Pontifícia Universidade Católica do Rio de Janeiro
Departamento de Informática
Disciplina: INF1301 - Programação Modular
Professor: Alessandro Garcia / Eiji Adachi Barbosa
Nome do Aluno / Matrícula:

Exercício de Testes Caixa Preta

–Função:

```
char* substring( char *str, int begin )
```

– Descrição:

Função recebe uma string que contém apenas caracteres alfanuméricos e um inteiro (*begin*) e retorna uma substring contendo os caracteres da string de entrada contidos entre *begin* (inclusive) e o último caractere.

–Exemplos:

substring(“hamburguer”, 4) == “urguer”

substring(“programação modular”, 1) == “rogramação modular”

substring(“vazio”, 5) == “”

Neste exercício, você deve:

- Definir as classes de equivalência (válidas e inválidas) para cada uma das entradas
- Fazer a análise de limite para a variável *begin*
  - Para definir os limites use as seguintes abreviações:
    - Limite Superior: LS
    - Acima do Limite Superior: AcLS
    - Abaixo do Limite Superior: AbLS
    - Limite Inferior: LI
    - Acima do Limite Inferior: AcLI
    - Abaixo do Limite Inferior: AbLI
- Descreva um conjunto de casos de testes que exercite todas as classes de equivalência definidas, bem como todos os limites das variáveis *begin*. Use a tabela na página seguinte.

(Possível Resposta)

A partir do enunciado extraímos duas assertivas de entrada:

1. A string de entrada *str* possui apenas caracteres alfanuméricos (letras e dígitos)
2. O parâmetro de entrada *begin* está restrito a valores entre 0 e strlen(*str*) (tamanho da string)

A partir das assertivas de entrada derivamos as classes de equivalência:

- Classe 1.a – Todas as strings que possuem apenas caracteres alfanuméricos (válida)
- Classe 1.b – Todas as string que possuem pelo menos um caractere não-alfanumérico (inválida)

Perceba que a segunda assertiva de entrada é uma relação entre *begin* e o tamanho de *str*. Neste caso, o domínio desta relação são pares (char*, int). As classes de equivalência serão definidas em termos destes pares:

- Classe 2.a – Todos os pares (s, x) em que $0 \leq x \leq$ strlen(s) (válida)
- Classe 2.b – Todos os pares (s, x) em que $x < 0$ (inválida)
- Classe 2.c – Todos o pares (s, x) em que $x >$ strlen(s) (inválida)

Agora refinamos as classes de equivalência derivadas da segunda assertiva de entrada usando o critério de Análise de Valores Limites:

- Classe 2.a – Limite inferior (LI) = 0
- Classe 2.a – Acima do limite inferior (AcLI) = 1
- Classe 2.b – Abaixo do limite inferior (AbLI) = -1
- Classe 2.a – Limite superior (LS) = strlen(str)
- Classe 2.a – Abaixo do limite superior (AbLS) = strlen(str) – 1
- Classe 2.c – Acima do limite superior (AcLS) = strlen(str) + 1

Agora, criamos um conjunto de casos testes que cubra todas as classes de equivalência e todos os limites encontrados. Além disso, é recomendável criar um caso de teste que simule um caso típico de uso da função.

ID	Entrada	Classe Válida e Limite Coberto	Classe Inválida e Limite Coberto
1	`("abcde", 0)`	Classe 1.a Classe 2.a(LI)	
2	`("abcde", 1)`	Classe 1.a Classe 2.a(AcLI)	
3	`("abcde", -1)`	Classe 1.a	Classe 2.b(AbLI)
4	`("a-(x*x)", 6)`	Classe 2.a(LS)	Classe 1.b
5	`("a-(x*x)", 5)`	Classe 2.a(AbLS)	Classe 1.b
6	`("a-(x*x)", 7)`		Classe 1.b Classe 2.a(AcLS)
7	`("abcde", 2)`	Classe 1.a Classe 2.a (Caso típico)	