SIMULADOR DE CIRCUITOS DIGITAIS

PROFESSOR: ADELARDO ADELINO DANTAS DE MEDEIROS

O objetivo é desenvolver em C++ um programa simulador de circuitos lógicos, composto por portas lógicas de 2 a 4 entradas (ou de uma entrada, no caso da NOT) dos seguintes tipos:

- NOT
- AND, NAND
- OR, NOR
- XOR, NXOR

O aplicativo deve ser programado baseando-se em objetos polimórficos para modelagem das portas lógicas. Ou seja, não deve haver instruções de controle de fluxo (`if`, `switch`, ternários, etc.) que mudem a forma de execução de acordo com o tipo da porta (OR, NOT, NAND, etc.). A exceção possível é no tratamento imediatamente seguinte à leitura (do arquivo ou da interface) do tipo de porta a ser incluído no circuito, caso em que podem ser previstas instruções de controle de fluxo para criação do objeto adequado. Também pode haver instruções de controle de fluxo para lidar com outros tipos de informação que não o tipo da porta.

Os dados de entrada, fornecidos pelo usuário, via interface ou através da leitura de um arquivo, são:

- Número de entradas e saídas do circuito.
- Número de portas lógicas do circuito.
- Para cada uma das portas lógicas:
  - O tipo de porta (AND, NOT, etc.).
  - O número de entradas da porta (exceto para portas NOT).
  - Para cada entrada da porta:
    - A origem do sinal lógico: uma porta ou uma das entradas do circuito.
- Para cada uma das saídas do circuito:
  - A origem do sinal lógico: uma porta ou uma das entradas do circuito.

O programa deve ter ao menos as seguintes funcionalidades:

1. Definição de um novo circuito a partir de dados fornecidos via interface.
2. Salvamento em arquivo de um circuito.
3. Leitura de um circuito a partir de um arquivo previamente salvo.
4. Para um circuito definido (via interface ou leitura de arquivo), gerar a sua tabela verdade (saídas para todas as possíveis entradas).

As entradas e saídas do circuito e das portas devem prever a possibilidade de sinais digitais verdadeiros (TRUE), falsos (FALSE) ou indefinidos (UNDEF). A simulação deve ser capaz de lidar com circuitos contendo ciclos, calculando as saídas quando for possível fazê-lo ou informando que uma ou mais saídas ficam UNDEF para aquelas entradas, apenas quando não for possível a sua determinação (TRUE ou FALSE).

ARQUIVO

Os arquivos de leitura e escrita dos circuitos devem seguir rigorosamente um padrão, de tal forma que possam ser reconhecidos pelos programas desenvolvidos por todos os alunos. Há exemplos do arquivo disponível na turma virtual no SIGAA (tanto arquivos corretos quanto arquivos contendo erros). O formato dos arquivos é o seguinte:

CIRCUITO: *Nin Nout Nportas*
PORTAS:
*id***)** *type num_in***:** *id_in1 id_in2*
...
*id***)** *type num_in***:** *id_in1 id_in2*
SAIDAS:
*num***)** *id_out*
...
*num***)** *id_out*

Os trechos escritos em **COURIER NEW negrito** devem estar presentes no arquivo exatamente como aparecem acima. Os trechos em *ARIAL itálico* correspondem aos locais onde serão salvos no arquivo os valores numéricos correspondentes ao circuito específico. O significado dos valores no arquivo é o seguinte:

- *Nin*: número de entradas do circuito
- *Nout*: número de saídas do circuito
- *Nportas*; número de portas do circuito
- *id*: identificador da porta ($1 \leq id \leq Nportas$)

- *type*: tipo da porta:
  - NT = porta NOT;
  - AN = porta AND;
  - NA = porta NAND;
  - OR = porta OR;
  - NO = porta NOR;
  - XO = porta XOR;
  - NX = porta NXOR
- *num_in*: número de entradas da porta lógica (1 para NOT; 2 a 4 para as demais)
- *id_in*: identificador da origem do sinal lógico de cada uma das entradas da porta (compatível com o número anterior).
  - > 0 se o sinal vem da saída de uma porta ($1 \leq$ id_in $\leq$ Nportas)
  - < 0 se o sinal vem de uma entrada do circuito ($-1 \geq$ id_in $\geq$ -Nin)
- *num*: número da saída ($1 \leq num \leq Nout$)
- *id_out*: identificador da origem do sinal lógico da saída do circuito:
  - > 0 se o sinal vem da saída de uma porta ($1 \leq$ id_in $\leq$ Nportas)
  - < 0 se o sinal vem de uma entrada do circuito ($-1 \geq$ id_in $\geq$ -Nin)

As portas e saídas devem estar ordenadas no arquivo, de modo que as linhas correspondentes à primeira porta e à primeira saída no arquivo devem ter *id* e *num* iguais a 1; as últimas devem ter *id* e *num* iguais a *Nportas* e *Nout*, respectivamente. As linhas intermediárias devem estar ordenadas.

Todo arquivo de descrição de circuito, durante a leitura, deve ser conferido para verificação de que está correto (não há entradas de portas ou saídas desconectadas, referência a uma *id* de porta inexistente, formato incorreto, etc.). Caso seja encontrada qualquer incoerência, o arquivo deverá ser desconsiderado.

ALGORITMOS

Os algoritmos em pseudocódigo listados a seguir podem ser úteis no desenvolvimento do aplicativo.

SIMULAR CIRCUITO:

```
Para i de 0 a Num_portas-1
| portas[i].saida <- UNDEF;
Fim Para

Repita
| tudo_def <- true;
| alguma_def <- false;
| Para i de 0 a Num_portas-1
| | Se (portas[i].saida == UNDEF)
| | | in_porta <- entradas
| | |          booleanas da porta i
| | | portas[i].simular(in_porta)
| | | Se (portas[i].saida == UNDEF)
| | | | tudo_def <- false
| | | Caso contrário
| | | | alguma_def <- true
| | | Fim Se
| | Fim Se
| Fim Para
Enquanto (!tudo_def && alguma_def)
```

GERAR TABELA VERDADE:

```
Para i de 0 a Num_inputs-1
| inputs[i] <- FALSE;
Fim Para

Repita
| simular_circuito(inputs);
| // Qual input incrementar?
| i <- Num_inputs-1
| Enquanto (i >= 0 &&
|           inputs[i] == UNDEF)
| | inputs[i] <- FALSE
| | i--
| Fim Enquanto
| // Incrementa a input escolhida
| Se (i >= 0)
| | Se (inputs[i] == FALSE)
| | | inputs[i] <- TRUE
| | Caso contrário // É TRUE
| | | inputs[i] <- UNDEF
| | Fim Se
| Fim Se
Enquanto (i >= 0)
```