UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO

SIMULADOR DE CIRCUITOS DIGITAIS

PROFESSOR: ADELARDO ADELINO DANTAS DE MEDEIROS

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

- NOT (NEGAÇÃO)
- AND (E), NAND (NOT AND)
- OR (OU), NOR (NOT OR)
- XOR (OU EXCLUSIVO), NXOR (NOT XOR)

As entradas e saídas do circuito e das portas devem lidar com sinais lógicos verdadeiros (T - TRUE), falsos (F - FALSE) ou indefinidos (? - UNDEF), realizando as operações lógicas básicas (AND, OR e NOT) das seguintes maneiras:

A	B	A AND B
?	?	?
?	F	F
?	T	?
F	?	F
F	F	F
F	T	F
T	?	?
T	F	F
T	T	T

A	B	A OR B
?	?	?
?	F	?
?	T	T
F	?	?
F	F	F
F	T	T
T	?	T
T	F	T
T	T	T

A	B	A XOR B
?	?	?
?	F	?
?	T	?
F	?	?
F	F	F
F	T	T
T	?	?
T	F	T
T	T	F

A	NOT A
?	?
F	T
T	F

A simulação deve ser capaz de lidar com circuitos contendo ciclos, calculando as saídas ou informando que uma ou mais saídas ficam UNDEF quando não for possível a sua determinação (TRUE ou FALSE).

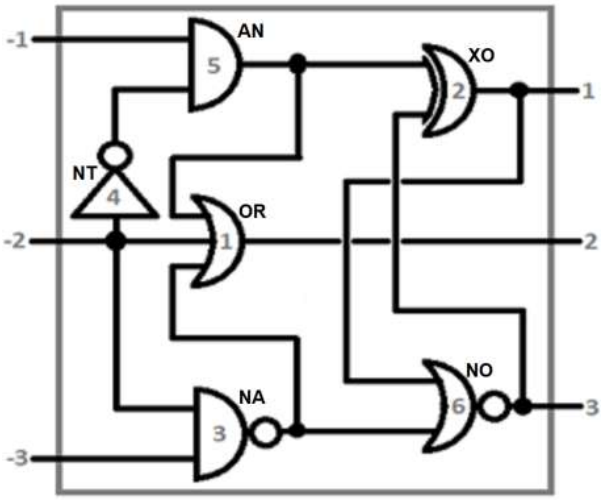
Os dados de entrada a serem fornecidos pelo usuário, via interface ou arquivo, são:

- Número de entradas e saídas do circuito.
- Número de portas lógicas do circuito.
- Para cada porta lógica:
  - O tipo de porta (AND, NOT, etc.).
  - O nº de entradas da porta (exceto NOT).
- Para cada entrada de cada uma das portas lógicas:
  - A origem do sinal lógico: uma porta ou uma das entradas do circuito.
- Para cada saída do circuito:
  - A origem do sinal lógico: uma porta ou uma das entradas do circuito.

Tendo em vista que um dos objetivos principais do projeto é praticar a utilização do polimorfismo baseado em métodos virtuais, além de utilizar regras de boa programação, algumas regras devem ser **obrigatoriamente** seguidas:

- O aplicativo deve utilizar objetos polimórficos e métodos virtuais para modelagem e implementação das portas lógicas. **NÃO** deve, por exemplo, utilizar vários `if`'s ou um `switch`, para obter um comportamento variável de acordo com o tipo de porta, quando poderia utilizar um método polimórfico baseado um funções virtuais.
- As classes que representam as portas lógicas e o simulador de circuitos devem se basear e utilizar o tipo `bool3S` fornecido, sem modificá-lo ou ignorá-lo.
- O programa deve se basear na implementação parcial fornecida das classes das portas e da classe `Circuito`, sem modificar as partes fornecidas, embora possam ser feitos acréscimos. Além das funcionalidades já concluídas, devem ser desenvolvidos ou completados **todos** os métodos declarados na implementação parcial, incluindo:
  - Funções necessárias nas diversas classes que representam as portas.
  - Construtores, destrutores e sobrecarga de operadores na classe `Circuito`.
  - Funções com implementação incompleta na classe `Circuito`.
  - Geração das saídas para uma dada combinação das entradas (`simular`) na classe `Circuito`.

ARQUIVO




```
CIRCUITO 3 3 6
PORTAS
1) OR 3
2) XO 2
3) NA 2
4) NT 1
5) AN 2
6) NO 2
CONEXOES
1) 5 -2 3
2) 5 6
3) -2 -3
4) -2
5) -1 4
6) 2 3
SAIDAS
1) 2
2) 1
3) 6
```

Os arquivos de leitura e escrita dos circuitos devem seguir um padrão. O formato que deve ser seguido **ao salvar**[1] um arquivo é o seguinte:

CIRCUITO *Nin Nout Nportas*
PORTAS
id_port) *type n_in*
...
id_port) *type n_in*
CONEXOES
id_port) $id\_orig\_in_1$ … $id\_orig\_in_{n\_in}$
...
id_port) $id\_orig\_in_1$ … $id\_orig\_in_{n\_in}$
SAIDAS
id_out) *id_orig_out*
...
id_out) *id_orig_out*

Os trechos em **negrito** devem estar presentes no arquivo salvo. Os trechos em *itálico* correspondem aos locais onde serão salvos no arquivo os valores correspondentes ao circuito. O significado dos valores é o seguinte:

- *Nin*: número de entradas do circuito
- *Nout*: número de saídas do circuito
- *Nportas*; número de portas do circuito
- *id_port*: identificador da porta
  ($1 \leq id\_port \leq Nportas$)
- *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
- *n_in*: número de entradas da porta lógica (1 para NOT; 2 ou mais para as outras).
- $id\_orig\_in_i$: identificador da origem do sinal lógico da i-ésima entrada da porta (compatível com o número de entradas *n_in*).
  - > 0 se o sinal vem da saída de uma porta ($1 \leq id\_orig\_in \leq Nportas$)
  - < 0 se o sinal vem de uma entrada do circuito ($-1 \geq id\_orig\_in \geq -Nin$)
- *id_out*: identificador da saída
  ($1 \leq id\_out \leq Nout$)
- *id_orig_out*: identificador da origem do sinal lógico da saída do circuito:
  - > 0 se o sinal vem da saída de uma porta ($1 \leq id\_orig\_out \leq Nportas$)
  - < 0 se o sinal vem de uma entrada do circuito ($-1 \geq id\_orig\_out \geq -Nin$)

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_port* e *id_out* iguais a 1; as últimas devem ter *id_port* e *id_out* iguais a *Nportas* e *Nout*, respectivamente.

[1] Em leitura, admitem-se arquivos que não sigam exatamente esse formato, desde que as informações necessárias estejam presentes, válidas e na ordem correta.

ALGORITMOS

SIMULAR CIRCUITO:

```
// NOTAÇÃO
in_circ_id: valor lógico (bool3S) da entrada
            id do Circuito.
out_port_id: valor lógico (bool3S) da saída da
            porta id do Circuito.
out_circ_id: valor lógico (bool3S) da saída id
            do Circuito.
id_in_id,j:  identificador da origem do sinal da
            j-ésima entrada da porta id do
            Circuito.
id_out_id:   identificador da origem do sinal da
            saída id do Circuito.
tudo_def, alguma_def,
id_orig, in_port[]: variáveis locais

// INICIALIZAÇÃO
Para todas as "id" das portas:
| out_port_id ← UNDEF
Fim Para

// ALGORITMO ITERATIVO
Repita
| tudo_def ← TRUE
| alguma_def ← FALSE
|
| Para todas as "id" das portas:
| | Se (out_port_id == UNDEF)
| | | Para todas as "j" entradas
| | | da porta "id":
| | | | // De onde vem a entrada?
| | | | id_orig ← id_in_id,j
| | | | // Valor bool3S da entrada
| | | | in_port[j] ← { out_port_id_orig , se id_orig>0
| | | |              { in_circ_id_orig  , se id_orig<0
| | | Fim Para
| | |
| | | // Simula a porta "id" com
| | | // entradas in_port
| | | simular_id(in_port)
| | |
| | | // Calcula os critérios
| | | // de parada do algoritmo
| | | Se (out_port_id == UNDEF)
| | | | tudo_def ← FALSE
| | | Caso contrário
| | | | alguma_def ← TRUE
| | | Fim Se
| | Fim Se
| Fim Para
Enquanto ( NÃO(tudo_def) E
           alguma_def )

// DETERMINAÇÃO DAS SAÍDAS
Para todas as "id" das saídas:
| // De onde vem a saída?
| id_orig ← id_out_id
| // Valor bool3S da saída
| out_circ_id ← { out_port_id_orig , se id_orig>0
|               { in_circ_id_orig  , se id_orig<0
Fim Para
```

SIMULAR PORTA:

Os operadores são associativos:

A AND B AND C = (A AND B) AND C
= A AND (B AND C)

A OR B OR C = (A OR B) OR C
= A OR (B OR C)

A XOR B XOR C = (A XOR B) XOR C
= A XOR (B XOR C)

Portanto, para simular uma porta com 2 ou mais entradas, basta inicializar o resultado com o valor da primeira entrada e, para todas as demais entradas, realizar a operação lógica entre o valor da nova entrada e o resultado anterior, que passa a ser o novo resultado.

GERAR TABELA VERDADE:

```
// INICIALIZAÇÃO
Para todas as "i" entradas do
circuito:
| in_circ_i ← UNDEF
Fim Para

// GERAÇÃO DA TABELA
Repita
| simular_circuito(in_circ)
| // Qual input incrementar?
| i ← Índice da última entrada
| Enquanto (i for índice válido E
|           in_circ_i == TRUE)
| | in_circ_i++ // TRUE → UNDEF
| | i--
| Fim Enquanto
| // Incrementa a input escolhida
| Se (i for índice válido)
| | in_circ_i++ // UNDEF → FALSE
| |             // FALSE → TRUE
| Fim Se
Enquanto (i for índice válido)
```

SUGESTÃO DE DESENVOLVIMENTO

1) Implemente todas as funcionalidades das portas. Faça um programa de teste (veja sugestão `teste1.cpp` no SIGAA) que:
   a) Utilize construtores com e sem parâmetros, válidos e inválidos, e use funções de consulta (`getNumInputs`, etc.) para testar se as portas foram criadas corretamente.
   b) Teste se a função `simular` só aceita vetor de `bool3S` com a dimensão correta.
   c) Verifique se a simulação está correta para todas as combinações de entrada.
   d) Crie objetos dinâmicos e teste se os métodos virtuais (`getName`, `simular`, etc.) exibem comportamento polimórfico e correto.

2) Implemente as obrigatoriedades da classe `Circuito` (construtores, destrutor, operadores de atribuição) e as funções `clear`, `resize` e `setPort`. Faça um programa de teste (veja sugestão `teste2.cpp`) que:
   a) Teste o construtor default.
   b) Teste as funções `resize` e `setPort` com parâmetros válidos e inválidos.
   c) Teste o construtor por cópia, verificando se o novo objeto e o antigo têm memórias dinâmicas independentes (alterar um deles não modifica o outro) e são idênticos.
   d) Teste o construtor por movimento, verificando se o novo objeto tem memória válida (pode ser alterada).
   e) Teste os operadores de atribuição por cópia e por movimento.

3) Implemente a função `simular` da classe `Circuito`. Teste com o programa principal da avaliação.
   a) Confira os resultados da simulação, utilizando, por exemplo, a avaliação simulada fornecida no SIGAA.