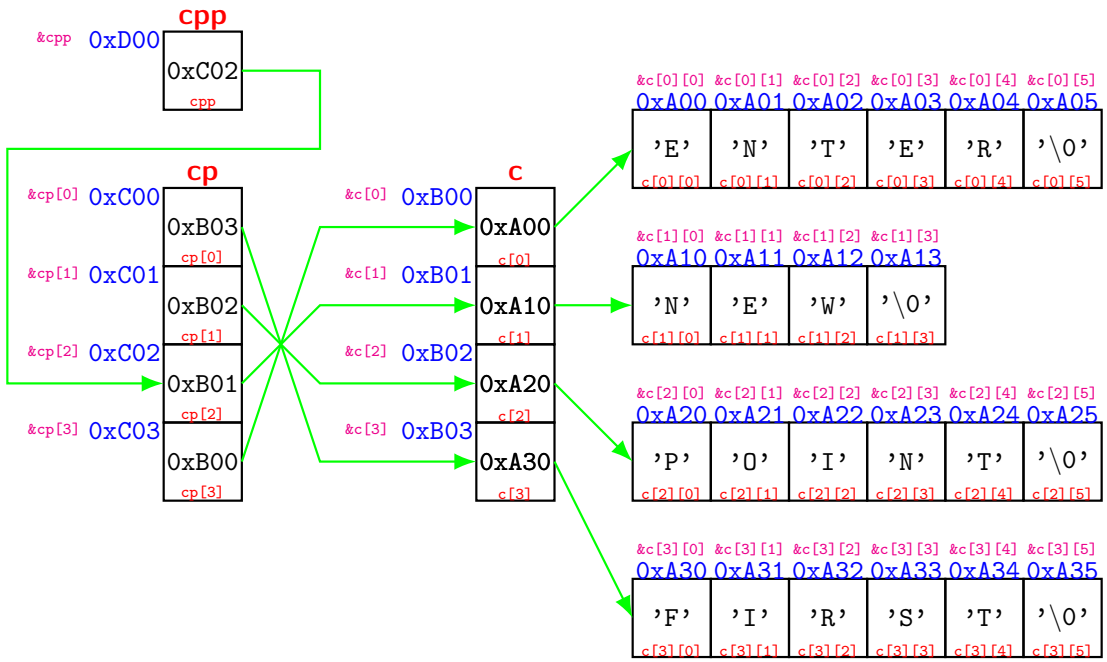
++cpp : on incrémente cpp, puis on l'évalue. cpp=&cp[1]+1=&cp[2].




printf ("%s",*--*++cpp+3); ⇑ *Après évaluation de* *++cpp* ⇑

$$*--*++cpp+3 \iff (*(--(*(++cpp))))+3$$

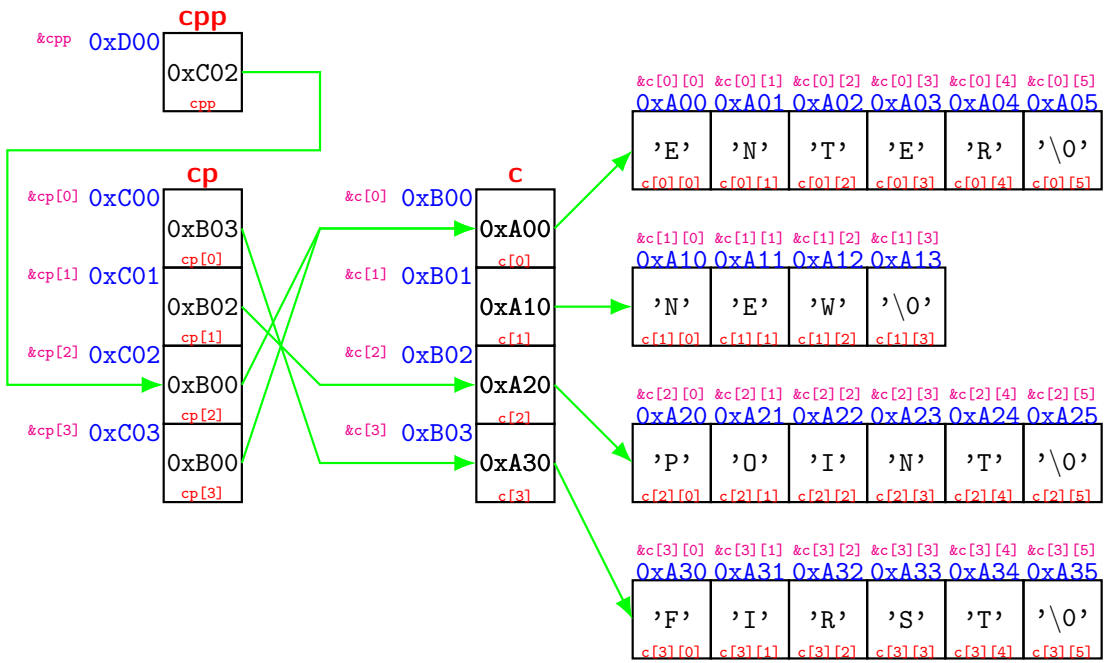
$*(++cpp)$ correspond à la valeur pointée par $++cpp$. $*(++cpp) \iff cp[2]$.




printf ("%s",*−−*++cpp+3); ⇑ *Après évaluation de* *++cpp* ⇑

$$*--*++cpp+3 \iff (*(--(*(++cpp))))+3$$

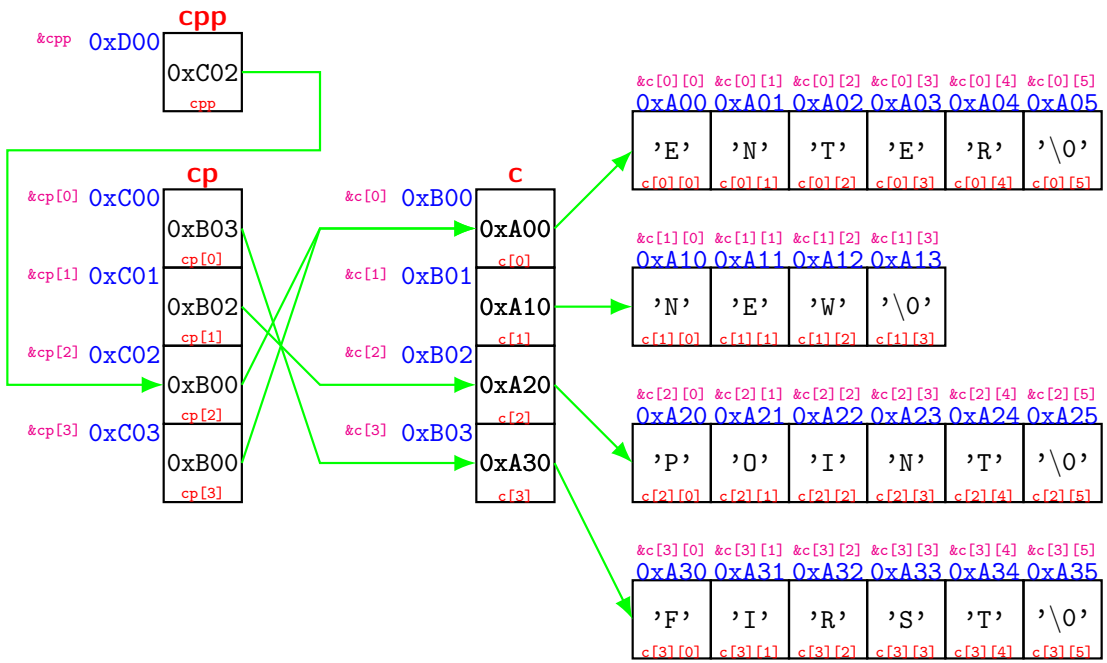
$--(*(++cpp)) \iff --(cp[2])$ : on décrémente $cp[2]$, puis on l'évalue.
$--(cp[2])=\&c[1]-1)=\&c[0]$.




printf ("%s",*--*++cpp+3); ⇑ *Après évaluation de* *--*++cpp* ⇑

$$*--*++cpp+3 \iff (*(--(*(++cpp))))+3$$

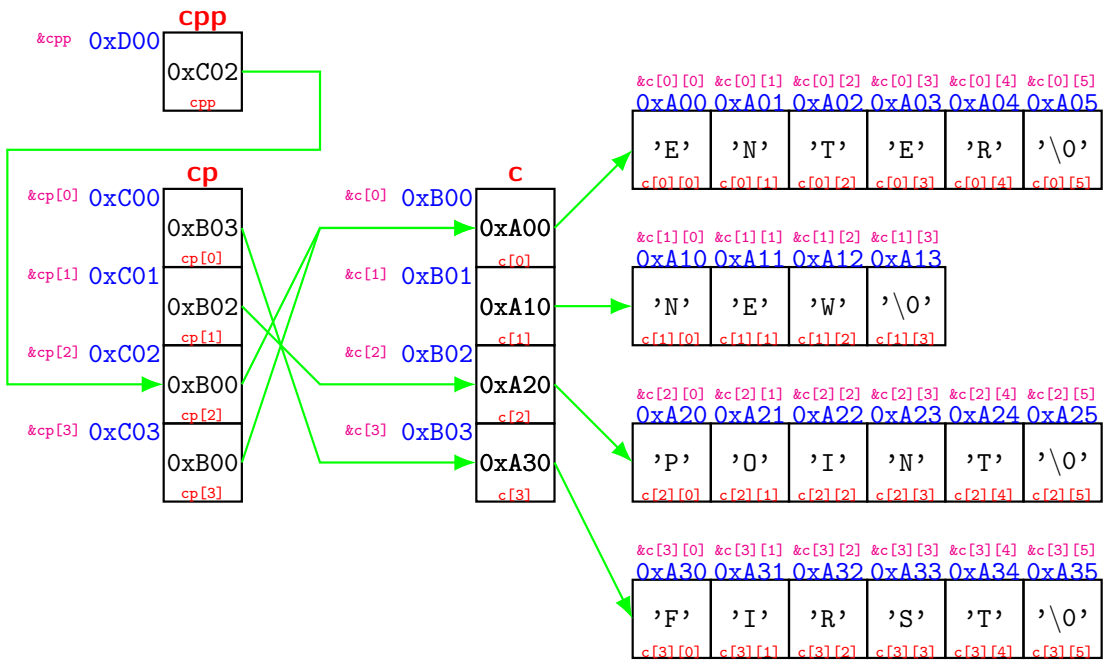
$--(*(++cpp)) \iff --(cp[2])$ : on décrémente $cp[2]$, puis on l'évalue.
$--(cp[2])=\&c[1]-1)=\&c[0]$.




printf ("%s",*--*++cpp+3); ⇑ *Après évaluation de* *--*++cpp* ⇑

$$*--*++cpp+3 \iff (*(--(*(++cpp))))+3$$

$(*(--(*(++cpp)))) \iff (**cpp)+3 \iff c[0]+3 \iff \&c[0][0]+3$
$\iff \&c[0][3]$.
on affiche donc la chaine "ER".




printf ("%s",*cpp[−2]+3); ⇑ *mémoire avant instruction* ⇑

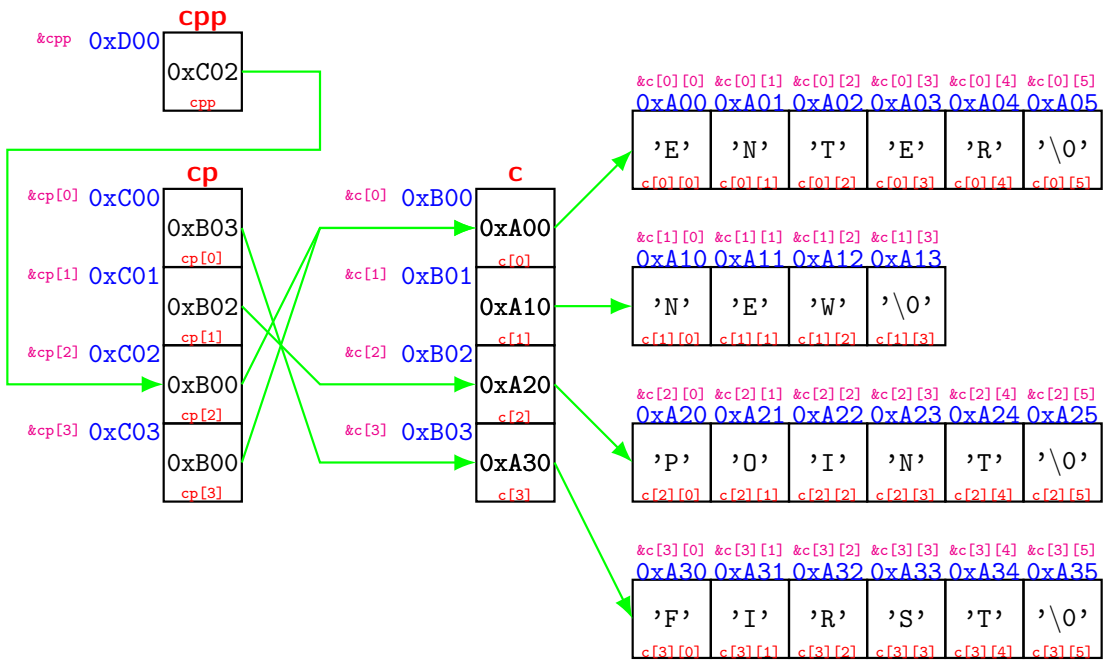
$$\boxed{*cpp[-2]+3 \iff (*(cpp[-2]))+3}$$

$cpp[-2] \hookrightarrow *(cpp-2) \iff *(\&cp[2]-2) \iff *(\&cp[0]) \iff cp[0] \iff \&c[3]$

$*(cpp[-2]) \iff *(\&c[3]) \iff c[3]$

$*(cpp[-2])+3 \iff c[3]+3 \iff \&c[3][0]+3 \iff \&c[3][3]$

on affiche donc la chaine "ST".




printf ("%s",*cpp[−2]+3); ⇑ *mémoire après instruction* ⇑

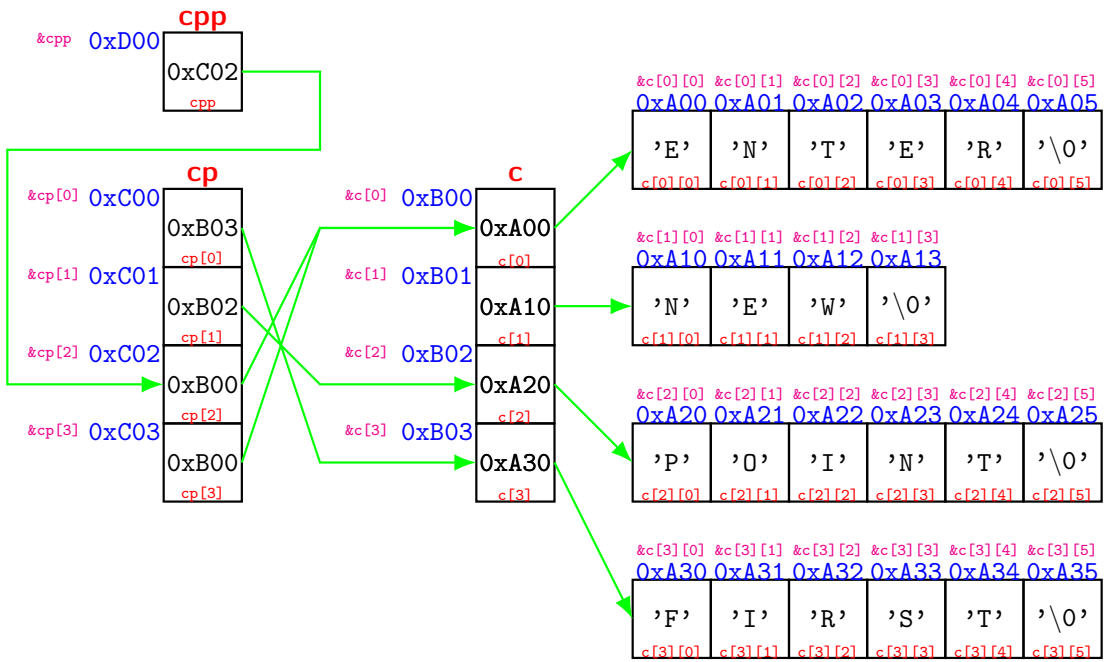
$$\boxed{*cpp[-2]+3 \iff (*(cpp[-2]))+3}$$

$cpp[-2] \hookrightarrow *(cpp-2) \iff *(\&cp[2]-2) \iff *(\&cp[0]) \iff cp[0] \iff \&c[3]$

$*(cpp[-2]) \iff *(\&c[3]) \iff c[3]$

$*(cpp[-2])+3 \iff c[3]+3 \iff \&c[3][0]+3 \iff \&c[3][3]$

on affiche donc la chaine "ST".




printf ("%s\n",cpp[−1][−1]+1); ⇑ *mémoire avant instruction* ⇑

$$\boxed{\texttt{cpp}[-1][-1]+1 \iff ((\texttt{cpp}[-1])[-1])+1}$$

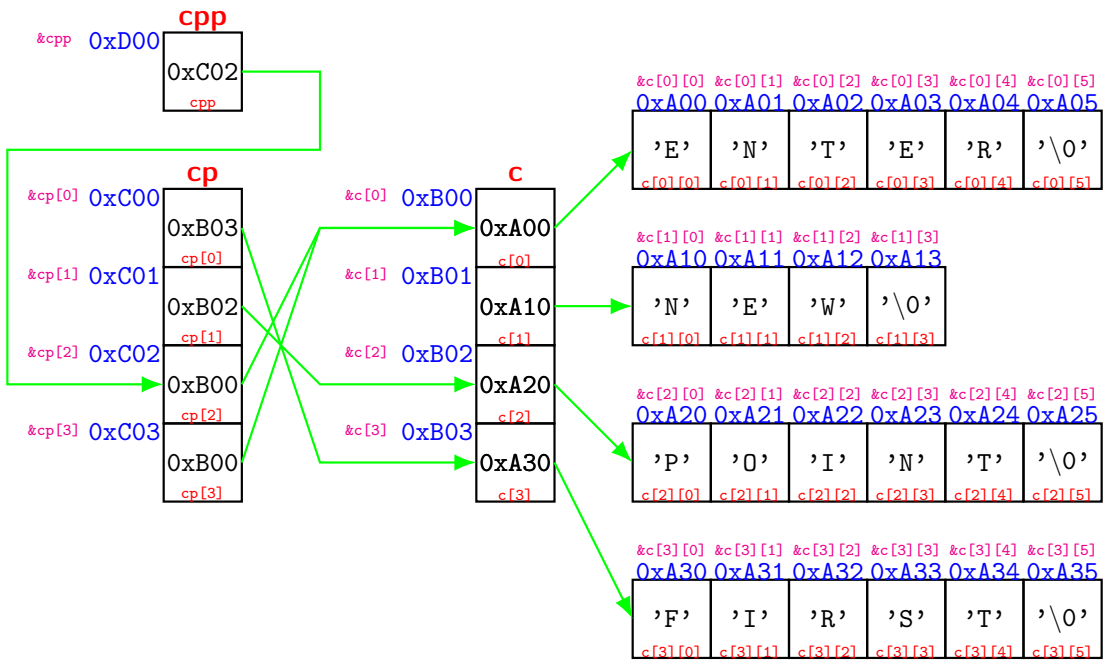
$\texttt{cpp}[-1] \hookrightarrow *(\texttt{cpp}-1) \iff *(\&\texttt{cp}[2]-1) \iff *(\&\texttt{cp}[1]) \iff \texttt{cp}[1] \iff \&\texttt{c}[2]$

$(\texttt{cpp}[-1])[-1] \iff (\texttt{cp}[1])[-1] \hookrightarrow *(\texttt{cp}[1]-1) \iff *(\&\texttt{c}[2]-1)$

$\iff *(\&\texttt{c}[1]) \iff \texttt{c}[1]$

$((\texttt{cpp}[-1])[-1])+1 \iff \texttt{c}[1]+1 \iff \&\texttt{c}[1][0]+1 \iff \&\texttt{c}[1][1]$

on affiche donc la chaine "EW" et on passe à la ligne.




printf ("%s\n",cpp[−1][−1]+1); ⇑ *mémoire après instruction* ⇑

$$\boxed{\texttt{cpp}[-1][-1]+1 \iff ((\texttt{cpp}[-1])[-1])+1}$$

$\texttt{cpp}[-1] \hookrightarrow *(\texttt{cpp}-1) \iff *(\&\texttt{cp}[2]-1) \iff *(\&\texttt{cp}[1]) \iff \texttt{cp}[1] \iff \&\texttt{c}[2]$

$(\texttt{cpp}[-1])[-1] \iff (\texttt{cp}[1])[-1] \hookrightarrow *(\texttt{cp}[1]-1) \iff *(\&\texttt{c}[2]-1)$

$\iff *(\&\texttt{c}[1]) \iff \texttt{c}[1]$

$((\texttt{cpp}[-1])[-1])+1 \iff \texttt{c}[1]+1 \iff \&\texttt{c}[1][0]+1 \iff \&\texttt{c}[1][1]$

on affiche donc la chaine "EW" et on passe à la ligne.