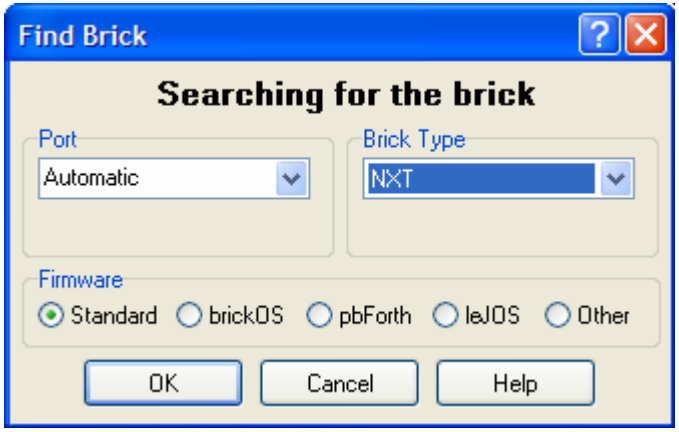
Programování LEGO robotů pomocí NXC

Textový jazyk odvozený od jazyka C běží v prostředí BricxCC na standardním firmwaru LEGO Mindstorms. Tato skutečnost je velmi příjemná pro ty, kteří chtějí programovat jak v NXT-G, tak v NXT, protože s každou změnou programovacího prostředí nemusí do kostky nahrávat nový firmware. Práce s jazykem zkracujícím spojení „Not eXactly C“ je velmi příjemná a programátor alespoň trochu znalý jazyka C si díky téměř stejné sémantice v tomto prostředí zvykne programovat velmi snadno. Další výhodou je, že se jedná o freewarovou aplikaci. Jako nevýhodu bych uvedl někdy nepříliš snadné debugování programů. Na rozdíl od NXT-G se jedná o čistě textové programování bez grafických prvků.

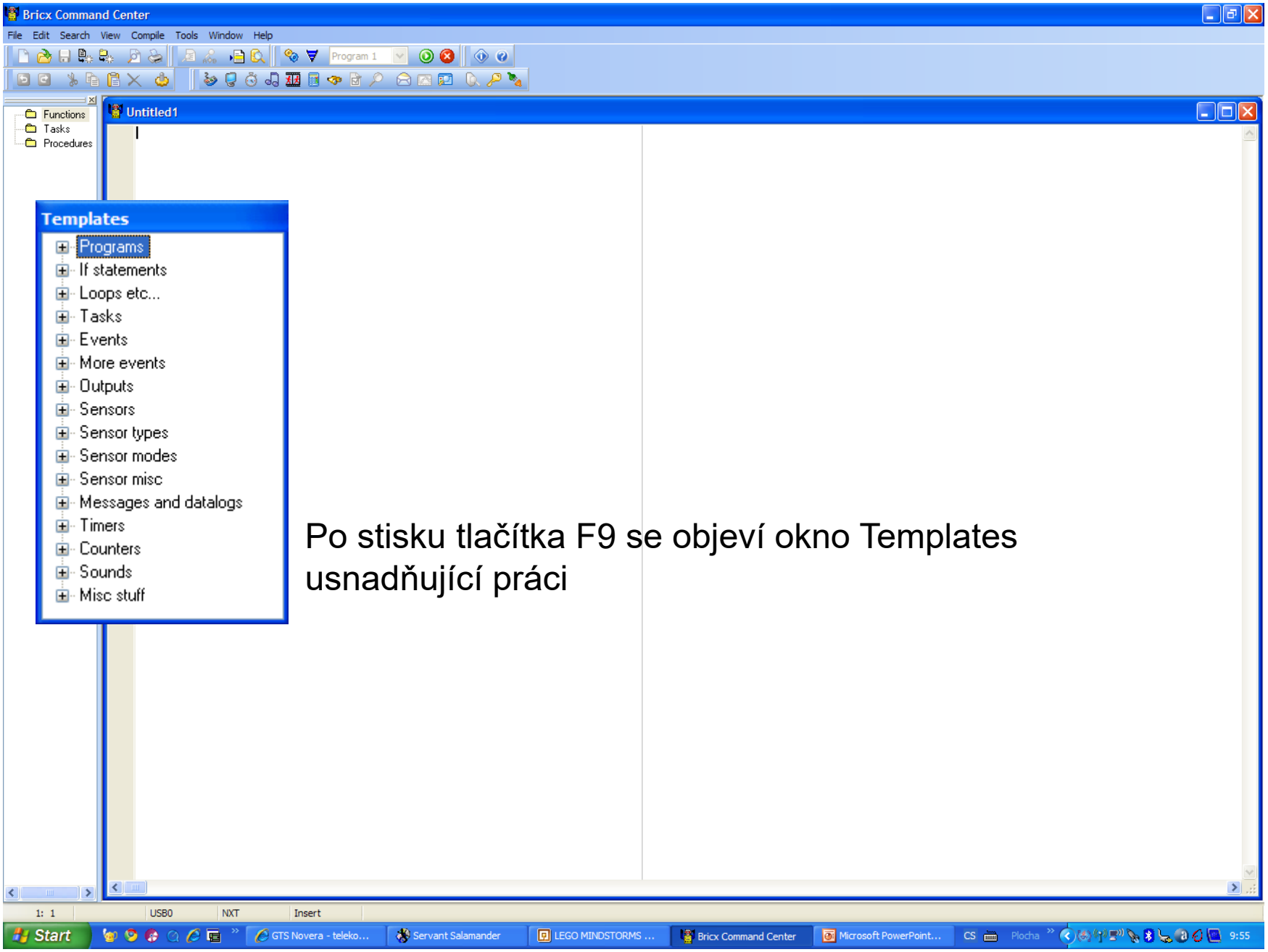
Vývojové prostředí jazyka zdarma ke stažení na:

http://bricxcc.sourceforge.net/

K dispozici je tutoriál k instalaci i programování robota.



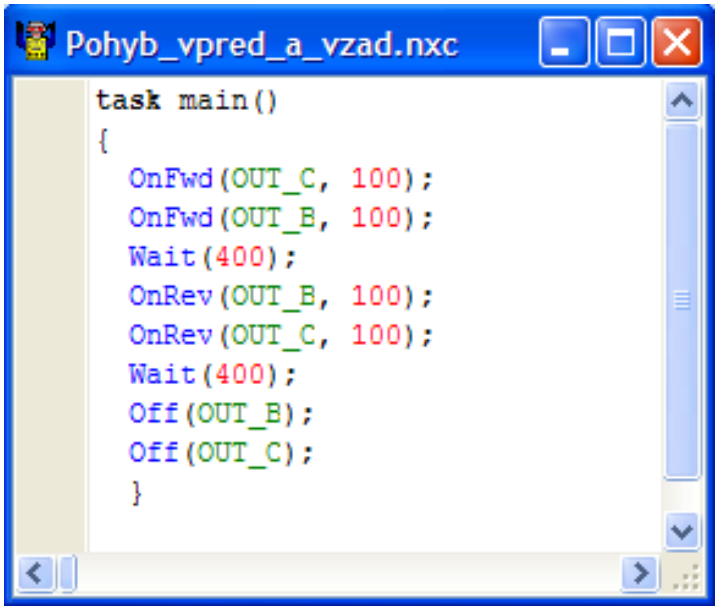
Před vlastním spuštěním programu Bricx Command Center je třeba provést základní nastavení.



Po stisku tlačítka F9 se objeví okno Templates usnadňující práci

Obrazovka Bricx Command Center

Psaní programu:
Jdeme psát nový program. Tedy stiskněme **New File** tlačítko pro vytvoření nového, prázdného okna. Do okna napište příklad uvedený na následujícím obrázku.



Na první pohled to vypadá velmi složitě, a tak se na něj podíváme zblízka. Program v NQC sestává z úloh (anglicky task). Náš program má jen jednu úlohu, nazvanou main. Každý program musí mít úlohu nazvanou main (anglicky hlavní), a tato úloha je zpracovávána při stisknutí tlačítka „RUN“. Úloha se skládá z množství příkazů, také nazývaných programové kroky (anglicky statement). Všechny programové kroky jsou pomocí složených závorek (tedy znaku { a }) uzavřeny do skupiny, aby bylo jasné, že všechny patří k této úloze. Každý programový krok je ukončen středníkem. Díky tomu je jasné, kde jeden programový krok končí a začíná druhý. Takže v podstatě každá úloha má strukturu jako v uvedeném příkladu.

Program je složen z několika programových řádků (kroků). Nyní je probereme jeden po druhém:

OnFwd(OUT_C, 100); Tento řádek říká robotu, aby zapnul výstup C *(On Forward Output C)*, tedy motor připojený na výstup označený „C“ pro pohyb vpřed. Bude se pohybovat maximální rychlostí, nastaveno jako druhý parametr tohoto příkazu 100“.

Wait(400); Nyní je čas chvíli počkat *(Wait)*. Řádek říká, aby se počkalo 0,4 sekundy. Číslo mezi závorkami, udává počet „tiků“. Každý „tik“ trvá cca 1/1000 sekundy, takže můžete velice přesně určovat dobu čekání. Ted tedy po 0,4 sekundy program nic nedělá (tzv. „spí“) a robot proto pokračuje v pohybu vpřed.

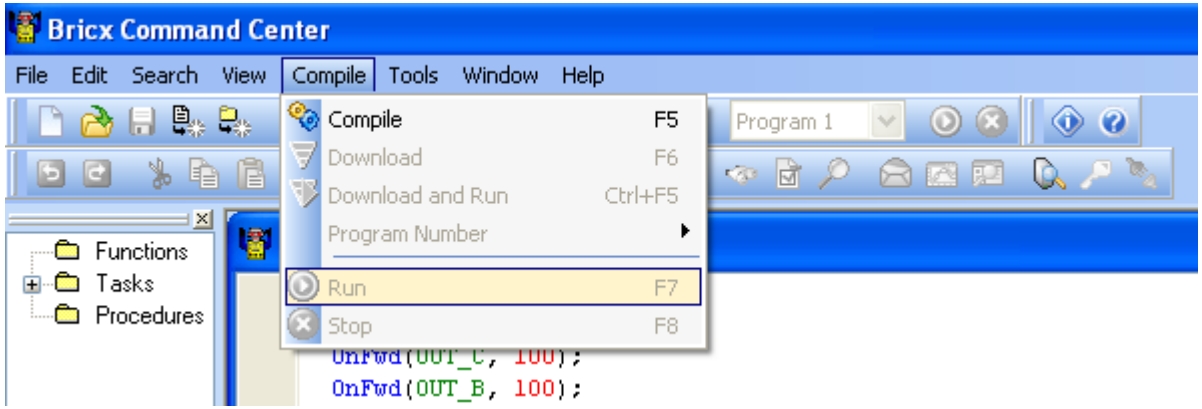
OnRev(OUT_C, 100); Robot už ujel dost daleko a tak je čas říci mu, aby jel v opačném směru *(On Reverse Output C)*, tedy zpět.

Off(OUT_C); Vypnutí *(Off)* motoru C.

Pravděpodobně jste si všimli barevného značení při psaní programu. Barvy se objevují automaticky. Vše co je v modré barvě, jsou příkazy pro robota, názvy motoru nebo jiných věcí které robot zná. Slovo „task“ je vypsáno tučně, protože je důležitým (rezervovaným) slovem NXC. Další důležitá slova se také objevují tučně, jak později uvidíme. Barvy jsou užitečné, protože už při psaní uvidíme, že jsme neudělali chybu.

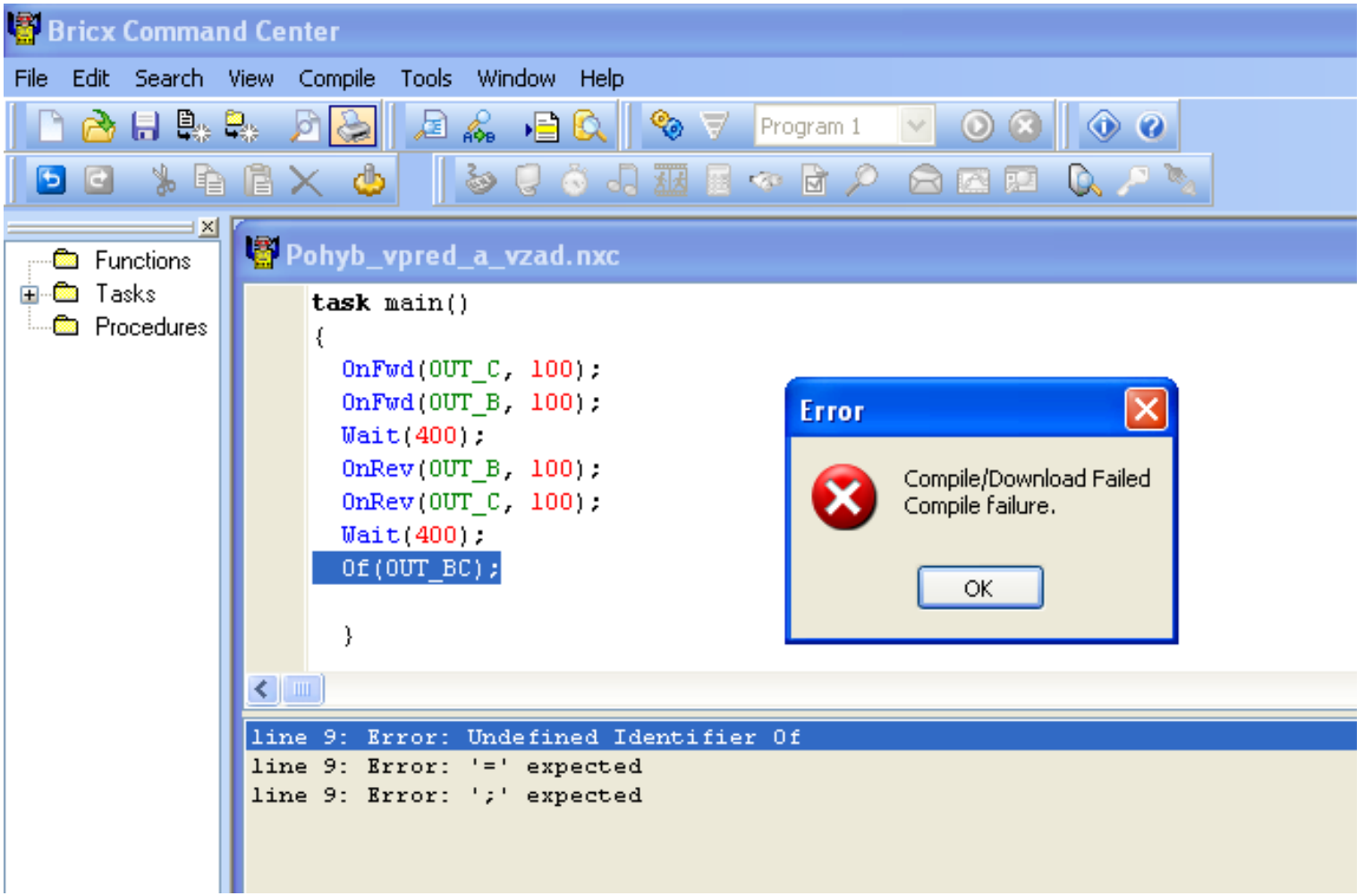
Uložení programu:
Po napsání programu je nutné program uložit Save As.. Pokud ponecháte název programu Untitled, program se normálně zkompiluje a nahraje do NXT kostky, ale nefunguje.

Spuštění programu:
Po napsání programu je nutné tento program zkompilovat (Compile - F5) a nahrát do NXT kostky (Download – F6) nebo provést obojí současně (Download and Run - CTRL F5).



Tady můžeme vidět od leva doprava příkazy Compile, Download, START a STOP

Chyby v programu:
Při psaní programu často vznikají chyby, které překladač při kompilování objeví a upozorní na ně, viz následující příklad.



Definování konstant, zatáčení, opakování příkazů, přidávání komentářů:

Pohyb_vpred_otoceni_opakovani_komentar.nxc

```
#define MOVE_TIME 400     // čas pro přímý pohyb
#define TURN_TIME 100     // čas pro otočení o 90 stupňů

task main()
{
  repeat (4)              // udělej čtverec
  {
  OnFwd(OUT_C, 100);      // jeď vpřed
  OnFwd(OUT_B, 100);
  Wait(MOVE_TIME);
  OnRev(OUT_C, 100);      // otoč se o 90 stupňů
  Wait(TURN_TIME);
  }
  Off(OUT_B);             // vypnutí motorů
  Off(OUT_C);
  }
```

#define Tento příkaz umožňuje definovat konstantu, která může být použita kdekoliv v programu.

Repeat() Příkaz umožňující opakování (počet opakování je číslo v závorkách).

Aby Váš program byl ještě srozumitelnější, je vhodné k němu přidat komentáre. Vložíte-li kdekoliv na řádek dvojznak „//“ (dvě lomítka), bude vše za tímto označením překladačem ignorováno a může sloužit jako poznámka či komentář. Víceřádkový komentár muže být vložen mezi dvojznaky „/*“ a „*/“.

Použití proměnných:

Proměnné jsou jednou z nejdůležitějších částí každého programovacího jazyka. Proměnné jsou místa v paměti, do kterých můžeme ukládat hodnoty. Můžeme tuto hodnotu použít na různých místech a také ji měnit.

pohyb_po_spirale.nxc

```
#define TURN_TIME 100    // čas pro otočení o 90 stupňů

int move_time;            // definujeme proměnnou

task main()
{
  move_time = 500;         // nastavíme počáteční hodnotu proměnné

  repeat (10)              // udělej čtverec
  {
  OnFwd(OUT_C, 100);      // jeď vpřed
  OnFwd(OUT_B, 100);
  Wait(move_time);
  OnRev(OUT_C, 100);      // otoč se o 90 stupňů
  Wait(TURN_TIME);
  move_time += 500;        // zvýšení hodnoty proměnné
  }
  Off(OUT_B);             // vypnutí motorů
  Off(OUT_C);
  }
```

Vedle zvyšování hodnoty proměnné můžeme také násobit proměnnou číslem použitím operátoru *=, odčítat použitím -= a dělit s pomocí /=. (Poznamenejme jen, že výsledek dělení je zaokrouhlen na nejbližší celé číslo.) Také můžete sčítat jednu proměnnou s druhou a vytvářet mnohem komplikovanější výrazy.

pouziti_promennych.nxc

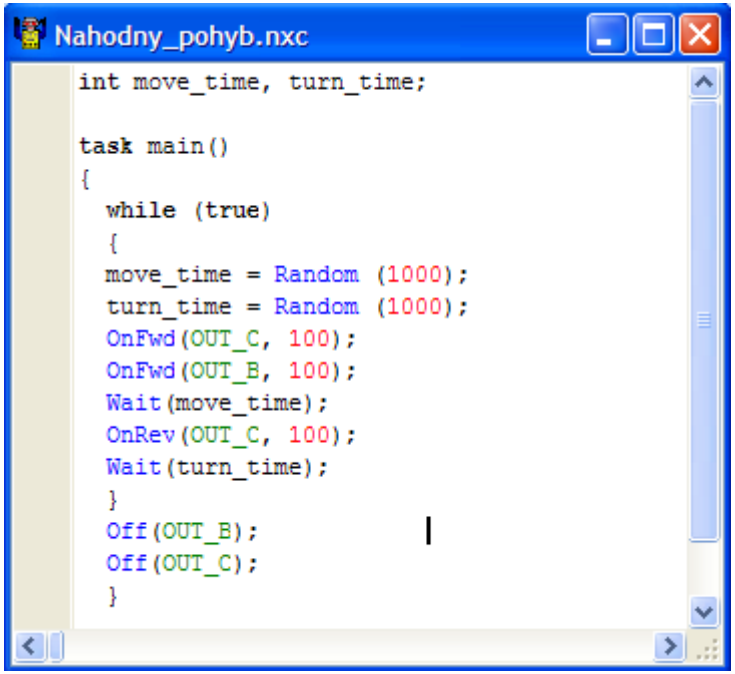
```
int aaa;
int bbb, ccc;

task main()
{
  aaa = 10;
  bbb = 20 * 5;
  ccc = bbb;
  ccc /= aaa;
  ccc -= 5;
  aaa = 10 * (ccc + 3);   // aaa je nyní rovno 80
  }
```

Náhodná čísla:

Ve všech předchozích příkladech jsme přesně určili, co má robot dělat. Vše je ale mnohem zajímavější, když nevíme, co bude robot dělat. Požadujeme určitou nahodilost v jeho pohybech. V NQC můžeme generovat náhodná čísla. Následující program je používá k jízdě robotu náhodným směrem. Robot jede vpřed po náhodnou dobu a pak provede otočení do náhodného směru.

Program definuje dvě proměnné a pak jim přiradí náhodná čísla. **Random(60)** znamená náhodné číslo mezi 0 a 60 (může to být i 0 i 60). Pokaždé bude hodnota jiná. (Poznamenejme, že při psaní programu jsme se mohli zbavit proměnných použitím konstrukce Wait(Random(60)).)

Také zde můžete vidět nový typ cyklu. Místo použití příkazu **repeat()** jsme napsali **while(true)**. Příkaz **while()** opakuje příkazy uvedené pod ním do té doby,



dokud je podmínka v závorkách splněna (pravdivá). Speciální slovo true je vždy pravda, takže řádky ve složených závorkách budou opakovány neustále, přesně jak jsme chtěli.

Příkaz ***if***:

Někdy potřebujeme, aby určitá část programu byla vykonána pouze v jisté situaci. V tomto případe použijeme příkaz ***if()***. Opět změníme program se kterým jsme až dosud pracovali, ale nyní to bude novým způsobem. Chceme, aby robot jel nejprve rovně a pak zatočil bud vlevo, nebo vpravo. K tomu opět potřebujeme náhodná čísla. Vezmeme náhodné číslo mezi nulou a jedničkou, tedy to bude buď nula, a nebo jednička. Jestliže bude číslo 0, otočíme se vpravo, jinak se otočíme vlevo.

Prikaz_if.nxc

```
#define MOVE_TIME 1000
#define TURN_TIME 1000

task main()
{
  while (true)
  {
   OnFwd(OUT_C, 100);
   OnFwd(OUT_B, 100);
   Wait(MOVE_TIME);
   if (RANDOM (1) == 0)
   {
    OnRev(OUT_C, 100);
   }
   else
   {
    OnRev (OUT_B, 100);
   }
   Wait(TURN_TIME);
   }
  }
```

==	je rovno s
<	je menší než
<=	je menší nebo rovno než
>	je větší než
>=	je větší nebo rovno než
!=	není rovno s

true	vždy pravda
false	vždy nepravda
ttt != 3	pravda pokud proměnná ttt není rovna třem
(ttt >= 5) && (ttt <= 10)	pravda pokud hodnota proměnné leží mezi 5 a 10
(aaa == 10) || (bbb == 10)	pravda pokud bud aaa, nebo bbb (nebo oba) jsou rovny 10

Příkaz ***do***:

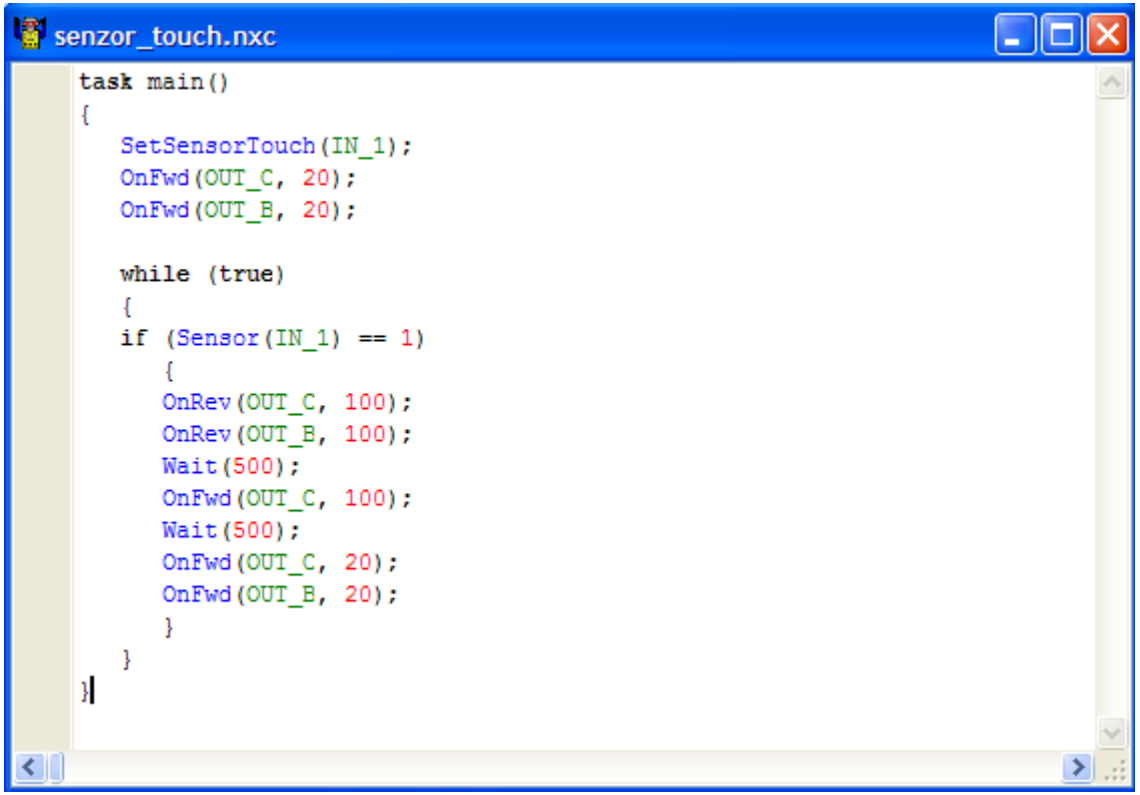
Příkazy mezi složenými závorkami jsou vykonávány tak dlouho, dokud je podmínka pravdivá (splněna).

Prikaz_do.nxc

```
int move_time, turn_time, total_time;

task main()
{
  total_time = 0;
  do
  {
   move_time = Random (1000);
   turn_time = Random (1000);
   OnFwd(OUT_C, 100);
   OnFwd(OUT_B, 100);
   Wait(move_time);
   OnRev(OUT_C, 100);
   Wait(turn_time);
   total_time += move_time; total_time += turn_time;
   }
   while (total_time < 5000);
   Off (OUT_B);
   Off (OUT_C);
  }
```

Touch sensor (dotykový senzor):

Jednou z vymožeností Lego robota je možnost připojit k nim senzory (čidla) a pak na ně nechat roboty reagovat.



Nyní zkusíme naprogramovat robota k vyhýbání se překážkám. Kdykoli robot do něčeho narazí, necháme ho couvnout maličko zpět, trochu se otočit a pak pokračovat.

Definování senzoru:

```
SetSensorEV3Type(IN_2, EV3_Touch);
SetSensorEV3Type(IN_1, EV3_Touch);
```

Vyčítání hodnoty senzoru:

```
SensorValueSlotted(IN_1, Touch_Touch);
SensorValueSlotted(IN_2, Touch_Touch);
```

Color sensor (barevný senzor):

Vedle dotykových senzorů také se stavebnicí MindStorms dostanete světelný senzor. Tento senzor měří množství světla přicházející z určitého směru. Světelný senzor také světlo vysílá. Díky tomu je možno namířit jej na nějaký objekt a změřit jeho odrazivost — množství světla, které se odrazí zpět k senzoru. To je velmi užitečné, když sestavíme robota na sledování čáry na podlaze.

senzor_light.nxc

```
#define TRESHOLD 40

task main()
{
   SetSensorLight(IN_3);
   OnFwd(OUT_BC, 75);

   while (true)
   {
   if (SENSOR_3 > TRESHOLD)
       {
       OnRev(OUT_C, 75);
       Wait(500);
       until (SENSOR_3 <= TRESHOLD);
       OnFwd(OUT_BC, 75);
       }
   }
}
```

Definování senzoru:

SetSensorEV3Type(IN_3, EV3_Color);

Vyčítání hodnoty senzoru:

SensorValueSlotted(IN_3, Color_Reflect);

Modes:

Color_Reflect - Returns amount of reflected red light. Values are in range from 0 to 100.

Color_ReflectRaw - Undocumented version of *Color_Reflect* returning sensor-internal values.

Color_Ambient - Returns amount of sunlight shining on the sensor. Values are in range from 0 to 100.

Color_ColorId - Identifies the color of the object that is in front of the color sensor (None, Black, Blue, Green, Yellow, Red, White, Brown).

Color_ColorRaw - Returns raw internal RGB measurements from the sensor.

Ultrasonic sensor (ultrazvukový senzor):

Ultrazvukový senzor pracuje jako sonar (ultrazvukový lokátor). V podstatě vysílá ultrazvukové vlny a měří dobu za jak dlouho se odrazí a vrátí tyto vlny zpět od objektu v dohledu. Na základě tohoto času je schopen určit vzdálenost k danému objektu a vyhnout se tak např. překážce. V příkladu jede robot rovně dokud neidentifikuje překážku ve vzdálenosti 15cm, vrátí se o kousek zpět, pootočí se a jede opět rovně, dokud neidentifikuje opět překážku.

senzor_ultrasonic.nxc

```
#define NEAR 15                 //cm

task main()
{
   SetSensorLowspeed(IN_4);

   while (true)
   {
   OnFwd(OUT_BC, 75);
   while (SensorUS(IN_4) > NEAR);
      Off(OUT_BC);
      OnRev(OUT_C, 100);
      Wait(800);
   }
}
```

Definování senzoru:

SetSensorEV3Type(IN_4, EV3_Sonic);

Vyčítání hodnoty senzoru:

SensorValueSlotted(IN_4, Sonic_Continuous_Mm);

Modes:

Sonic_Continuous_Mm - Returns the distance between an obstacle and the sensor in millimeters. If the measurement is out of range, *SONIC_NONE_MM* is returned.

Sonic_Continuous_In - Returns the distance between an obstacle and the sensor in inches. If the measurement is out of range, SONIC_NONE_IN is returned.

Sonic_Single_Mm - Makes one distance measurement during the mode switch and returns the measured value in millimeters. Out-of-range returns *SONIC_NONE_MM*.

Sonic_Single_In - Makes one distance measurement during the mode switch and returns the measured value in inches. Out-of-range returns *SONIC_NONE_IN*).

Sonic_Listen - Detects whether there is another ultrasonic sensor running nearby. Zero is returned if not.

Gyro sensor (gyroskopický senzor)

Definování senzoru:

SetSensorEV3Type(IN_4, EV3_Gyro);

Vyčítání hodnoty senzoru:

SensorValueSlotted(IN_4, Gyro_Angle);

Modes:

Gyro_Angle - Returns angular position difference from the last sensor mode change.

Gyro_Rate - Returns current angular velocity

Gyro_AngleRate - Returns both the angular position and velocity

Gyro_Fas - Undocumented sensor-internal value

For a quick illustration, we can use the following demo program. At the beginning of the main function the sensors are configured. Then, the sensor values are being read and then written on the screen in the main loop.

```
#define L(x) (((x)-1)*10)

void output(int line, string what, int value) {
  TextOut(0, L(line), what);
  NumOut(60, L(line), value);
}
 task main(){
  SetSensorEV3Type(IN_4, EV3_Touch);
  SetSensorEV3Mode(IN_4, Touch_Touch);
  SetSensorEV3Type(IN_3, EV3_Color);
  SetSensorEV3Mode(IN_3, Color_Reflect);

  while (true) {
    int pressed   = SensorValueSlotted(IN_4, Touch_Only);
    int reflected = SensorValueSlotted(IN_3, Color_Only);

    output(1, "PRESS", pressed);
    output(2, "LIGHT", reflected);

    Wait(10);
  }
}
```

Úlohy a podprogramy:

ulohy_a_podprogramy.nxc

```
mutex moveMutex;

task move_square()
{
  while (true)
  {
  Acquire(moveMutex);
  OnFwd(OUT_AC, 75); Wait(1000);
  OnRev(OUT_AC, 75); Wait(500);
  Release(moveMutex);
  }
}

task check_sensors()
{
  while (true)
  {
  if (SENSOR_1 == 1)
   {
   Acquire(moveMutex);
   OnRev(OUT_AC, 75); Wait(500);
   OnFwd(OUT_A, 75); Wait(500);
   Release(moveMutex);
   }
  }
}

task main()
{
  Precedes(move_square, check_sensors);
  SetSensorTouch(IN_1);
}
```

Úlohy a podprogramy:

Podprogram.nxc

```
sub turn_around(int pwr)
{
 OnRev(OUT_C, pwr); Wait(900);
 OnFwd(OUT_AC, pwr);
}

task main()
{
  OnFwd(OUT_AC, 75);
  Wait(1000);
  turn_around(75);
  Wait(2000);
  turn_around(75);
  Wait(1000);
  turn_around(75);
  Off(OUT_AC);
}
```

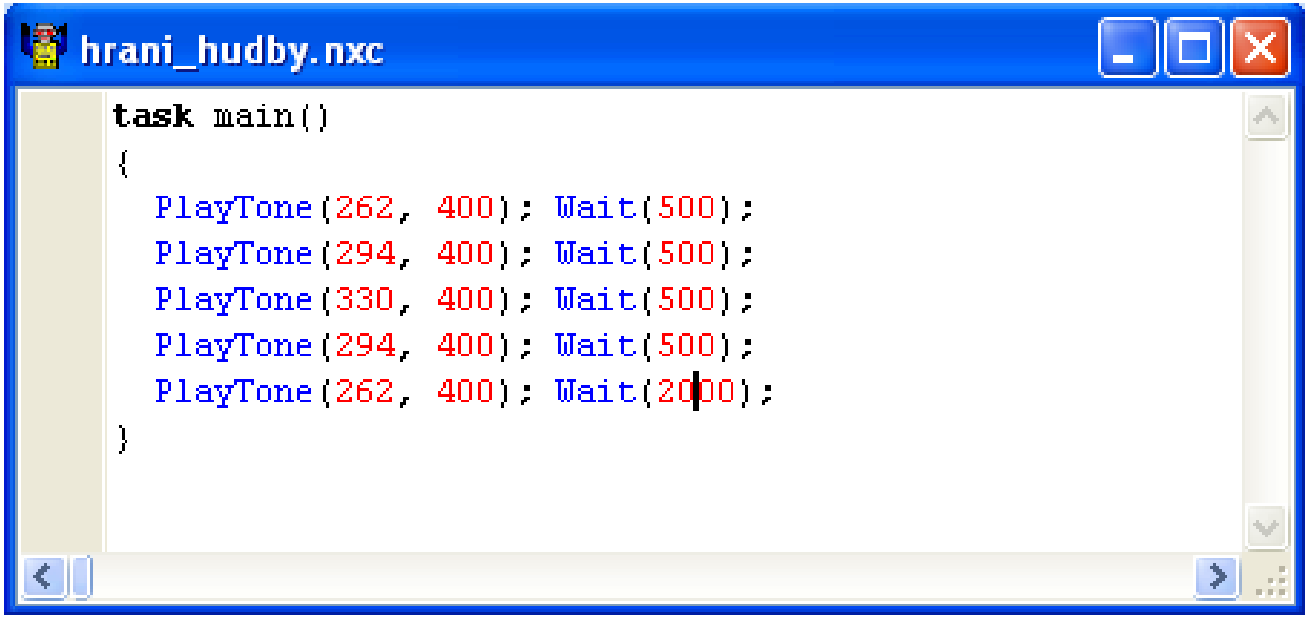
Hraní hudby:

NXT má vestavěný reproduktor schopný vydávat zvuky a dokonce hrát jednoduchou hudbu. To je užitečné zejména, když chcete, aby Vám NXT řeklo, že se něco děje. Také ale může být zábavné mít robota hrajícího hudbu, zatímco jezdí kolem.

	Oktáva							
Tón	1	2	3	4	5	6	7	8
G#	52	104	208	415	831	1661	3322	
G	49	98	196	392	784	1568	3136	
F#	46	92	185	370	740	1480	2960	
F	44	87	175	349	698	1397	2794	
E	41	82	165	330	659	1319	2637	
D#	39	78	156	311	622	1245	2489	
D	37	73	147	294	587	1175	2349	
C#	35	69	139	277	554	1109	2217	
C	33	65	131	262	523	1047	2093	4186
B	31	62	123	247	494	988	1976	3951
A#	29	58	117	233	466	932	1865	3729
A	28	55	110	220	440	880	1760	3520

Tabulka 1: frekvence různých tónů

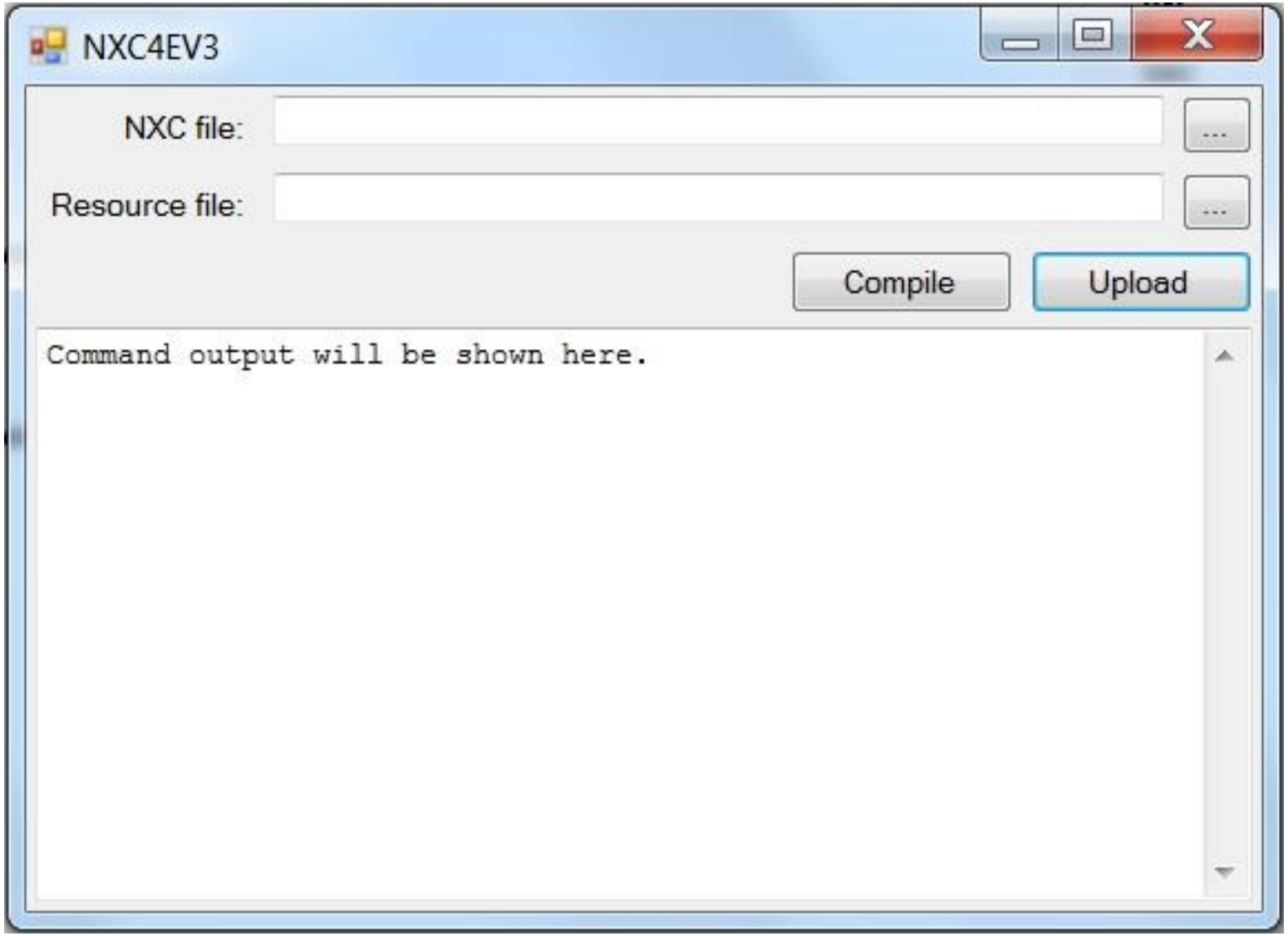
Pro tvorbu zajímavější hudby má NXC příkaz PlayTone(). Ten má dva argumenty. První udává frekvenci, druhý délku tónu (v„ticích“ dlouhých 1/1000 sekundy podobně, jako v příkazu wait). Tabulka1 zobrazuje užitečné frekvence. Jak jsme již uvedli v odstavci o zvucích, ani zde NXT nečeká, až nota skončí. Takže pokud hrajete hodně not za sebou, přidejte(raději o něco delší) příkaz wait() mezi nimi. Program je v následujícím příkladu.

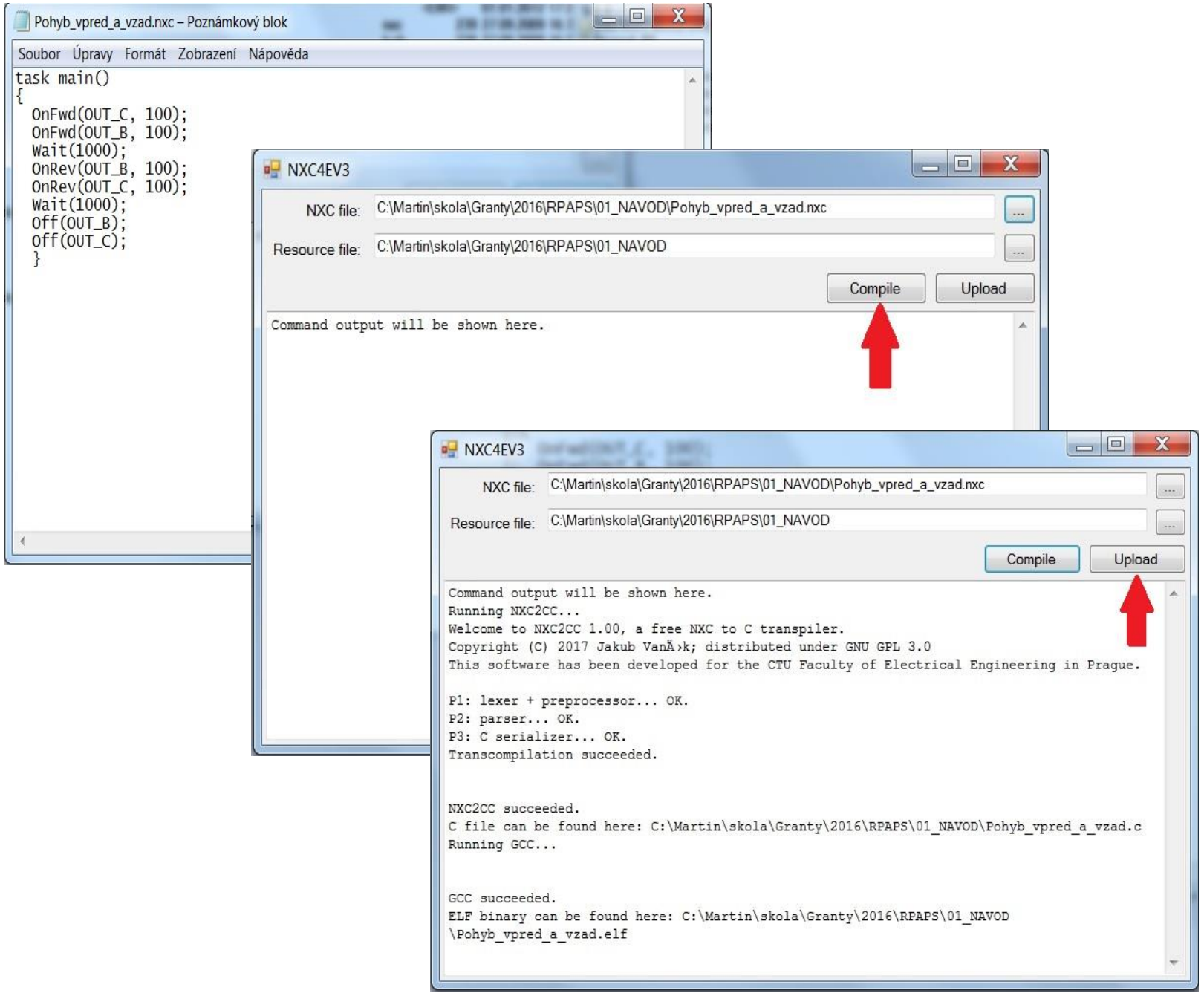


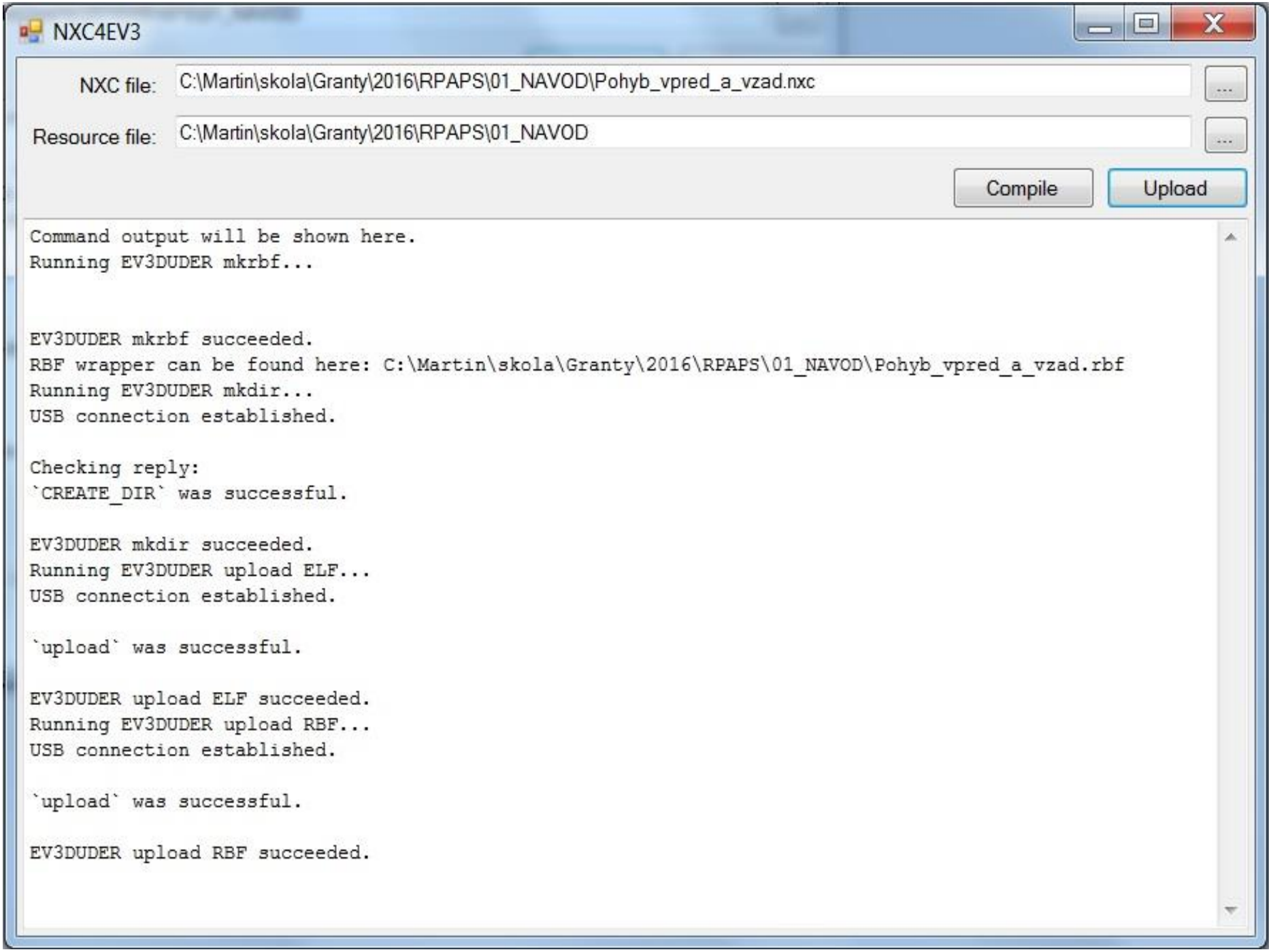
NXC4EV3

NXC4EV3 je nástroj pro spouštění programů napsaných v programovacím jazyce NXC na EV3 kostce. A jak program NXC4EV3 použít? Je to velice jednoduché:

1. Otevřete program NXC4EV3
2. Vyberte zdrojový soubor v jazyce NXC pomocí tlačítka vedle pole pro „NXC file“ – použijeme příklad z předchozí kapitoly pro pohyb vpřed a vzad). „Resource file“ se doplní automaticky. Pro editaci zdrojových kódů třeba používat např. původní Bricx Control Center, NXC4EV3 se stará pouze o kompatibilitu s EV3.
3. Pro sestavení klepněte na „Compile“. Pokud překlad proběhl bez problémů, můžete pokračovat dále.
4. Pro nahrání na EV3 kostku klepněte na „Upload“. Kostka musí být k počítači připojena přes USB. Pokud nahrání bylo úspěšné, program naleznete na kostce v adresáři, který se jmenuje dle vstupního NXC souboru.


NXC4EV3
NXC file:
...
Resource file:
...
Compile
Upload
Command output will be shown here.


Pohyb_vpred_a_vzad.nxc – Poznámkový blok
Soubor Úpravy Formát Zobrazení Nápověda
task main()
{
OnFwd(OUT_C, 100);
OnFwd(OUT_B, 100);
Wait(1000);
OnRev(OUT_B, 100);
OnRev(OUT_C, 100);
Wait(1000);
Off(OUT_B);
Off(OUT_C);
}
NXC4EV3
NXC file:
C:\Martin\skola\Granty\2016\RPAPS\01_NAVOD\Pohyb_vpred_a_vzad.nxc
Resource file:
C:\Martin\skola\Granty\2016\RPAPS\01_NAVOD
Compile
Upload
Command output will be shown here.
NXC4EV3
NXC file:
C:\Martin\skola\Granty\2016\RPAPS\01_NAVOD\Pohyb_vpred_a_vzad.nxc
Resource file:
C:\Martin\skola\Granty\2016\RPAPS\01_NAVOD
Compile
Upload
Command output will be shown here.
Running NXC2CC...
Welcome to NXC2CC 1.00, a free NXC to C transpiler.
Copyright (C) 2017 Jakub VanÄ›k; distributed under GNU GPL 3.0
This software has been developed for the CTU Faculty of Electrical Engineering in Prague.
P1: lexer + preprocessor... OK.
P2: parser... OK.
P3: C serializer... OK.
Transcompilation succeeded.
NXC2CC succeeded.
C file can be found here: C:\Martin\skola\Granty\2016\RPAPS\01_NAVOD\Pohyb_vpred_a_vzad.c
Running GCC...
GCC succeeded.
ELF binary can be found here: C:\Martin\skola\Granty\2016\RPAPS\01_NAVOD
\Pohyb_vpred_a_vzad.elf


NXC4EV3
NXC file:
C:\Martin\skola\Granty\2016\RPAPS\01_NAVOD\Pohyb_vpred_a_vzad.nxc
...
Resource file:
C:\Martin\skola\Granty\2016\RPAPS\01_NAVOD
...
Compile
Upload
Command output will be shown here.
Running EV3DUDER mkrbf...
EV3DUDER mkrbf succeeded.
RBF wrapper can be found here: C:\Martin\skola\Granty\2016\RPAPS\01_NAVOD\Pohyb_vpred_a_vzad.rbf
Running EV3DUDER mkdir...
USB connection established.
Checking reply:
`CREATE_DIR` was successful.
EV3DUDER mkdir succeeded.
Running EV3DUDER upload ELF...
USB connection established.
`upload` was successful.
EV3DUDER upload ELF succeeded.
Running EV3DUDER upload RBF...
USB connection established.
`upload` was successful.
EV3DUDER upload RBF succeeded.