

# Logic Gates

## C++ zápočtový

Tomáš Karella

2. května 2017

### Téma:

Tento program slouží k tvorbě hradlových sítí, následně k simulaci jejich výpočtu a jejich opakovanému využití v dalších hradlových sítích. Propojení sítě je koncipováno přes načítání konstrukčního souboru, který je popsát dále.

### Kompilace a spouštění:

Pro spuštění na Linux distribucích je nutný překladač g++-6 a GNU Make. Na github branch: linuxB

- make compile - zkompiluje zdrojáky do spustitelného souboru "/bin/main"
- make clean - smaže zdrojové zkompileované soubory
- make convert - změní eol všech příkladů z CRLF na LF
- make test1 - make test6 - zkonstruuje příklad a vyzkouší všechny možné vstupy
  1. XORAND ADDW
  2. XORAND ADDW ADD4
  3. allgate
  4. DAND
  5. DXOR
  6. example.txt

Pro spuštění na Windows distribucích je program dostupný pouze jako Visual Studio projekt (VS 2015 a vyšší). Na github branch: master

# Uživatelská dokumentace:

## Interaktivní režim:

Po spuštění bez parametrů se otevře interaktivní režim, který vás vyzve k zadání cesty konstrukčního souboru. Po jeho úspěšném zkonstruování se přepne do režimu vkládání vstupu, kdy pro daný vstup spočítá a vrátí výstup. Dále poskytuje možnost zkonstruovat hradlo pro další použití (klíčové slovo: c). Po úspěšné konstrukci se opět dostane do režimu načítání souborů. Nyní už může používat jméno prvního konstruovaného hradla jako typ.

## Formát vstupu:

Vstupy pro hradlovou síť jsou ve formátu řetězce znaků a to 1 pro logickou 1 a 0 pro logickou 0.

## Interaktivní režim - klíčová slova:

**exit, e** - slouží k ukončení aplikace

**const, c** - konstruuje zadané hradlo, (jen v režimu, kdy je načtený konstrukční soubor)

**h, help, man** - zobrazí klíčová slova

## Pasivní režim:

Pro spuštění pouze konstrukce a simulace jednotlivých hradel. Lze používat následující argumenty při spuštění main fce. Pro spuštění v pasivní režimu musí být nastaven alespoň jeden konstrukční soubor a alespoň jeden vstup.

**-f [file..]** - cesty ke konstrukčním souborům, které se konstrují dle pořadí

**-h** - vytiskne argumenty fce

**-i** - vstupy pro poslední generované hradlo, ve formát 1 - pro logickou 1 a 0 - pro logickou nulu, př: pro hradlo DAND: 10

**-a** - vyzkouší všechny možné vstupy pro poslední konstruované hradlo a -i ignoruje

**-d** - zobrazí debug informace

## Konstrukční soubor - formát:

Modelový soubor lze nalézt "examples/model.txt". Ve zmíněné složce je i celá řada příkladů k vyzkoušení programu.

Soubor se skládá ze dvou hlavních částí. Pojmenování hradel, kde deklarujete jméno hradla (noCASE sensitive a smí obsahovat pouze číslice a písmena) k jménu typ hradla. Část druhá, kde se řeší jejich vzájemné propojení. Jednotlivé tagy jsou odděleny tabulátorem.

```

#GATE      MYNAME (1)
#INPUT      sizeOfInput
#OUPUT      sizeOfOutput
NameOfGate      Type (2)
NameOfGate      Type
#CONNECT (3)
NameOfGate[OutputPinID]    ->      NameOfGate[InputPinID] (4)
NameOfGate[OutputPinID]    ->      NameOfGate[InputPinID]
# (5)

```

1. kontrolní tag pro pojmenovací část souboru a jméno vašeho hradla (oddělené tabulátorem) na dalších řádcích tagy pro zadání velikosti vstupu & výstupu
2. jméno hradla(pouze písmena a číslice) dále typ(predefinovaný či už zkonstruovaný) vzájemně odděleny tabulátorem.
3. kontrolní tag pro začátek propojovací část souboru
4. jméno hradla a v hranatých závorkách číslo výstupního pinu dále "->"oddělená z obou stran tabulátorem jméno hradla s číslem vstupního pinu. Pro připojení vstupu je přednastaveno jméno I a pro připojení výstupu přednastaveno jméno O.
5. kontrolní tag konce konstr. souboru

Pro konstrukci musí být připojeny u hradel všechny vstupní i výstupní piny, konstrukt musí obsahovat alespoň jedno hradlo vstupní a alespoň jedno výstupní.

## Konstrukční soubor - seznam předdefinovaných typů hradel

- Základní logické fce:
  - NOT
  - AND
  - OR
  - XOR
  - NAND
  - NOR
  - XNOR

- Ostatní
  - Input
  - Output
  - Blank (má pouze vstup a nikam není dále posílán)
  - ConstIn0 (má pouze výstup a stále nastaven na 1)
  - ConstIn1 (má pouze výstup a stále nastaven na 1)
  - Double (na dva výstupy použít stejný 1 vstup)

## Konstrukční soubor - příklady

- model.txt - obecný model konstrukčního souboru
- allgate.txt - hradlo využívající všechno předdefinované hradla
- DAND.txt - double and, and s 2 výstupy
- DXOR.txt - double xor, xor s 2 výstupy
- XORAND.txt - xor a and na stejný vstup, první výstup xor druhý and
- ADDW.txt - sčítanka dvou čísel a přetečení, musí být zkonstruovaný XORAND
- ADD4.txt - sčítanka 2x2 bitových čísel, musí být zkonstruovaný XORAND a ADDW

## Implementace:

Implementaci je rozdělena do následujících částí:

- **graph**(graph.h) - implementuje multigrafu, každý vrchol a hrana nese generickou hodnotu.
- **gates** (gates.h, gates.cpp) - deklaruje obecně třídu hradlo a její konkrétní implementace.
- **workbench** (workbench.h, workbench.cpp) - implementuje propojování jednotlivých hradel v hradlovou síť, kontroluje jejich korektnost a konstruuje uživatelsky definovaná hradla.
- **workbenchTUI** (workbenchTUI.cpp, workbench.TUI.h) - řeší komunikaci mezi uživatelem a workbench, načítá konstrukční soubory.
- **main** (main.cpp) - parsuje vstupní parametry a spouští metody workbenchTUI.

### graph

Obsahuje následující šablonové třídy s typovými parametry VertexValue, EdgeValue:

- Vertex - drží hodnotu VertexValue
- Edge - orientovaná hrana mezi Vertex s hodnotou EdgeValue
- Graph - orientovaný multigraf nad Vertex a Edge (vtype = Vertex<VertexValue>, etype = Edge<VertexValue,EdgeValue> )
  - vtype \* add\_vertex(VertexValue value) => přidává vrchol do grafu bez hran
  - etype\* connect(vtype\* from, vtype\* to, EdgeValue value) => vytváří hranu z from do to s hodnotou value
  - void disconnect(etype\* e) => smaže v grafu hranu e
  - vector<etype\*> edges\_from(vtype\* a) => vrátí seznam hran z vrcholu a
  - vector<etype \*> edges\_to(vtype\* a) => vrátí seznam hran do vrcholu a
  - unordered\_set<vtype \*> vertices\_from(vtype\* a) => vrátí seznam vrcholů dosažitelných z a
  - bool cycle\_detection() => testuje, zda daný graf obsahuje cyklus, implementováno DFS, které pokud se vrátí do už uzavřeného vrcholu nahlásí nalezení cyklu
  - bool all\_vertices\_available\_from(vector<vtype\*> from) => testuje, zda z daných vrcholů je dosažitelný celý graf, pomocí DFS projde graf. Pokud počet uzavřených vrcholů není shodný s velikostí grafu zahlásí false

## gates

- výčtový typ Status - Zero, One, Floating(logická 1,0 a nenastaveno)
- třída Signal - obsahuje proměnné : toID, fromID - pořadí pinů u vstupního a výstupního hradla a Status aktuální nastavení signálu
- abstraktní třída Gate -
  - obsahuje vlastnosti velikost vstupu, výstupu, název, id - unikátní číslo, result - pravdivý, když jsou nastaveny výstupy, resultValues - hodnota výstupů
  - virtuální metoda Update - spočítá výstup z hradla
- Třídy všech předdefinovaných hradel s přetíženou metodou Update, která počítá jejich logické fce.
  - NOT AND OR XOR NAND NOR XNOR Input Output Blank ConstIn0 ConstIn1 Double
- třídu UserDefinedGateModel - třída uchovávající hradlovou síť sestavenou uživatelem, drží ukazatel na její graf, vstupní a výstupní hradla
  - metoda Update - nastaví vstupní hradla, simuluje průchod skrz graf hradlovou síť a vrátí hodnoty z výstupních hradel
  - metoda getGate - vrací objekt třídy UserDefinedGate, která lze použít v dalších hradlových sítích
- třída UserDefinedGate - třída pro využití v dalších hradlových sítích, ukazuje na UserDefinedGateModel
  - metoda Update volá Update na modelu daného hradla

## workbench

- výčtový typ WorkbenchStatus značící stav workbench - UnderConstruction, Constructed, Calculating, Calculated
- třídu Workbench
  - obsahuje Graph<Gate,Signal> - která je vlastní hradlovou sítí, ukazatele na modely uživatelsky definovaný hradel, na vstupní/výstupní/constatní hradla, ukazatele na ještě nepřipojená hradla(funkce pro jejich výpis)
  - ve stavu konstrukce - jsou k dispozici metody pro přidávání nových hradel, propojování.
    - \* bool Connect(const std::size\_t & freeInputPosition,const std::size\_t & freeInputID, const std::size\_t & freeOutputPosition, const std::size\_t & freeOutputID)  
=> připojí na vstupní pin hradla na freeInputPosition(pozice v listu hradel s volným vstupem) na výstupní pin hradla na freeOutputPosition pozici, \*ID - říká číslo pinu
    - \* bool Connect(gvertex from, gvertex to, std::size\_t fromPin, std::size\_t toPin)  
=> spojí vrchol from s vrcholem to na zvolených pinech
    - \* gvertex Add(const std::size\_t & num) => přidá hradlo do sítě, num určuje jaké standartní hradlo přidá podle pořadí ve vektoru StandardGates
    - \* gvertex GetType(string typeName) => přidá hradlo typu podle typeName(název předdefinovaného hradla, či název už nahraného hradla)
    - \* bool GetVertex(string name, gvertex& v) => vrátí pojmenovaný(name) vrchol do v, pokud name není jméno žádného vrcholu vrátí false
    - \* bool AddNamedGate(string gateName, string typeName) => přidá pojmenovaný vrchol typu typeName do grafu, pokud už existuje pojmenování vrátí false
    - \* bool AddUserDefineGate(const std::size\_t & positionInList) => přidá do grafu už vytvořené hradlo dle pozice v listu
    - \* gvertex AddInputGate() a vertex AddOutputGate() => přidá vstupní či výstupní hradlo
  - Pro konstrukci:
    - \* bool ConstructBench() => Otestuje, zda je hradlová síť korektní. Obsahuje alespoň 1 vstupní a 1 výstupní hradlo, zda nevznikl cykl a zda je hradlo celé dosažitelné. K tomu používá metody graph. Pak se přepne workbench do stavu Constructed. Pokud testy neuspějí vrátí false
  - Při úspěšně zkonstruovaném hradle:
    - \* bool SetInput(vector<bool> input) => nastaví vstupní hradla na hodnoty z argumentu, a simuluje průchod sítí v grafu a skončí až budou nastavené

všechny výstupní hradla. Vráti false, pokud je špatný formát vstupních hodnot.

- \* `vector<bool> ReadOutput()` => vrátí vypočítané výstupy, pokud byla předtím volána fce `SetInput(input)`

- Konstrukce hradla:

- \* `bool ConstructUserGate(string name)` => Zkonstruje z aktuální sítě `UserDefinedGateModel`, přidá jej do seznamu všech `UserDefinedGates`, nastaví novému hradlu typ dle `name`, nakonec zavolá `ResetWorkbench(false)`

- Další funkce:

- \* `vector<string> ListAllGates()`, `string GetTestOutput()` => výpisové funkce všech aktuálních typů hradel a výpis z testů při konstrukci hradlové sítě

- \* `void ResetWorkbench(bool deleteUDG)` => smaže aktuální už zkonstruovanou část hradla, pokud `deleteUDG` smaže i uživatelské typy

## workbenchTUI

- Tvoří vrstvu mezi uživatelem a workbench. Funguje v několika verzích. Vrací výstup na streambuf output a čte streambuf input dané při konstrukci objektu.

- `bool ReadFile(string path)` => načte hradlo z filu dle struktury konstrukčního souboru, vypisuje jednotlivé fáze čtení, pokud čtení selže vrátí false

- `void InteraktiveMode()`, `void InteraktiveMode(string path)` => spustí interaktivní mód, který načte daný soubor, pak umožňuje konstrukci hradla, čtení vstupů a přidávání dalších hradel

- `void PassiveMode(string path, string inputSettings)` => zkonstruuje daný konstrukční soubor, vypočítá výstup podle `inputSettings` a vypíše výstup.

## main

Vytváří své `workbenchTUI`, nastaví input na `std::cin` a output `std::cout`, dále parsuje argumenty, dle jejich počtu volá příslušné metody na `workbenchTUI`.