Trabalho Prático 10

Overflow

Algoritmos e Estruturas de Dados I

Prazo: **28** de **outubro** de 2017

1 Introdução

Pedro resolveu utilizar de um software de código aberto para controlar as finanças de sua papelaria. Por falta de experiência, acabou utilizando de um software muito antigo que a muitos anos não recebia atualizações. Nos primeiros dias, tudo correu bem, mas depois de algumas semanas, resultados muito estranhos começaram a aparecer, como, por exemplo, saldos negativos e trocos impossíveis.

Após uma rápida consulta à ferramenta Poodle, ele descobriu que esses erros eram por causa de um problema chamado *overflow*. Esse problema decorre do número máximo que um computador consegue representar dentro de sua memória. Caso alguma conta executada pelo computador dê um resultado acima desse número, ocorrerá o *overflow*, que é quando o computador faz uma conta e o resultado não pode ser representado, por ser maior do que o valor máximo permitido (em inglês *overflow* significa transbordar). Por exemplo, se um computador só pode representar números menores ou iguais a 1023 e mandamos ele executar a conta $1022 + 5$, vai ocorrer *overflow*.

Esse número máximo (M) está relacionado com o número de *bits* (N_b) que a memória do computador utiliza, dado pela Equação 1.

$$M = 2^{N_b} - 1 \tag{1}$$

2 Tarefa

Pedro pediu que crie um programa que, dado o número de *bits* utilizado pela memória e uma expressão de soma ou multiplicação entre dois inteiros, determine se ocorrerá ou não *overflow*. **Você deverá implementar a expressão 2^n como uma função com chamada recursiva. Não será permitido utilizar funções ou operadores já prontos.** Mais especificamente, quando for calcular o número máximo M que pode ser representado, você deverá chamar essa função.

3 Entrada

A primeira linha da entrada contém dois inteiros N e B representando, respectivamente, a quantidade de expressões que deverá testar e o número de *bits* utilizados pela memória. As próximas N linhas são formadas de um inteiro P, seguido de um espaço em branco, seguido de um caractere C (que pode ser '`+`' ou '`*`', representando os operadores de adição e multiplicação, respectivamente), seguido de um espaço em branco, seguido de um outro inteiro Q. Essa linha representa a expressão $P + Q$, se o caractere C for '`+`', ou $P \times Q$, se o caractere C for '`*`'. Tanto P quanto Q são sempre maiores ou iguais a 0.

4 Saída

Para cada umas das N expressões, seu programa deve imprimir uma única linha, contendo a palavra "`OVERFLOW`" se o resultado da expressão causar um *overflow*, ou a palavra "`OK`" caso contrário. Ambas as palavras devem ser escritas com letras maiúsculas, desconsiderando as aspas.

5 Exemplo

Entrada

```
8 10
100 * 100
10 + 5
1020 + 10
1023 + 1
50 * 0
1022 + 1
25 * 700
0 * 0
```

Saída

```
OVERFLOW
OK
OVERFLOW
OVERFLOW
OK
OK
OVERFLOW
OK
```

Outros exemplos estão disponíveis na página deste trabalho no Moodle.

6 Avisos

Avisos mandatórios para o envio do trabalho:

- O código deve ser escrito em linguagem `C`
- As entradas que serão utilizadas para teste não conterão erros, então não será necessário testar a validade das mesmas
- Não utilize chamadas da função `system` (por exemplo, `system("pause")`) pois essas podem variar de acordo com o sistema operacional e os programas instalados da máquina onde o programa está rodando
- Deixe seu código bem comentado para facilitar a correção
- Não utilize espaços ou caracteres especiais nos nomes dos arquivos. Utilize apenas os caracteres de `A` a `Z` (tanto maiúsculas como minúsculas) sem acento, os números de `0` a `9` e os caracteres `-` (hífen), `_` (*underscore*) e `.` (ponto final)
- Utilize a extensão `.c` para arquivos de código e `.h` para arquivos de cabeçalho, quando aplicar
- Se for submeter os arquivos via upload, não envie um arquivo comprimido (por exemplo, `.zip`, `.rar`, etc.). Utilize os diversos campos da aba "Submissão", um para cada arquivo
- Envie apenas os arquivos `.c` e `.h`
- Não copie o trabalho de algum colega ou baixe da internet. Lembre-se que o prejudicado será você pois o aprendizado obtido nessa disciplina será utilizado durante diversas outras etapas do seu curso

7 Dicas

Dicas importantes para o desenvolvimento deste trabalho:

- Utilize a função `scanf` para ler as entradas do usuário
- Utilize a função `printf` para imprimir os resultados das operações
- Lembre-se que apenas a primeira linha da entrada não produz saída
- Para criar a função recursiva, pense que $2^n = 2 \times 2^{n-1}$. **Cuidado com a condição de parada!**
- Utilize arranjos de caracteres (*strings*) para ler o operando, pois o `scanf` com `%c` também lê espaços em branco!
- Você deve utilizar *strings* de tamanho no mínimo 2 para conseguir guardar 1 caractere (devido ao caractere `\0` ao final)

- Note que `scanf("%s", operador)` lê um arranjo de caracteres (*string*) para a variável `operador`
- Você pode acessar um caractere de uma *string* através dos colchetes, assim como é feito em arranjos. Por exemplo, na *string* `"abcdef"`, o índice 0 é a letra `a`, o índice 1 é a letra `b` e assim sucessivamente
- Qualquer dúvida que tiver, utilize o fórum de dúvidas no Moodle. Inicie o assunto do tópico com a tag `[TP10]`
- Caso prefira, participe das monitorias toda quarta das 17h às 18h na sala 2012

8 Checklist

Checklist não exaustiva de passos até a entrega do trabalho:

1. Pesquise o funcionamento das funções citadas acima para facilitar o uso
2. Implemente e compile o programa
3. Teste para o exemplo dado acima. Compare as saídas para garantir o funcionamento correto
4. Faça o mesmo do item anterior para os exemplos disponibilizados na página deste trabalho no Moodle
5. Envie o trabalho pelo Moodle, onde ele será testado automaticamente para todos os casos disponíveis
6. Caso algum teste dê errado, volte ao passo 2

Bom trabalho e divirta-se!