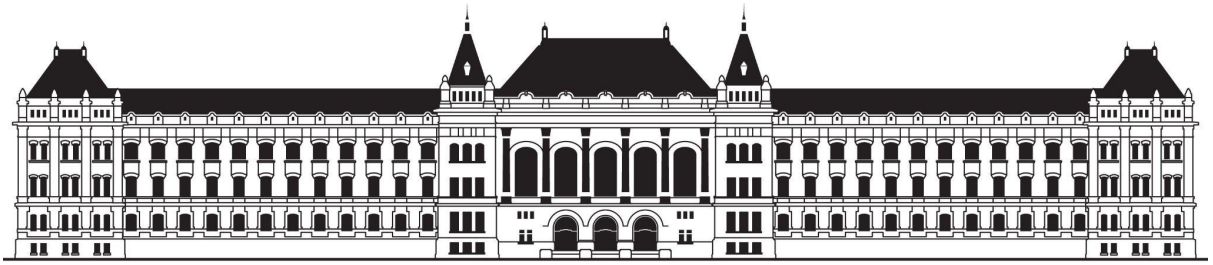


Témalabor

Kódgenerálás és fordítóprogramok

Visual Connect

Kovács Róbert Kristóf

Pintér Bence Zsolt

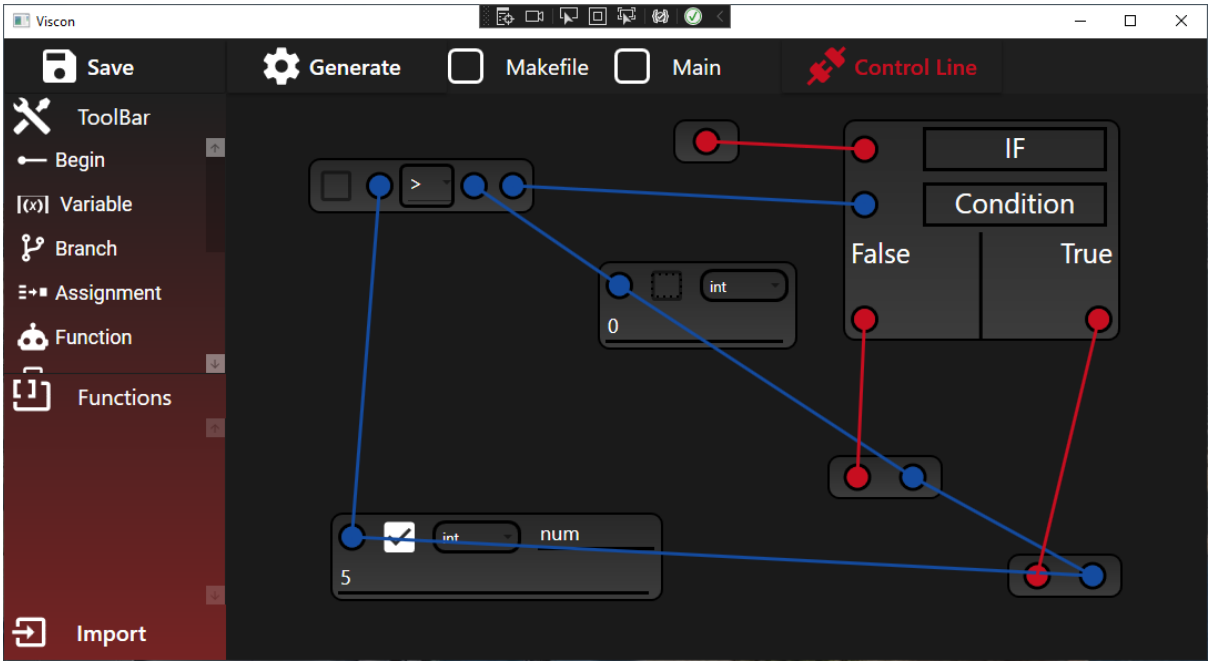
Ábrahám Bálint

1. A feladat leírása

A feladat egy vizuális programozási környezet elkészítése, amely képes c kód és a hozzá tartozó header fájl generálására. A nyelvi elemek és vezérlési szerkezetek a tervrajzon eszközdobozok formájában jelennek meg. Az eszközök között vezérlési éleket, illetve adatkapcsolati éleket húzhatunk. A tervező felületen az eszközök és kapcsolatok szabadon hozzáadhatók, mozgathatók, szerkeszthetők és törölhetők.

Az eszközök között találunk változókat, elágazásokat, ciklusokat. Továbbá lehetőséget biztosít az alkalmazás a változók közötti aritmetikai és logikai műveletek elvégzésére, valamint azok összehasonlítására is. Fontos funkcionalitás továbbá a natív c kódban előre definiált függvények importálása és felhasználhatóvá tétele a grafikus felületen. A munka befejezése után a projekt állapota elmenthető, később vissza is tölthető és folytathatjuk a munkát.

A programot VisCon névre kereszteltük. Ez a Visual Connect szavak rövidített változatainak összevonásából született. A program célja azonban nem csak az, hogy a vezérlési szerkezetek között teremtsen kapcsolatot, hanem az is, hogy akár laikusokat közelebb hozzunk vele a programozáshoz és kutatást végezzünk a kódgenerálás terén.



2. Használt technológiák

- A program elkészítéséhez több programozási nyelv használatát is megfontoltuk, többek között java, c#, c++ is mind felmerült. Végül közös megegyezés alapján a c# .NET Framework mellett döntöttünk könnyen megszokható szintaktika és rengeteg jól működő könyvtár segíti a munkánkat. A c# további előnye, hogy csapat minden tagja mély ismeretekkel és meghatározó mennyiségű tapasztalattal rendelkezett a nyelv használata terén. A megvalósítás során is a számos nyelvi elem megkönnyítette a program fejlesztését és a szoftver megfelelő strukturálását.
- Alkalmazott könyvtárak és technológiák közt kulcsfontosságú szerepet tölt be a C# szerializáló bővítménye, mely a projekt és a szerkesztő állapotainak perzisztens tárolásához szükséges munkát végzi. A projekt fájljai a megszokott módon xml fájlokban tárolódik, melynek formátumát kellő gonddal fejlesztettük ki.
- A grafikus felülethez a WPF (Windows Presentation Foundation) grafikus könyvtárat használtuk. A felület tervezéséhez messzemenőkig személyre szabható és bőséges eszköztárat kínál a nyelv. A felhasználói interfészek fejlesztésében elterjedt jelölőnyelvek családjába tartozó xaml formátumban írt dokumentumokkal készíthetünk statikusan felületleírásokat. Továbbá stílus módosítók, triggerek és eseménytáblák is hozzáadhatók a felhasználói élmény maximalizálása érdekében. A statikusan megírt felület C# kódból könnyeden módosítható és bővíthető.

3. A tervezési és fejlesztési munka menete és annak nehézségei

A fejlesztés folyamán személyes konzultációk mellett Teams értekezleteket is tartottunk. Haladásunkat heti kétszer stand-up meetingeken szinkronizáltuk.

A közös verzió kezeléshez git-et alkalmaztunk, a kódoláshoz és teszteléshez pedig VisualStudio-t.

A tervezési folyamatban ellentétes megközelítést alkalmaztunk, mint a szoftver projekt laboratórium esetében. Mindenek előtt a felhasználói felület homlokzati tervét és stílusát terveztük meg, hogy az mindenképp megfeleljen az előzetes elvárásoknak. Mi több ebben a folyamatban véglegesítettük a programmal szemben támasztott követelmények és funkciók listáját, mely leginkább bővült a program hasznosabbá tétele érdekében.

A tervezés második lépése a teljes logikai réteg megtervezése volt, mely rengeteg átgondolást igényelt. A szerkesztőben prezentált funkcionális elemek üzleti rétegbeli megvalósítása, ezek közti kapcsolat, valamint a programváz többi részének megtervezése volt a legfontosabb, mivel ennek végeztével párhuzamosan lett fejleszthető a grafikus felület és a háttérlogika. Mindössze egy héttel a terv elkészülte után már a kezdetleges implementációk és a rétegek közti kapcsolat is elkészült. A program ekkor már sokkal felügyelhetőbbé és tetsztelhetőbbé vált.

Az előzetes tervezési folyamatból azonban a projekt mentési és visszatöltési funkciója sem maradt ki. Hosszas egyezkedés és merengés után sikerült megegyezni a megfelelő módszerben és szintaktikában. Azonban ennek fejlesztése csak akkor kezdődött el, amikor a program többi részét már sínre tettük.

A további hetekben szorgosan dolgoztunk a program teljessé formálásán, a munka során észrevett tervezési hiányosságok utólagos kiküszöbölésén. Minden funkciót igyekeztünk átfogóan tesztelni annak közvetlen elkészülte után.

Végére hagytuk a legfontosabb és legizgalmasabb funkció megkonstruálását. Magát a C kódgenerátort. Ennek megtervezése, lekódolása és tesztelése nagyszerű feladatnak bizonyult. Végre láttunk, hogy mi mindenre is képes a programunk.

Időközben adódtak azonban nehézségeink. A WPF rengeteg specifikus tulajdonsággal rendelkezik melyeknek nem ismerete miatt sok probléma adódott. Ráadásul sok gyakorlást igényelt, mire mindannyian elsajátítottunk fejlesztésének legfontosabb ismereteit. A grafikus felületek fejlesztésekor általánosan felmerül, hogy a megfelelőnek tűnő, esztétikus dizájn megalkotásra egyenesen rabolja az időt az érdemi funkcionális elemek fejlesztése elől.

A logikai rétegben és a kódgenerátorban a vártnál többször fordult elő, hogy sokáig lappangó hibák csak nagyon későn ütötték fel a fejüket. Mi több számos hiba csak bizonyos időközönként jött elő, ami bonyolította a kiváltó okok feltárását. Ezen a hibák javítása gyakran nem volt egyszerű, mint ahogy az sem, hogy sikerüljön bebizonyítani a javítás sikerességét.

A programunk komplexitása és a kód mérete sajnos a fejleszthetőség és kezelhetőség rovására ment bizonyos mértékben.

4. Elvégzett tesztek és betekintés a program működésébe

A program elkészült komponenseit elsőként statikus tesztelésnek vetettük alá. Majd második lépésben a verifikáció tesztesetek lejátszásával zajlott. Minden egyes elkészült egységet igyekeztünk kipróbálni a lehető legtöbb különböző bemenetre.

A modell esetében az első komolyabb teszt a kapcsolat vizsgálata volt. A kérdés az volt, hogy a modell hitelesen és konzisztensen leköveti-e a felületen történt változásokat. A második komplexebb teszt ennek fordítottja. A modell által küldött utasítások és adatok alapján a felhasználói felület képes-e azt hitelesen reprezentáló munkaablakot felépíteni. A harmadik komolyabb teszt a szerializálás volt. A kérdés ekkor az volt, hogy a modell a visszaolvasott adatok alapján abba az állapotba kerül-e, amelyben az állomány létrehozásának pillanatában volt.

A kódgenerálás esetén azt ellenőriztük, hogy a grafikus felületen felvázolt funkcionalitásnak megfelel-e a generált kód. A generált kódot statikus teszt alá vetettük. Ezután – ahogy a puding próbája az evés – fordítottunk és futtattuk a legenerált kódot. A generált komplex tesztesetekre látható néhány példa alább.

Elsőként tekintsük meg a projektek xml állományait!

A layout fájlok a szerkesztőben elhelyezett funkcionális elemek pozícióját adják meg. Az attribútumként szereplő ID az egyes elemek projekten belüli egyértelmű azonosítóját jelenti.

```
<?xml version="1.0" encoding="utf-8"?>
<Layout xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Nodes>
    <NodePosition ID="4" X="225" Y="75" />
    <NodePosition ID="9" X="618" Y="151" />
    <NodePosition ID="11" X="233" Y="89" />
    <NodePosition ID="13" X="603" Y="73" />
    <NodePosition ID="18" X="326" Y="383" />
    <NodePosition ID="21" X="308" Y="177" />
    <NodePosition ID="26" X="225" Y="75" />
    <NodePosition ID="29" X="636" Y="423" />
    <NodePosition ID="33" X="732" Y="421" />
  </Nodes>
</Layout>
```

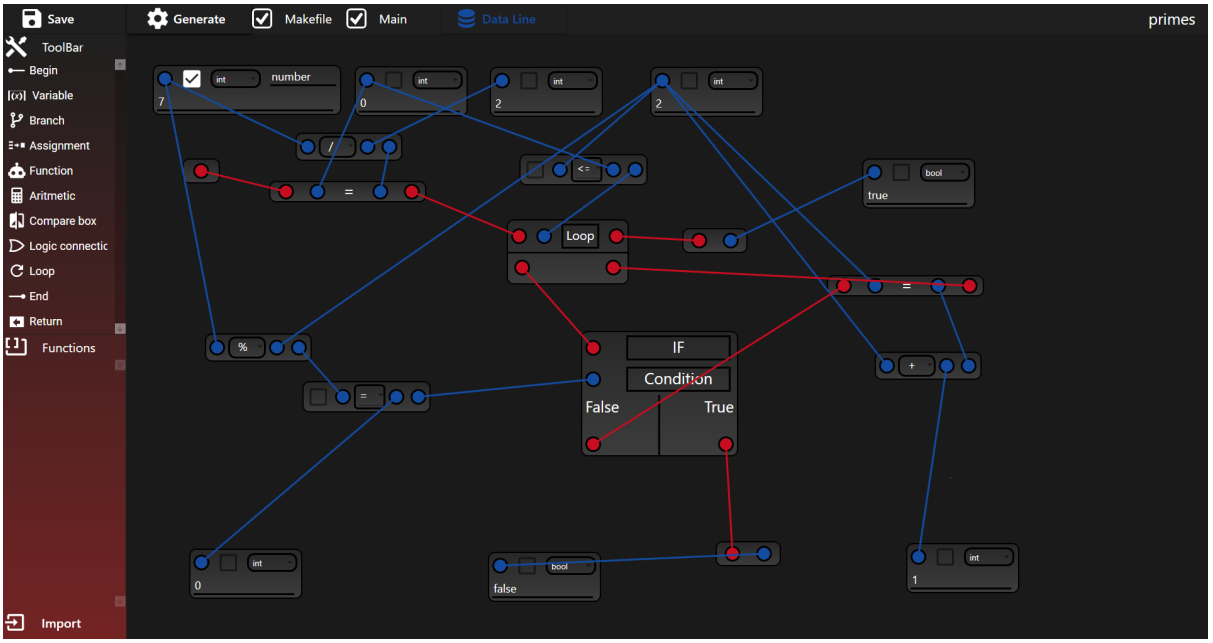
A másik generált fájlban már részletesebb leírást látunk a projektről. Láthatjuk az egyes funkcionális elemek típusát, azonosítóját és egyéb specifikus paramétereket. Továbbá a függvény felépítését megadó adat és vezérlési kapcsolatok tagjeit is itt lelhetjük fel. Összefoglalva tehát ez a függvény struktúrális leírását adja meg, míg az előző fájl a felületen elfoglalt helyüket. Ennek a megoldásnak a célja a tiszta funkcionalitás és a dizájn elszeparálása.

```xml
<?xml version="1.0" encoding="utf-8"?>
<Workspace xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Imports />
  <NodeArray>
    <Node xsi:type="Branch" ID="9">
      <FlowIn ID="6" Description="Branch_FlowIn" />
      <FlowOutTrue ID="7" Description="Branch_FlowOutTrue" />
      <FlowOutFalse ID="8" Description="Branch_FlowOutFalse" />
      <Condition ID="5" Name="Branch_Condition" Type="BOOL" />
    </Node>
    <Node xsi:type="Variable" ID="11" IsParam="true" Name="num" Type="INT" Value="5">
      <Connection ID="10" Name="VariableParam" Type="INT" />
    </Node>
    <Node xsi:type="BeginNode" ID="13">
      <FlowOut ID="12" Description="Begin Flow Out" />
    </Node>
    <Node xsi:type="CompareBox" ID="18" Negated="false">
      <dataOut ID="15" Name="CompareBox_DataOut" Type="BOOL" />
      <dataInLeft ID="16" Name="CompareBox_DataInLeft" Type="BOOL" />
      <dataInRight ID="17" Name="CompareBox_DataInRight" Type="BOOL" />
      <arithOperator>_EQ</arithOperator>
    </Node>
    <Node xsi:type="Variable" ID="21" IsParam="false" Name="" Type="INT" Value="0">
      <Connection ID="20" Name="VariableParam" Type="INT" />
    </Node>
    <Node xsi:type="ReturnNode" ID="26">
      <FlowIn ID="24" Description="Function Flow End" />
      <ReturnValue ID="25" Name="ReturnNode_ReturnValue" Type="BOOL" />
    </Node>
    <Node xsi:type="ReturnNode" ID="29">
      <FlowIn ID="27" Description="Function Flow End" />
      <ReturnValue ID="28" Name="ReturnNode_ReturnValue" Type="BOOL" />
    </Node>
  </NodeArray>
  <DataConnectionArray>
    <DataConnection ID="19" FlowInID="10" FlowOutID="16" />
    <DataConnection ID="22" FlowInID="20" FlowOutID="17" />
    <DataConnection ID="23" FlowInID="15" FlowOutID="5" />
  </DataConnectionArray>
  <FlowConnectionArray>
    <FlowConnection ID="14" FlowInID="12" FlowOutID="6" />
  </FlowConnectionArray>
  <Name>Function</Name>
  <ReturnValue>INT</ReturnValue>
</Workspace>
```

Komplex tesztesetek

Első teszt:

- Teszteset leírása
  - A függvény visszadja egy számról hogy prímszám-e vagy sem.
- Bemenet
  - szam : int -> Az adott szám amelyről el szeretnénk dönteni, hogy prím-e.
- Visszatérési érték:
  - bool ->
    - TRUE -> A szám prím
    - FALSE -> A szám nem prím
- A teszt függvény VisCon-ban való megvalósítása:



- A program által generált C kód:

```
bool primes(int number) {
    int variable5 = 0;
    int variable12 = 2;
    int variable23 = 2;
    int variable54 = 0;
    int variable73 = 1;
    bool variable94 = false;
    bool variable97 = true;


    variable5 = (number / variable12);
    while((variable23 <= variable5))
    {
        if(((number % variable23) == variable54))
        {
            return variable94;
        }
        else
        {
            variable23 = (variable23 + variable73);
        }
    }
    return variable97;
}
```

- Generált kód tesztelése
    - 1. Teszt
        - Bemenet: 8
        - Kimenet:

```
D:\temalab\viscon\Vicon\Vicon\bin\Debug\primes\gen>main.exe
false
```

    - 2. Teszt
        - Bemenet 97
        - Kimenet:

```
D:\temalab\viscon\Vicon\Vicon\bin\Debug\primes\gen>main.exe
true
```

Második teszt:

- Teszteset leírása
  - FizzBuzz
    - A függvény egytől kezdve egy adott számig eldönti az összes számról, hogy oszthatóak-e hárommal vagy öttel. Amennyiben egy szám osztható hárommal a fizz kerül kiírásra, ha öttel osztható akkor buzz kerül ki, ha mindkét számmal osztható akkor fizzbuzz kerül kiírásra. Minden más esetben a szám kerül kiíratásra.
- Bemenet
  - szam : int -> Egy határérték, a függvény ezt a számot veszi felső határnak.
- Importált fájlok
  - helper.h:

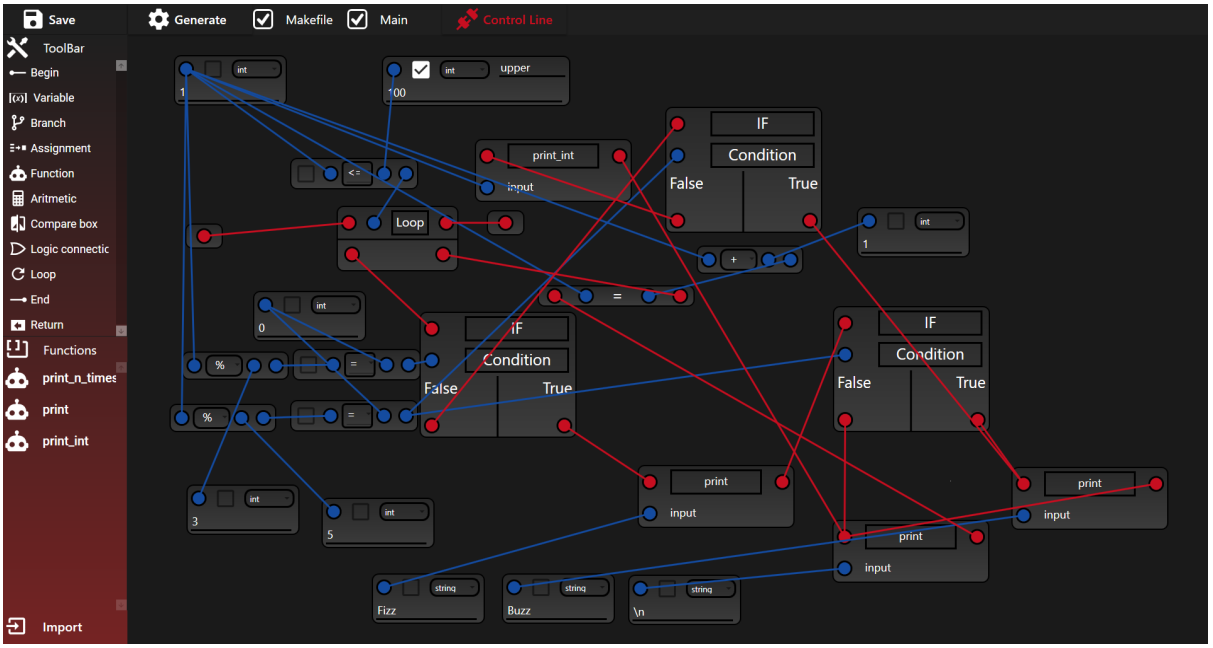
```c
#include <stdbool.h>
#include <stdio.h>

// [print_n_times:void::input:string,n:int]
void print_n_times(const char* input, int n) {
    for (int i = 0; i < n; i++) {
        printf(input);
    }
}

// [print:void::input:string]
void print(const char* input) {
    printf(input);
}

// [print_int:void::input:int]
void print_int(const int input) {
    printf("%d", input);
}
```

- Kimenet
    - A számok sorban kerülnek kiíratásra a megadott szabályok alapján.
- A teszt függvény VisCon-ban való megvalósítása:



- A program által generált C kód:

```c
#include "fizz_buzz_real.h"

void fizz_buzz_real(int upper) {
    int variable18 = 1;
    int variable38 = 3;
    int variable40 = 5;
    const char* variable56 = "Fizz";
    const char* variable58 = "Buzz";
    const char* variable60 = "\n";
    int variable111 = 1;
    int variable125 = 0;


    while((variable18 <= upper))
    {
        if(((variable18 % variable38) == variable125))
        {
            print(variable56);
            if(((variable18 % variable40) == variable125))
            {
                print(variable58);
            }
            print(variable60);
        }
        else
        {
            if(((variable18 % variable40) == variable125))
            {
                print(variable58);
            }
            else
            {
                print_int(variable18);
            }
            print(variable60);
        }
        variable18 = (variable18 + variable111);
    }
    return;
}


int main() {
    fizz_buzz_real(100);
}
```

- Generált kód tesztelése
    - 1. Teszt
        - Bemenet: 5
        - Kimenet:

```
D:\temalab\viscon\Vicon\Vicon\bin\Debug\fizz_buzz_real\gen>main.exe
1
2
Fizz
4
Buzz
```

    - 2. Teszt
        - Bemenet 45
        - Kimenet:

```
D:\temalab\viscon\Vicon\Vicon\bin\Debug\fizz_buzz_real\gen>main.exe
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
Fizz
22
23
Fizz
Buzz
26
Fizz
28
29
FizzBuzz
31
32
Fizz
34
Buzz
Fizz
37
38
Fizz
Buzz
41
Fizz
43
44
FizzBuzz
```