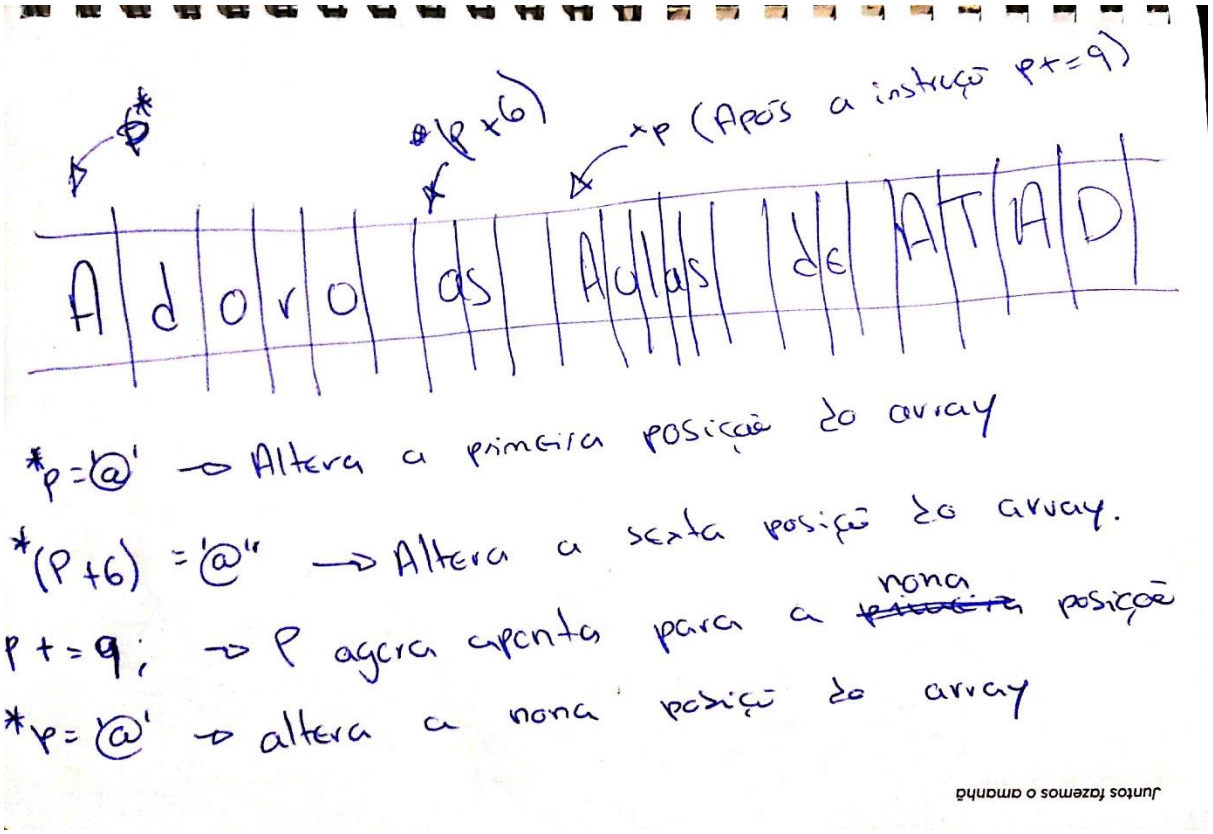
Ex. 8

A função ‘sum’ está a retornar o endereço de uma variável local, ou seja, uma variável que só existe no escopo da função ‘sum’. Isto significa que quando a função terminar, a variável já não existe em qualquer contexto, e está a retornar um endereço que já não é válido neste programa.

Ex. 10

O programa retorna: @doro @s @ul@s de @TAD

A ilustração do bloco de memória do array s.



A instrução *p = ‘@’ altera a primeira posição do array para ‘@’ visto *p apontar sempre para a primeira posição do array.

Uma instrução *(p + n) vai aceder á n-ésima posição do array, sem alterar o valor para que *p aponta.

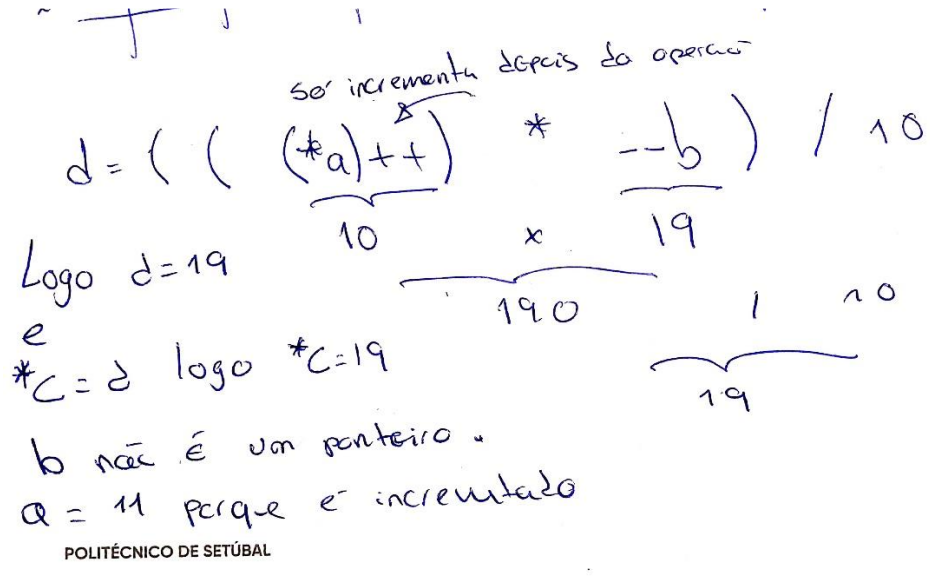
Uma instrução p += n altera a posição para que p aponta. Logo quando no programa do enunciado se executa “ p += 9 ”, p passa a apontar para s[8]. Assim a instrução seguinte *p = ‘@’ já não altera a primeira posição, mas sim a nona.

Ex.11

Apenas ‘a’ e ‘f’ são passados por referência por isso podemos ignorar o valor final de ‘b’, pois será 20 independentemente do que acontecer.

Ao calcular o valor da variável local ‘d’, o a é incrementado, logo o valor final de a será 11.

A explicação do que acontece à variável ‘d’:



Como no fim é atribuído o valor de ‘d’ a ‘*c’, a variável ‘f’ fica com o valor 19.

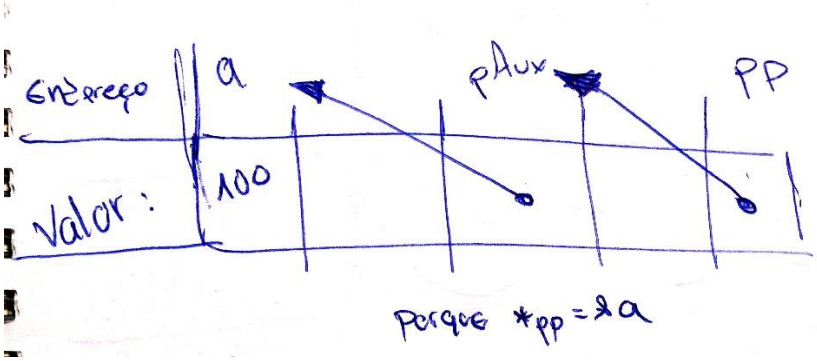
Ex.12

```
int a = 100;
int *pAux;
int **pp;
pp = &pAux;
*pp = &a;
printf("**p = %d\n", **pp);
```

O output será: **p = 100

Explicação:

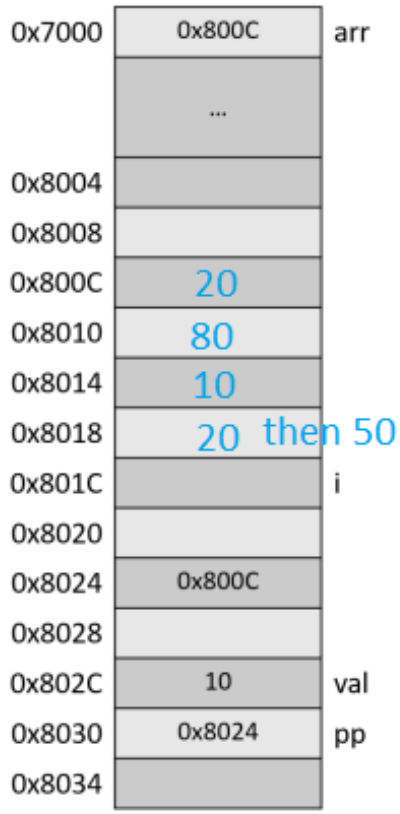
‘**pp’ é um ponteiro que aponta para ponteiros. ‘pp’ vai apontar para ‘pAux’ (pp = pAux), logo *pp é o valor de ‘pAux’, que neste caso fica um ponteiro para ‘a’ porque (*pp = &a)



Ex.13

```
int main(){

   (...)

   int arr[4], i, val = 10;
   int **pp;

   (...)

   **pp = 20;
   for(i = 1; i < 3; i++)
      (*pp)[i + 1] = val * i;
   *(arr + 1) = 80;
   *(*pp + 3) = 50;

    (...)

    return EXIT_SUCCESS;
}
```

a)



b)

**pp + 2 = 22

*pp + 2 = 0x800C + 2 = 0x8014

pp + 2 = 0x8038

(*pp)[3] = 50

&(*pp)[3] = 0x8018

&(*pp) = 0x8024

&(**pp) = 0x800C

*(arr+3) = 50

&*(arr+3) = 0x8018

Arr[3] = 50