01. Implemente a função `char *asciiCreate(int n, char init)` que devolve uma *string*, com `n` caracteres, criada dinamicamente. A *string* é iniciada com os caracteres ASCII a partir do parâmetro `init`. Exemplos:

```
asciiCreate(3, 'A') →    "ABC"
asciiCreate(5, 'p') →    "pqrst"
asciiCreate(0, 'A') →    ""
```

02. Implemente a função `char *repete(char *string, int n)` que cria dinamicamente uma nova *string* com `n` cópias da *string* original, separadas por espaço, exceto a última ocorrência. Exemplos:

```
repete("Olá", 4)     →    "Olá Olá Olá Olá"
repete("Oi!", 5)     →    "Oi! Oi! Oi! Oi! Oi!"
repete("Fulano", 0)  →    ""
```

03. Implemente a função `char *metade(char *string)` que cria dinamicamente uma nova *string* contendo apenas a metade da *string* original. Exemplos:

```
metade("Fulano")     →    "Ful"
metade("Ciclano")    →    "Cic"
```

04. Implemente a função `char *inverte(char *string)` que cria dinamicamente uma nova *string*, contendo a *string* original invertida. Exemplos:

```
inverte("Fulano")    →    "onaluF"
inverte("Ciclano")   →    "onalciC"
```

05. Implemente a função `int *ordenar(int *vetor, int length)` que cria dinamicamente um novo vetor, com os elementos do vetor original ordenados em ordem crescente.

06. Implemente a função `int *pares(int *vetor, int lenght)` que cria dinamicamente um novo vetor, contendo apenas os elementos pares do vetor original. O último elemento do vetor retornado pelo método deverá ser o elemento `-1`, de modo a sinalizar o fim do vetor de retorno.

07. Desenvolver a função `int *primos(int quantidade)` na linguagem C, que retorne um vetor de inteiros contendo os primeiros números primos necessários para a quantidade desejada.

```
primos(5) → 1 2 3 5 7
```

08. Implemente a função `char *extract(char *string)` que cria dinamicamente uma nova *string*, contendo apenas as consoantes da *string* original. Exemplos:

```
extract("Fulano")    →    "Fln"
extract("Ciclano")   →    "Ccln"
```