Инструкции за инсталиране и работа с NETDesigner

Съдържание

Системни изисквания

Unix: Qt5, LLVM, Clang, gcc, gcc-g++ и някой от следните терминални емулатори - konsole, gnome-terminal, terminal или xterm.

Windows: Инсталиран MinGW пакет съдържащ gcc и g++. /bin директорията на MinGW трябва да присъства в системната променлива PATH.

Инсталация

Unix:

Последна версия на сорс кода може да се клонира чрез git:

git clone https://github.com/amentis/NETDesigner.git

или изтегли като .zip архив от GitHub хранилището на проекта:

https://github.com/amentis/NETDesigner

Приложението се компилира и инсталира с

qmake

make

```
sudo make install
```

след което да се стартира с командата

NETDesigner

Windows:

До този момент няма разработен инсталатор за NETDesigner.

RAR архив с приложението, компилирано за Windows, съдържащ всички нужни библиотеки за изпълнение, както и минимална част от clang, копирана от проекта ClangOnWin, за момента може да се изтегли от този линк:

https://www.dropbox.com/sh/zx12iugnedjooat/AABRdJXvYI2kiRzoVsCN0vX7a?dl=0

Вътре в архива се намира и малък .bat файл целящ да улесни настройването на PATH променливата за правилно намиране на C/C++ библиотеката в MinGW.

Да се вземе в предвид, че изпълнимите файлове в този архив не винаги са обновени до последна версия. При инсталирани Qt5 и някаква make програма (например nmake, предоставен с visual studio или GNU make предоставен с MinGW) последна версия на сорс кода може да се клонира чрез git:

git clone https://github.com/amentis/NETDesigner.git

или изтегли като .zip архив от GitHub хранилището на проекта:

https://github.com/amentis/NETDesigner

Приложението се компилира и инсталира с

qmake

nmake (или make или каквато инсталирана алтернатива присъства на машината)

Деклариране на примитиви

Примитивите са подпрограми в езика Net, които се управляват от хода на мрежите, описани на Net.

Всяка база от примитиви трябва да има своя отделна папка с името на базата от примитиви. Вътре в тази директория трябва да присъстват два файла - <име на базата>.info и <име на базата>.cpp.

Пример:
Базата се казва AddAB. Представлява папка на име AddAB, съдържаща файловете AddAB.info и AddAB.cpp

.info файл

Представлява текстов файл с информация за примитивите, съдържани в базата. Използва се за зареждане на базата примитиви в редактора. При грешка в този файл базата няма да може да бъде заредена.

Синтаксис:

<име примитив>: <списък с аргументи>

<име примитив>: <списък с аргументи>

Където името на примитива съвпада с името на примитива в .cpp файла. В списъка с аргументи аргументите имат тип и име, разделени с интервал, като дефинициите на аргументите са разделени със запетайка. Допустимите типове са int, real, bool, string.

Пример:

```
readAB: int a, int b

addBtoA: int a, int b

printA: int a
```

съответно описващи примитив readAB с аргументи a от тип int и b от тип int, примитив addBtoA с аргументи a от тип int и b от тип int и примитив printA с аргумент a от тип a. В случай на примитив без аргументи той се декларира с име на примитив, символ ":" и никакви символи до края на реда, например

```
primitive:

//..//
```

.cpp файл

В този сорс файл трябва да присъства namespace с името на базата, съдържащ всички декларирани примитиви. Всеки примитив е функция, връщаща стойност от тип bool. При успешно изпълнение примитивът трябва да връща true, при неуспешно – false. При обратен ход на примитива няма значение каква стойност се върне, но трябва да се върне стойност за да не се получи segmentation fault. На примитивите се дава флаг от булев тип на име FORW. Неговата стойност се наглася преди стартирането на всеки примитив. Ако FORW е true това значи, че стрелката, в която е примитива, с изпълнява напред, а ако FORW е false – връща се назад. При връщането назад е важно да се възстановят всякакви данни, които са дадени при изпълнение на преден ход на примитива.

Пример:

```
#include <iostream>


namespace AddAB {

    bool readAB(int &a, int &b){
```

```
        std::cout << "a: ";
        std::cin >> a;
        std::cout << "b: ";
        std::cin >> b;
        return true;
    }
    bool addBtoA(int &a, int &b){
        if (FORW){
            a += b;
            return true;
        } else {
            a -= b;
            return true;
        }
    }
    bool printA(int &a){
        std::cout << a;
        return true;
    }
}
```

Работа с основния прозорец на NETDesigner

Менюта

File – съдържа действията New Project..., Open Project..., Save Project, Close Project, Exit

Edit – съдържа действията Add Net, Remove Net, Save Current Net, Save All Nets, Browse Primitives Bases, Program Options

Build and Run – съдържа действията Build, Run, Debug, Clean, View Output Browser

Help – Help, About

Действия

- **New Project...** - създаване на нов проект. След създаването проекта бива отворен
- **Open Project...** - отваряне на проект
- **Save Project** – запазване на проект. Запазва проекта и всички мрежи в него.
- **Close Project** – затваряне на проект. Ако проекта не е запазен на потребителя бива представен диалог, питащ дали да запази проекта.
- **Exit** – Изход от програмата. Преди него проекта се затваря. Ако потребителят откаже запазването на проекта при затваряне на проекта изхода от програмата се прекъсва
- **Add Net** – добавя мрежа в проекта. Мрежата става видима в списъка с мрежи.
- **Remove Net** – премахване на мрежа. Мрежата се изтрива от файловата система и не може да бъде възвърната
- **Save Current Net** – запазва мрежата в отворения в момента редактор
- **Save All Nets** – запазва всички мрежи в проекта
- **Browse Primitives Bases** – показва диалога за разглеждане на бази от примитиви
- **Programs Options** – показва диалога за настройки на програмата
- **Build** – компилира програмата. Преди компилиране проекта се почиства
- **Run** – стартира програмата. Преди стартиране програмата се прекомпилира
- **Debug** – програмата се стартира в режим на debug. В прототипа на NETDesigner тази функционалност не е представена
- **Clean** – почиства проекта. Това означава, че всичко, генерирано от предходно компилиране, се изтрива
- **View Output Browser** – показва полето за четене на изхода
- **Help** – показва прост съобщителен прозорец с кратки инструкции за работа с редактора
- **About** – показва прост съобщителен прозорец с кратка информация за продукта и автора

Работа с редактора за мрежи

При отваряне на мрежа в режим на редактиране се появява редактора на мрежи.

За добавяне на нов възел се клика с ляв бутон на празно място в редактора. Появява се диалог за добавяне на възел, от който се избира типа възел и при нужда се добавя израз.

За редактиране на възел се клика с ляв бутон върху възел.

За премахване на възел се клика с десен бутон на възел.

За добавяне на стрелка се позиционира курсова ма мишката върху възела, от който ще тръгва стрелката. Появява се малък бутон с формата на стрелка в горния десен ъгъл на възела. След, като той се кликне с левия бутон на мишката се избира възел, до който да води стрелката. При кликане върху възел се създава стрелка. При кликане на празно място в редактора се прекъсва заявката за добавяне на стрелка. Ако стрелката тръгва от case или proximity node излиза меню, в което потребителят да въведе константа или израз за този възел.

За редакция на стрелка се клика с ляв бутон върху стрелката. Излиза диалогът за редакция на стрелка. Тук потребителят може да смени настройките за изпълнение на стрелката, да настрои викане на подмрежа или да промени списъка с примитиви, изпълнявани от стрелката. Списъка с примитиви, които могат да бъдат изпълнявани от стрелката се определя от списъка с бази от примитиви заредени в проекта.

При дясно копче върху стрелка се изтрива стрелка.