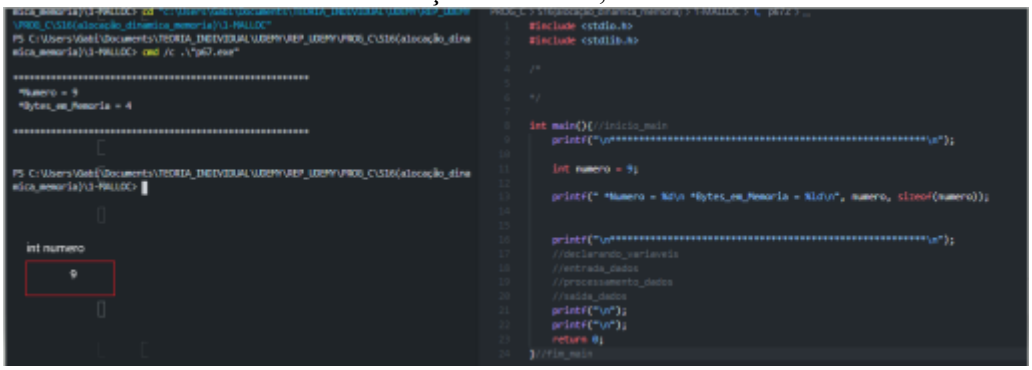
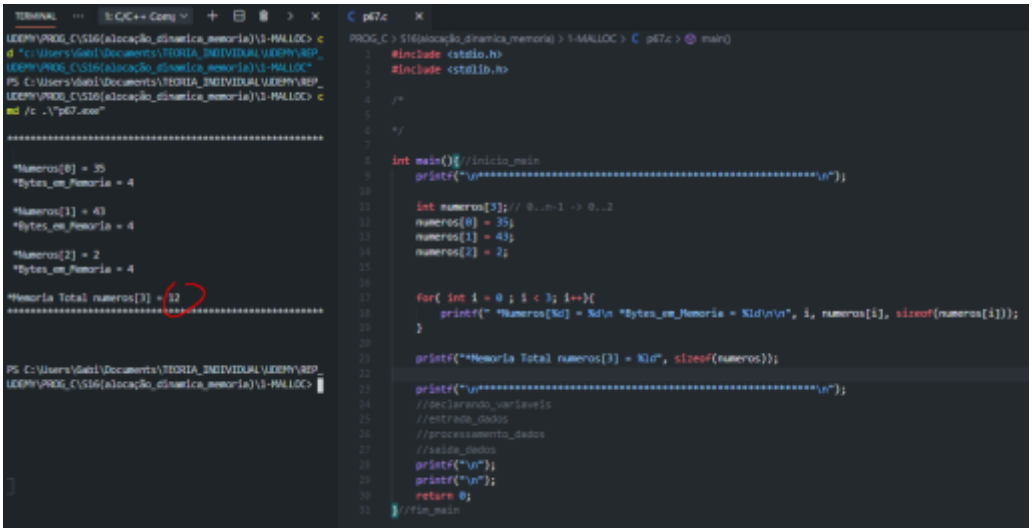
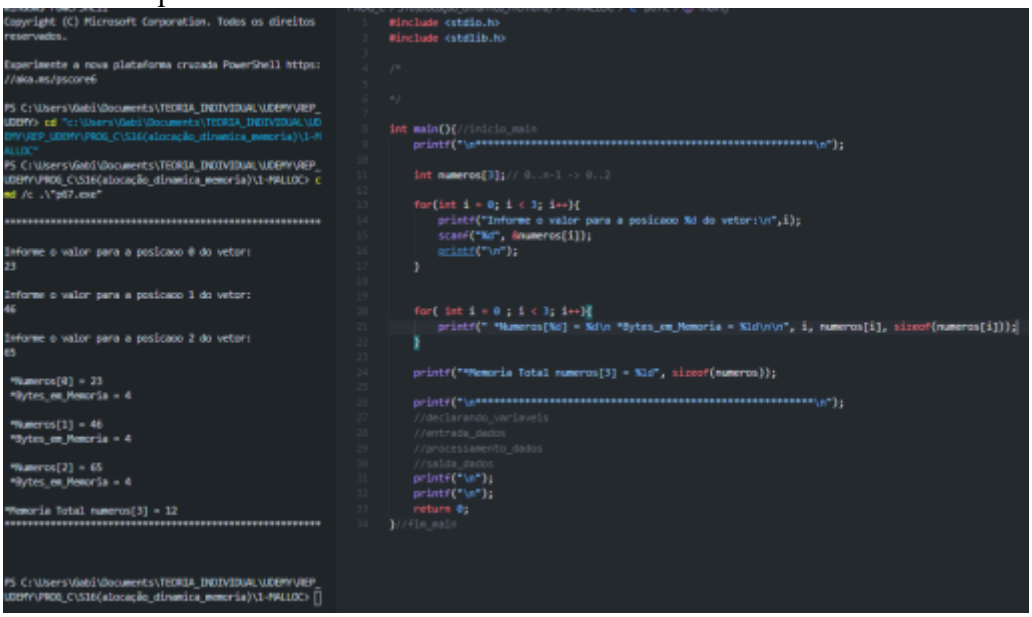
1-MALLOC

- Como estamos falando de alocação dinamica, temos estaticamente valores declarados.





- Nem sempre sabemos quantos valores haverá no vetor, talvez a gente precise perguntar ao usuario quantos valores ele quer informar.

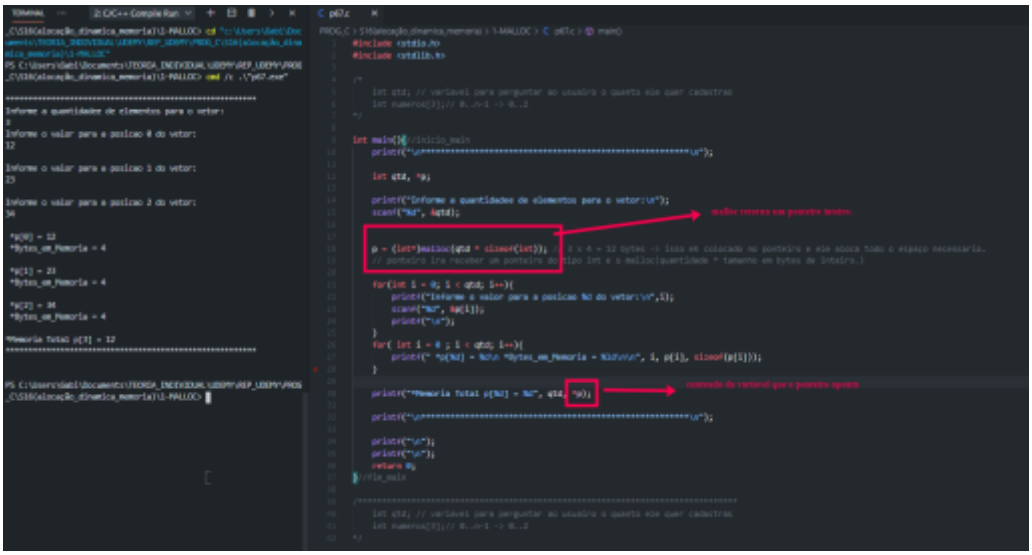


- Transformamos o processo em loop.

- Para utilizarmos essa alocação dinamica temos que importar uma outra biblioteca chamada <stdlib.h>
- Vamos tbm declara um ponteiro pois eh a partir dele que conseguimos fazer a alocação.
- Duas maneiras dee fazer...

```c
int main(){//inicio_main
    printf("\n*********************************************************\n");

    int qtd, *p;

    printf("Informe a quantidadee de elementos para o vetor:\n");
    scanf("%d", &qtd);

    int bytes = qtd * sizeof(int);

    p = (int*)malloc(bytes); // 3 x 4 = 12 bytes -> isso eh colocado no ponteiro e ele aloca todo o espaço necessario.
    // ponteiro ira receber um ponteiro do tipo int e o malloc(quantidade * tamanho em bytes de inteiro.)

    for(int i = 0; i < qtd; i++){
        printf("Informe o valor para a posicao %d do vetor:\n",i);
        scanf("%d", &p[i]);
        printf("\n");
    }
    for( int i = 0 ; i < qtd; i++){
        printf(" *p[%d] = %d\n *Bytes_em_Memoria = %ld\n\n", i, p[i], sizeof(p[i]));
    }

    printf("*Memoria Total p[%d] = %ld", qtd, bytes);

    printf("\n*********************************************************\n");

    printf("\n");
    printf("\n");
    return 0;
}//fim_main
```

malloc retorna um ponteiro para um int



- Outra maneira

```c
int qtd, *p;

printf("Informe a quantidade de elementos para o vetor: ");
scanf("%d", &qtd);

p = (int*)malloc(qtd * sizeof(int)); //3 x 4 == 12 bytes

for(int i = 0; i < qtd; i++){
    printf("Informe o valor para a posiçã  %d do vetor: ", i);
    scanf("%d", &p[i]);
}

for(int i = 0; i < qtd; i++){
    printf("No vetor 'numeros[%d]' está o valor %d: \n", i, p[i]);
}
printf("A variável 'p' ocupa %ld bytes em memória.\n", qtd * sizeof(int));

return 0;
```

- A partir do momento que fazemos a alocação de memoria, essa memoria ficará alocada independente se seu progama foi encerrado ou nao. Por isso, sempre temos que liberar a memoria.

free(p); // liberar a memoria (desalocar)

- O parametro de free é o proprio elemento que foi alocado.
- E por ultimo, temos que colocar nosso ponteiro recebendo NULL, uma medida de segurança para o ponteiro não ser reutilizado. Caso não seja anulado, um virus ou outro progama pode utilizar esse ponteiro e ele possui permissões dentro do seu codigo. Ao anulalo, desacoplamos o ponteiro ao endereço de memoria que ele foi alocado.