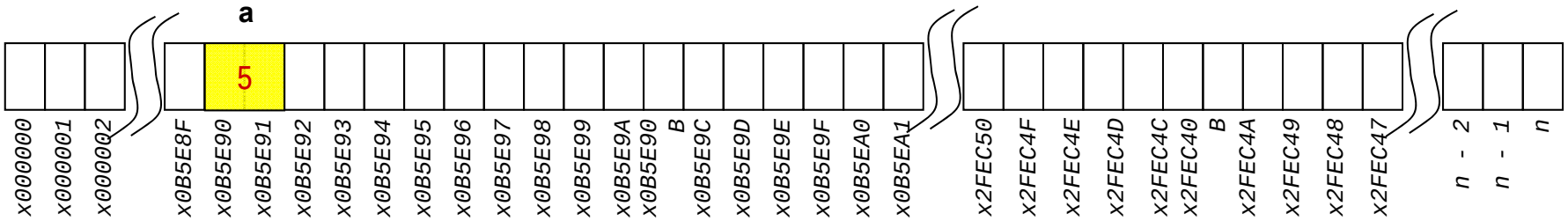
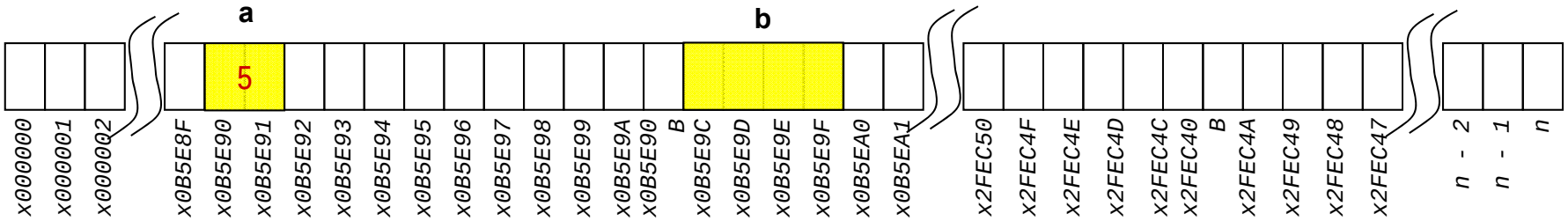
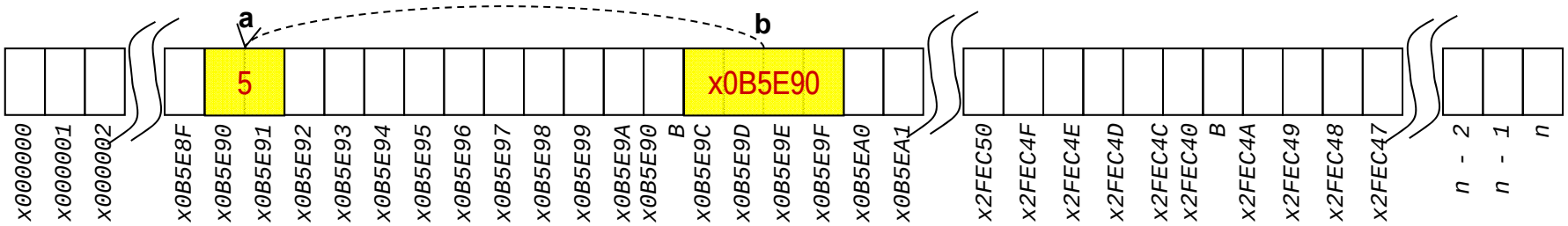
Punteros



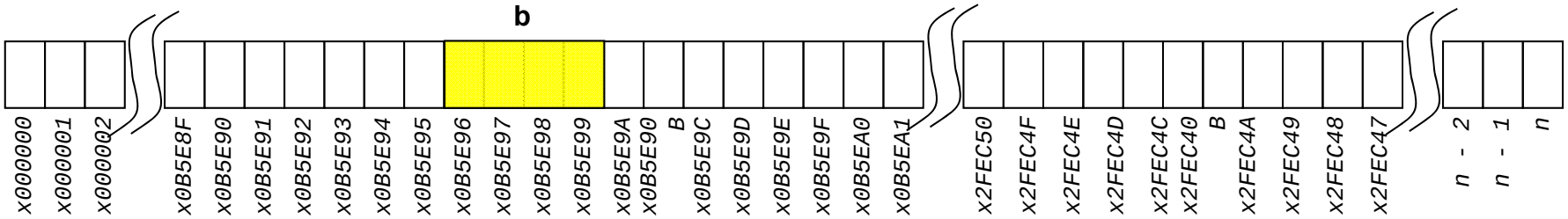

```
înt main () {
      int a;
      a = 5;

      printf (“%d”, a); // imprime: 5
      printf (“%x”, &a); // imprime: 0B5E90

      }
```



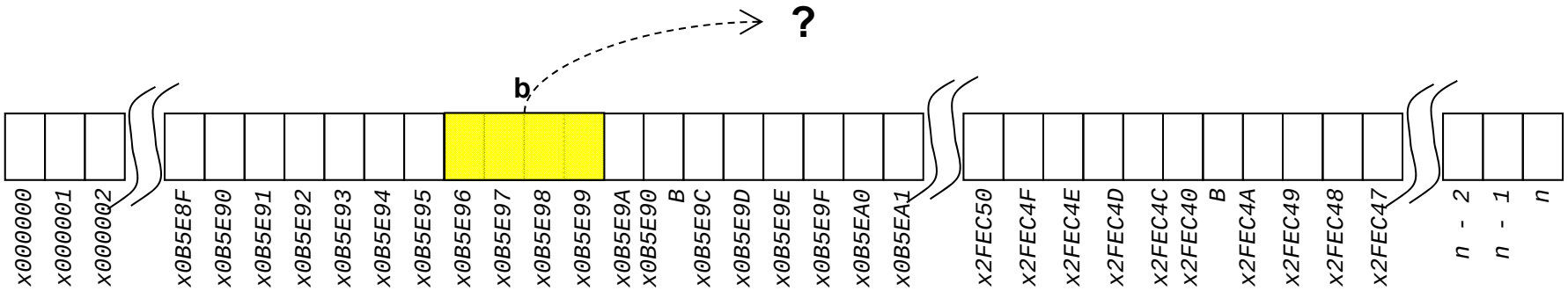

```
înt main () {
      int a;
      a = 5;

      int *b;
      .
      .
      .
      }
```




```
înt main () {
      int a;
      a = 5;

      int *b;

      b = &a;

      printf (“%d”, a); // imprime: 5
      printf (“%x”, &a); // imprime: B5E90

      printf (“%x”, b); // imprime: B5E90
      printf (“%d”, *b); // imprime: 5

      printf (“%x”, &b); // imprime:B5E9C
      }
```

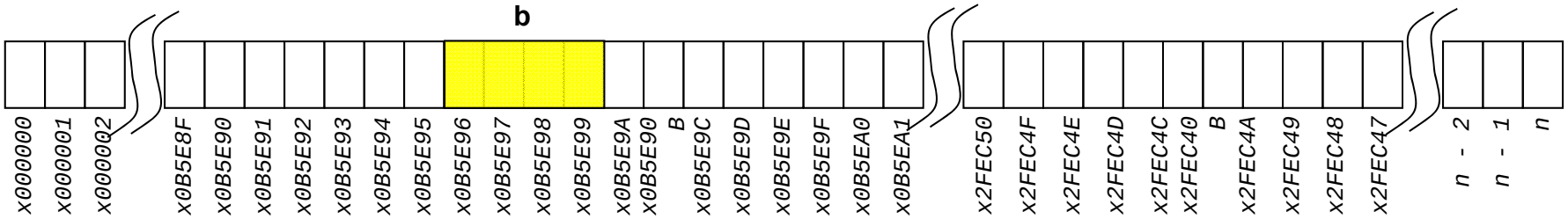



```
înt main () {
        int *b;
        .
        .
        .
        }
```

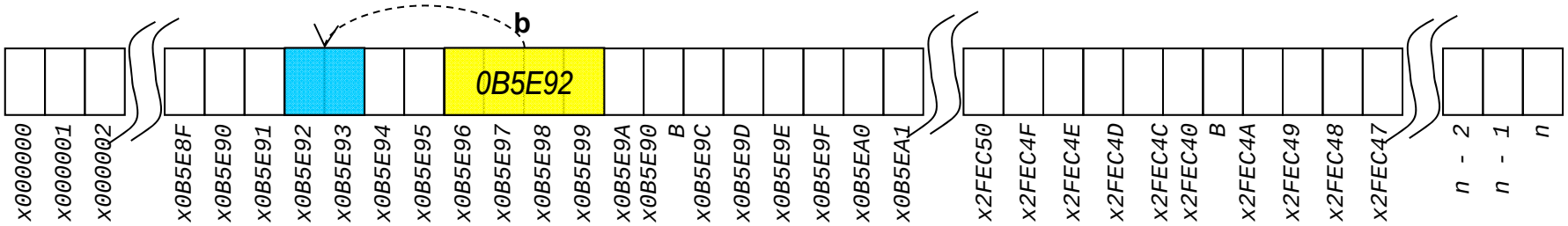



```
înt main () {
      int *b;

      printf (“%x”, b); // imprime: ??????


      }
```

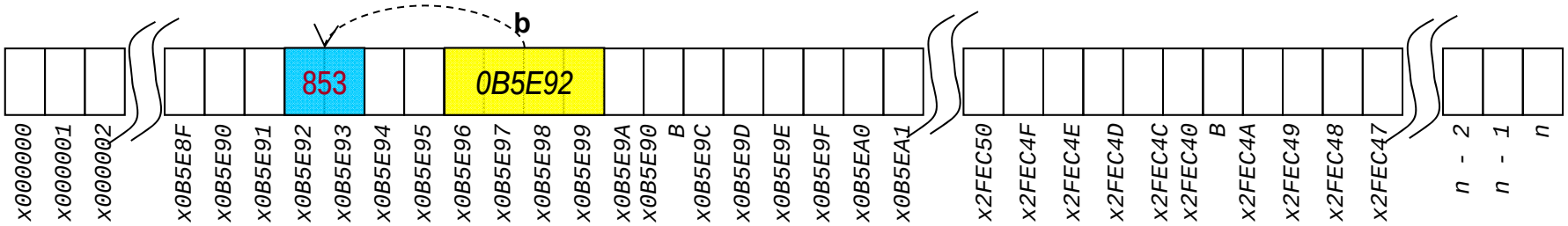
1ª Regla de oro: todo apuntador DEBE ser inicializado, si no se tiene una dirección válida debe tomar el valor de una dirección inválida (NULL)



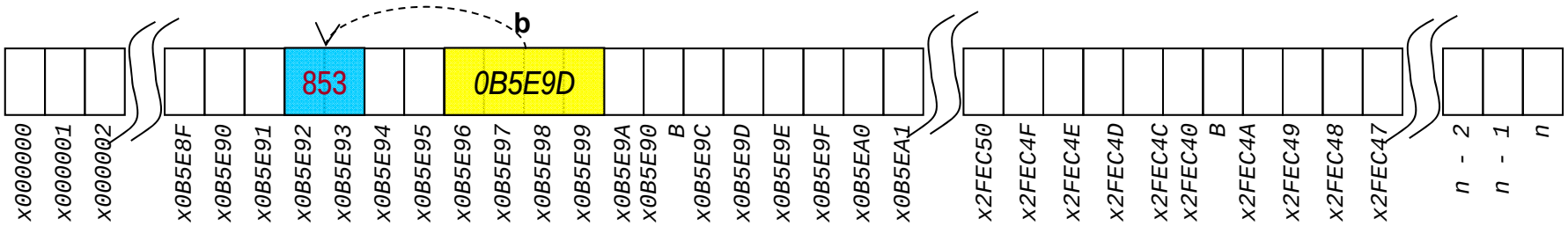

```
înt main () {
      int *b;

      b = (int*)malloc(sizeof(int));
      .
      .
      .
      }
```



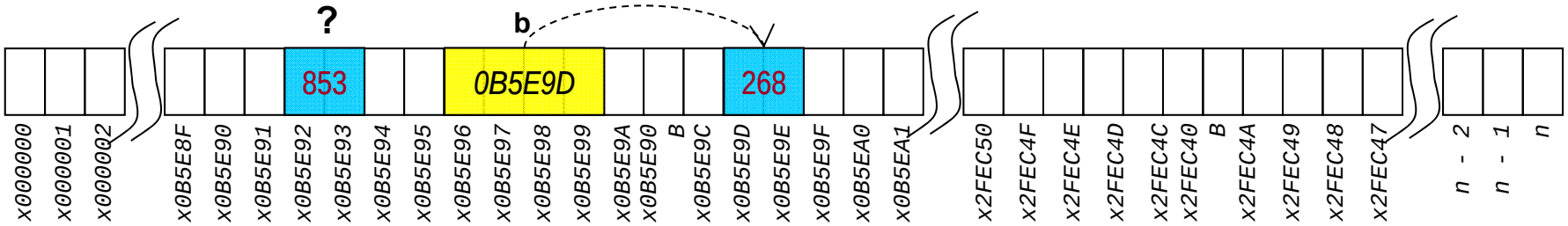

```
înt main () {
      int *b;

      b = (int*)malloc(sizeof(int));
      .
      .
      .
      }
```



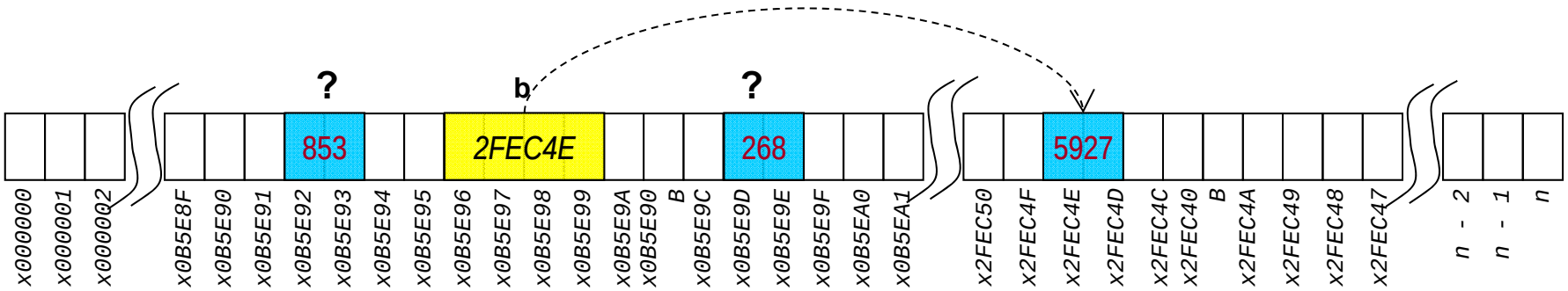

```
înt main () {
      int *b;

      b = (int*)malloc(sizeof(int));

      *b = 853;

      printf (“%x”, b); // imprime: B5E92
      printf (“%d”, *b); // imprime: 853
      .
      .
      .
      }
```




```
înt main () {
        int *b;

        b = (int*)malloc(sizeof(int));
        *b = 853;

        printf (“%x”, b); // imprime: B5E92
        printf (“%d”, *b); // imprime: 853
..
        }
```

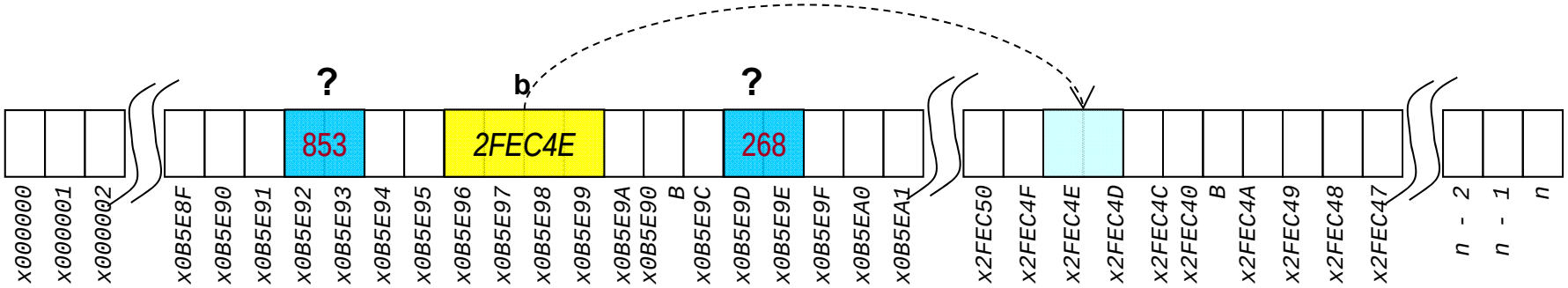



```
înt main () {
        int *b;

        b = (int*)malloc(sizeof(int));
        *b = 853;

        printf (“%x”, b); // imprime: B5E92
        printf (“%d”, *b); // imprime: 853
.
        b = (int*)malloc(sizeof(int));
        *b = 268;
.
        }
```

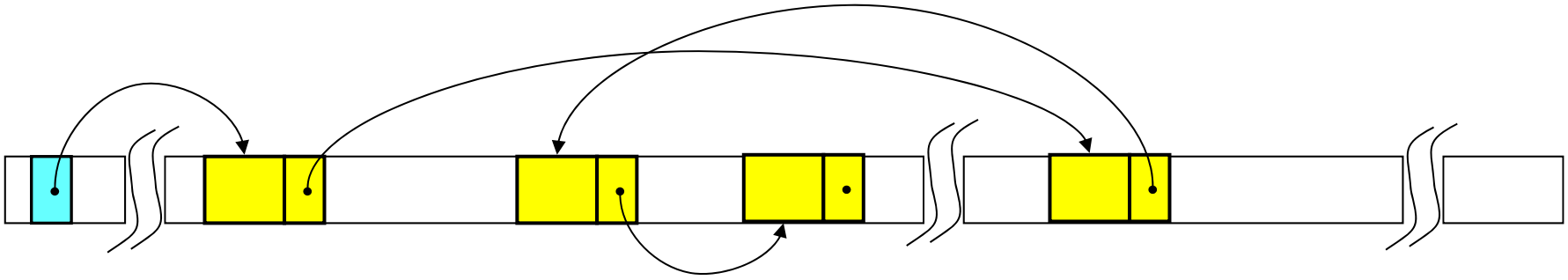



```
înt main () {
      int *b;

      b = (int*)malloc(sizeof(int));
      *b = 853;

      printf (“%x”, b); // imprime: B5E92
      printf (“%d”, *b); // imprime: 853
.
      b = (int*)malloc(sizeof(int));
      *b = 268;
.
      b = (int*)malloc(sizeof(int));
      *b = 5927;
.
      }
```

2ª Regla de oro: Debe liberarse el espacio de memoria de todo elemento creado dinámicamente, cuando éste deja de ser necesario.
La aplicación de esta regla exige luego la aplicación de la 1ª regla de oro.




```
înt main () {
      int *b;

      b = (int*)malloc(sizeof(int));
      *b = 853;

      printf (“%x”, b); // imprime: B5E92
      printf (“%d”, *b); // imprime: 853
.
      b = (int*)malloc(sizeof(int));
      *b = 268;
.
      b = (int*)malloc(sizeof(int));
      *b = 5927;

      free (b);
      }
```



Creando nodos

```
typedef __int__ tipo_dato;

typedef struct tipo_nodo{
      tipo_dato elem;
      struct tipo_nodo *sig;
      } tipo_nodo;
int main() {
      tipo_nodo* a;
      a = (tipo_nodo*)malloc (sizeof (tipo_nodo));
      a->elem = “10”;
      a->sig = (tipo_nodo*)malloc (sizeof (tipo_nodo));
      a->sig->elem = “20”;
      a->sig->sig = (tipo_nodo*)malloc (sizeof (tipo_nodo));
      a->sig->sig->elem = “30”;
      a->sig->sig->sig = NULL;
      printf (%d, a->elem);
      printf (%d, a->sig->elem);
      printf (%d, a-> sig->sig->elem);
```

free (a): // sólo destruye el primer nodo

free (a->sig->sig); // se destruye sólo el ultimo
a->sig->sig = NULL;

free (a->sig->sig);
free (a->sig);
free (a);

a = NULL;