EV3RT C++ API dokumentace

Vydání 1.0

Jaroslav Páral

kvě 30, 2017

Obsah

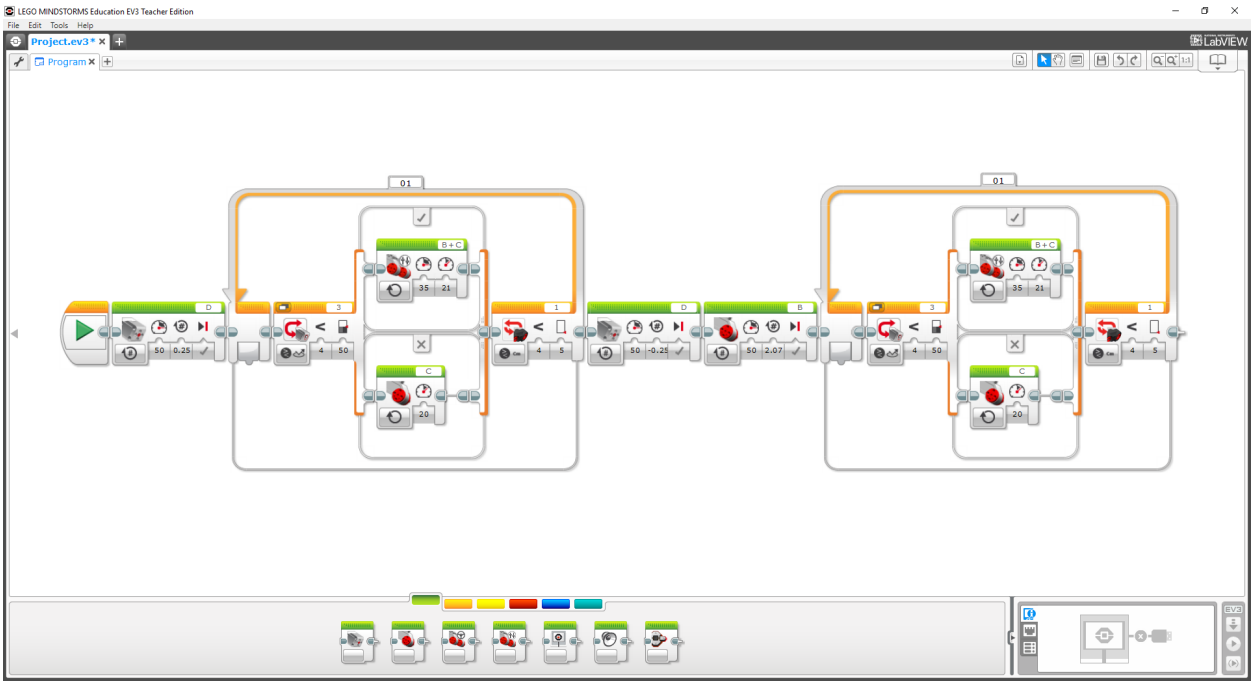
EV3RT C++ API vzniklo jako C++ nástavba EV3RT C API (ev3api), které je standardní součástí systému EV3RT.

Systém EV3RT je port japonského real-time operačního systém TOPPERS/HRP2 pro stavebnici LEGO MINDSTORMS EV3.

Mezi hlavní přednosti systému EV3RT patří:

- malá velikost systému (do 5 MB) i uživatelských aplikací (do 1 MB)
- velmi rychlý start (do 5 sekund) a prakticky okamžité vypnutí
- jednodušší úprava a doprogramování vlastních funkcí do jádra systému
- podpora dynamické alokace paměti
- preemptivní multitasking a rychlé přepínání tasků (do 8 μs)
- multiplatformní

Hlavním cílem EV3RT C++ API je umožnit jednoduchý přechod uživatelům zvyklým na standardní LEGO vývojové prostředí (LEGO MINDSTORMS EV3 Software) Proto jim je celé API přizpůsobeno a většina funkcí se jmenuje a chová stejně jako v originální vývojovém prostředí.



Contents:

KAPITOLA 1

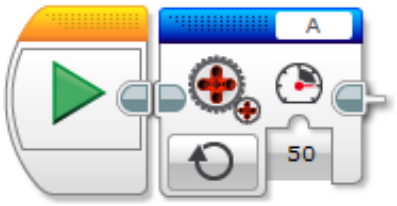
Motory

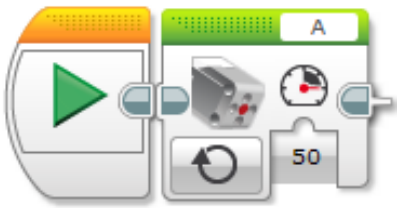
Motory jsou jednou ze základních komponent robota a proto s nimi začneme. Nejprve je potřeba vytvořit si instanci motorů:

```
ev3cxx::Motor motor(ev3cxx::MotorPort::A, ev3cxx::MotorType::LARGE);
```

Vytvořili jsme objekt `motor`, který je nastaven na port `A` a typ `LARGE`.

K dispozici máme všechny motorové porty na *Bricku* : `A`, `B`, `C` a `D`. U typů máme 3 volby, které odpovídají stejným blokům v originálním LEGO Softwaru: `UNREGULATED`, `MEDIUM` a `LARGE`.





- neregulované motory (`UNREGULATED`): u motorů se nastavuje jen výkon, změny zatížení (jízda do kopce) budou značně ovlivňovat rychlost

- regulované motory střední a velké (`MEDIUM` a `LARGE`): u motorů se nastavuje rychlost a motor se tuto rychlost snaží udržovat, upravuje tak výkon v závislosti na okolním prostředí (nerovnosti, překážky, atd.)

Při inicializaci je potřeba se rozhodnout v jakém režimu budete chtít s motorem pracovat.

Poznámka:

Pokud nebude řečeno jinak:

- při zadání parametru mimo rozsah se automaticky nastavuje maximální/minimální povolená hodnota.
- výchozí hodnoty metod odpovídají standardním hodnotám v LEGO Softwaru.

Příklad: Rozsah povolených hodnot je v rozmezí od -100 do 100. Při zadání hodnoty -101, dojde k ořezání na hodnotu -100. Při zadání hodnoty 101, dojde k ořezání na hodnotu 100.

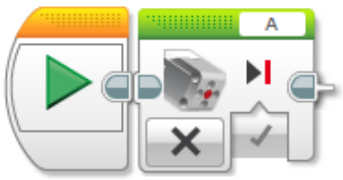
Výkon a rychlost

Poznámka: Parametry při nastavování rychlosti a výkonu.

- `speed`: rychlost motoru při jízdě; rozsah od -100 do 100
- `brake`: brzdění; `true` - motor brzdí, `false` - motor lze volně protáčet

---

off()

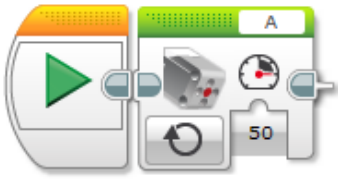



```
void off(bool brake = true)
```

Metoda `off()` zastavuje motor. Nastavuje rychlost nebo výkon (v závislosti na daném režimu) na 0. Jako parametr se předává zda má motor zároveň brzdit (`true`) nebo se volně protáčet (`false`). Ve výchozím stavu brzdí (`true`).

Použití: `motor.off();`

on()




```
void on(int power = 50)
```

Metoda `on()` nastavuje rychlost motoru. Jako parametr se předává požadovaná rychlost v rozsahu -100 až 100. Ve výchozím stavu je hodnota 50.

Použití: `motor.on(50);`

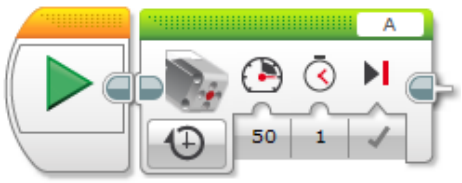
Čas a otáčky

Poznámka: Nové parametry při nastavování otáček.

- `speed`: rychlost motoru při běhu; rozsah od -100 do 100
- `time_ms`: čas v milisekundách, po který se bude motor točit;
- `degrees`: počet stupňů, o které se má motor otočit; lze otáčet i o více než +- 360 stupňů
- `rotations`: počet otáček, které má motor udělat; lze zadávat i desetinná čísla
- `brake`: brzdění po otočení o daný počet stupňů; `true` - motor po dotočení brzdí, `false` - motor lze volně protáčet
- `blocking`: když `true` - metoda blokuje další provádění programu, dokud nedokončí svůj úkol
- `wait_after_ms`: parametr, který nastavuje čekání po ukončení dané akce (jen v případě `blocking = true`); nechte výchozí hodnotu

---

onForSeconds()




```
void onForSeconds(int speed = 50,
                  unsigned int time_ms = 1000,
                  bool brake = true)
```

Metoda `onForSeconds()` nastavuje čas, jak dlouho se má motor točit. Jako parametry se předávají: `speed`, `time_ms`, `brake`.

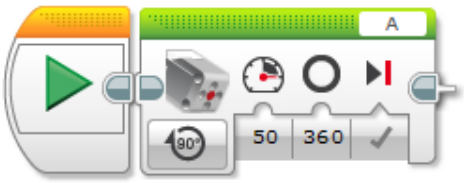
Použití: `motor.onForSeconds(50, 1000);`

Poznámka: LEGO pracuje se sekundami a desetinnými čísly, EV3CXX používá milisekundy a celá čísla

---

Varování: Metoda je vždy blokující. Další příkazy v programu se začnou vykonávat až metoda skončí.

onForDegrees()

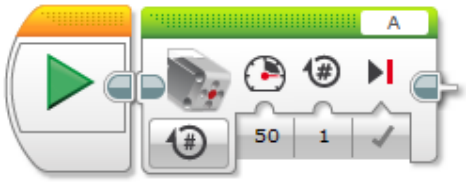



```
void onForDegrees(int speed = 50,
                  int degrees = 360,
                  bool brake = true,
                  bool blocking = true,
                  unsigned int wait_after_ms = 60)
```

Metoda `onForDegrees()` nastavuje počet stupňů, o které se má motor otočit. Jedna otáčka motoru odpovídá 360 stupňům. Jako parametry se předávají: `speed`, `degrees`, `brake`, `blocking`, `wait_after_ms`.

Použití: `motor.onForDegrees(50, 360);`

onForRotations()



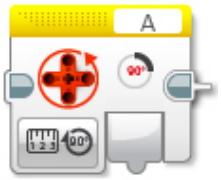

```
void onForRotations(int speed = 50,
                    float rotations = 1,
                    bool brake = true,
                    bool blocking = true,
                    unsigned int wait_after_ms = 60)
```

Metoda `onForRotations()` nastavuje počet otáček, o které se má motor otočit. Jako parametry se předávají: `speed`, `rotations`, `brake`, `blocking`, `wait_after_ms`.

Použití: `motor.onForRotations(50, 1);`

Čtení polohy a rychlosti

degrees()

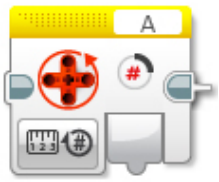



```
int degrees()
```

Metoda `degrees()` vrací polohu motoru ve stupních.

Použití: `motor.degrees();`

rotations()




```
float rotations()
```

Metoda `rotations()` vrací polohu motoru v otáčkách (`float` = desetinné číslo).

Použití: `motor.rotations();`

resetPosition()


```
void resetPosition()
```

Metoda `resetPosition()` vyresetuje pozici motoru (ovlivní metodu `degrees()` a `rotations()`).

Použití: `motor.resetPosition();`

currentPower()


```
int currentPower()
```

Metoda `currentPower()` vrací aktuální rychlost motoru.

Použití: `motor.currentPower();`

Dostupné metody

Po vytvoření objektu `motor` lze na něm volat metody:

- `off()` - vypne motory a začne brzdit
- `on()` - nastaví rychlost na motorech
- `onForSeconds()` - jede po zadanou dobu
- `onForDegrees()` - otočí se o daný počet stupňů
- `onForRotations()` - otočí se o daný počet otáček
- `degrees()` - vrátí aktuální počet stupňů na motoru
- `rotations()` - vrátí aktuální počet otáček na motoru
- `resetPosition()` - vyresetuje pozici motoru (ovlivní metodu `degrees()` a `rotations()`)
- `currentPower()` - vrátí aktuální rychlost motoru
- `getType()` - vrátí aktuálně nastavený typ motoru na daném portu v systému EV3RT

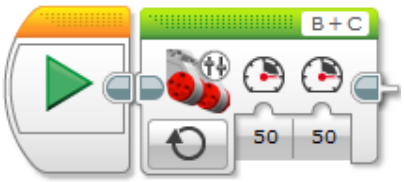
KAPITOLA 2

Tank motory

Pokud chceš řídit robota jako pásové vozidlo, můžeš využít třídu MotorTank

```
ev3cxx::MotorTank motors(ev3cxx::MotorPort::B, ev3cxx::MotorPort::C);
```

Vytvořili jsme si objekt `motors`, kde levý motor je nastaven na port `B` a pravý na port `C`. Tankový mód aktuálně podporuje jen `LARGE` motory. Ty se tedy nastavuji automaticky.



Poznámka:

Pokud nebude řečeno jinak:

- při zadání parametru mimo rozsah se automaticky nastavuje maximální/minimální povolená hodnota.
- výchozí hodnoty metod odpovídají standardním hodnotám v LEGO Softwaru.

Příklad: Rozsah povolených hodnot je v rozmezí od -100 do 100. Při zadání hodnoty -101, dojde k ořezání na hodnotu -100 Při zadání hodnoty 101, dojde k ořezání na hodnotu 100.

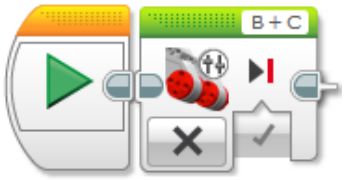
Výkon a rychlost

Poznámka: Parametry při nastavování rychlosti a výkonu.

- `left_speed`: rychlost levého motoru při jízdě; rozsah od -100 do 100
- `right_speed`: rychlost pravého motoru při jízdě; rozsah od -100 do 100

- `brake`: brzdění; `true` - motor brzdí, `false` - motor lze volně protáčet

off()

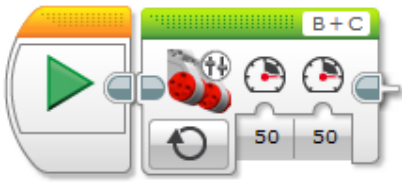



```
void off(bool brake = true)
```

Metoda `off()` zastavuje motory. Nastavuje rychlost na 0. Jako parametr se předává zda mají být motory zabrzděny (`true`) nebo se mají volně protáčet (`false`). Ve výchozím stavu brzdí (`true`).

Použití: `motors.off();`

on()




```
void on(int left_speed = 50, int right_speed = 50)
```

Metoda `on()` nastavuje rychlost motorů. Jako parametry se předávají rychlosti motorů v rozsahu -100 až 100. Ve výchozím stavu jsou `left_speed` a `right_speed` rovny hodnotě 50.

Použití: `motors.on(50, 50);`

Čas a otáčky

Poznámka: Nové parametry při nastavování otáček.

- `left_speed`: rychlost levého motoru při jízdě; rozsah od -100 do 100
- `right_speed`: rychlost pravého motoru při jízdě; rozsah od -100 do 100
- `time_ms`: čas v milisekundách, po který se budou motory točit;
- `degrees`: počet stupňů, o které se má motor otočit; lze otáčet i o více než +- 360 stupňů
- `rotations`: počet otáček, které má motor udělat; lze zadávat i desetinná čísla
- `brake`: brzdění po otočení o daný počet stupňů; `true` - motor po dotočení brzdí, `false` - motor lze volně protáčet
- `blocking`: když `true` - metoda blokuje další provádění programu, dokud nedokončí svůj úkol

- `wait_after_ms`: parametr, který nastavuje čekání po před zahájením dané akce (jen v případě `blocking = true`); nechte výchozí hodnotu

onForSeconds()


```
void onForSeconds(int left_speed = 50,
                  int right_speed = 50,
                  unsigned int time_ms = 1000,
                  bool brake = true)
```

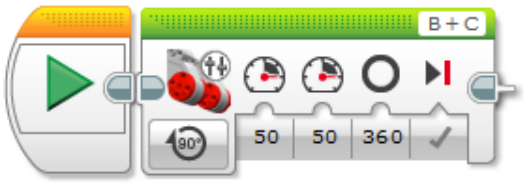
Metoda `onForSeconds()` nastavuje čas, jak dlouho se mají motory točit. Jako parametry se předávají: `left_speed`, `right_speed`, `time_ms`, `brake`.

Použití: `motors.onForSeconds(50, 50, 1000);`

Poznámka: LEGO Software pracuje se sekundami a desetinnými čísly, EV3CXX používá milisekundy a celá čísla

Varování: Metoda je vždy blokující. Další příkazy v programu se začnou vykonávat až metoda skončí.

onForDegrees()

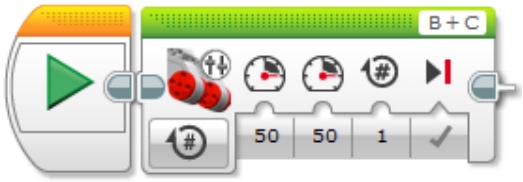



```
void onForDegrees(int left_speed = 50,
                  int right_speed = 50,
                  int degrees = 360,
                  bool brake = true,
                  bool blocking = true,
                  unsigned int wait_after_ms = 60)
```

Metoda `onForDegrees()` nastavuje počet stupňů, o které se má rychlejší motor otočit. Jedna otáčka motoru odpovídá 360 stupňům. Jako parametry se předávají: `left_speed`, `right_speed`, `degrees`, `brake`, `blocking`, `wait_after_ms`.

Použití: `motors.onForDegrees(50, 50, 360);`

onForRotations()




```
void onForRotations(int left_speed = 50,
                    int right_speed = 50
                    float rotations = 1,
                    bool brake = true,
                    bool blocking = true,
                    unsigned int wait_after_ms = 60)
```

Metoda `onForRotations()` nastavuje počet otáček, o které se má rychlejší motor otočit. Jako parametry se předávají: `left_speed`, `right_speed`, `rotations`, `brake`, `blocking`, `wait_after_ms`.

Použití: `motors.onForDegrees(50, 50, 1);`

leftMotor() a rightMotor()

```
Motor& rightMotor();
```

Přes tyto metody, lze ovládat jen jeden motor z páru. Nemusíte si tedy vytvářet nový objekt, pokud budete chtít v určitých situacích ovládat jen jeden motor. Metoda `leftMotor()` vrací instanci motoru, který byl při vytvoření objektu předán jako první, `rightMotor()` vrací druhý motor v pořadí.

Metody vrací instanci daného motoru a následně nad ní lze volat všechny metody dostupné ve třídě `Motor`.

Použití: `motors.rightMotor().onForDegrees(50, 1);`

Dostupné metody

Po vytvoření objektu `motor` lze na něm volat metody:

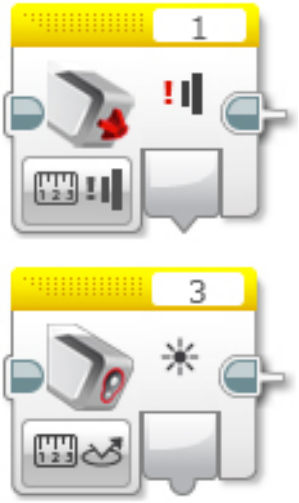
- `off()` - vypne motory a začne brzdit
- `on()` - nastaví rychlost na motorech
- `onForSeconds()` - jede po zadanou dobu
- `onForDegrees()` - otočí se o daný počet stupňů
- `onForRotations()` - otočí se o daný počet otáček
- `leftMotor()` - vrátí instanci levého motoru
- `rightMotor()` - vrátí instanci pravého motoru

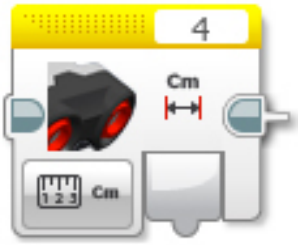
KAPITOLA 3

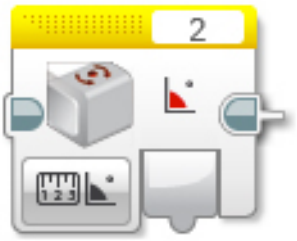
Senzory

Když už umíme ovládat motory, měli bychom se naučit pracovat se senzory. Pomocí senzorů můžeme získávat informace z okolí a reagovat na ně. Lze tak třeba řídit rychlost motorů, podle pozice robota na čáře nebo zastavit robota před překážkou. V EV3CXX jsou k dispozici všechny základní senzory z LEGO MINDSTORMS EV3.

- `TouchSensor` - dotykový senzor (detekce nárazu, překážky, STOP tlačítko)
- `ColorSensor` - barevný senzor (jízda po čáře, třídění dle barvy)
- `UltrasonicSensor` - ultrazvukový senzor (měření vzdálenosti od překážky nebo mantinelu)
- `GyroSensor` - gyro senzor (určení o kolik stupňů se robot otočil - jízda rovně)







Inicializace

Všechny senzory se inicializují:

```
//ev3cxx::nazev_tridy_senzoru nazev_objektu(ev3cxx::SensorPort::oznaceni_portu);
ev3cxx::TouchSensor touchS(ev3cxx::SensorPort::S1);
```

Vytvořili jsme tedy objekt `touchS`, která je nastavena na port číslo `S1`.

Na *Bricku* můžeme využít všechny porty pro senzory: `S1`, `S2`, `S3` a `S4`.

TouchSensor

Metody dostupné ve třídě `TouchSensor`:

- `isPressed()` - vrací `true` pokud je senzor zmáčklý
- `waitForPress()` - čekání, dokud se senzor nezmáčkne
- `waitForRelease()` - čekání, dokud se senzor neuvolní
- `waitForClick()` - čekání na zmáčknutí a uvolnění senzoru

isPressed()


```
int isPressed();
```

Vrací `true` v případě, že je dotykový senzor zmáčklý, jinak `false`.

waitForPress()


```
void waitForPress();
```

Program je pozastaven, dokud nebude dotykový senzor zmáčknut.

waitForRelease()


```
void waitForRelease();
```

Program je pozastaven, dokud nebude dotykový senzor uvolněn.

Varování: Nezapomínejte, že v běžném stavu může být dotykový senzor uvolněn. Volání této metody program pozastaví pouze pokud je v daný okamžik dotykový senzor zmáčknutý.

waitForClick()


```
void waitForClick();
```

Program je pozastaven, dokud neproběhne zmáčknutí a uvolnění dotykového senzoru.

ColorSensor

Barevný senzor může pracovat v několika režimech:

- `reflected()` - vrací naměřenou intenzitu odrazu
- `reflectedRawRgb()` - vrací naměřenou intenzitu odrazu pro jednotlivé barevné složky (RGB - červená, zelená, modrá)

- `ambient()` - vrací naměřenou intenzitu odrazu bez přisvětlení (vhodné pro kalibraci)
- `color()` - vrací rozpoznanou barvu

reflected()

```
int reflected();
```

Vrací naměřenou intenzitu odraženého světla z povrchu. Lze tak rozpoznat barvu povrchu a například tak detekovat černou čáru na bílém podkladu.

Rozsah výstupních hodnot je od 0 do 100.

Poznámka: Senzor si při své činnosti snímanou plochu přisvětluje vlastními světly, tak aby mohl lépe určit odrazivost povrchu a nebyl tolik závislý na okolním osvětlení. Přes metodu `ambient()` je možné určit odrazivost při vypnutém přisvětlení. Po odečtení této hodnoty od `reflected()` by měly být hodnoty za různých světelných podmínek pro stejné povrchy konstantní.

---

reflectedRawRgb()

```
rgb_raw_t reflectedRawRgb();
```

Vrací strukturu s naměřenými hodnotami jednotlivých barevných složek. Jako návratová hodnota je použita struktura `rgb_raw_t`, která obsahuje jednotlivé složky (r,g,b).

Poznámka: Tato metoda nemá odpovídající blok v LEGO Softwaru.

Příklad:

```
rgb_raw_t rgb_values;
rgb_values = colorS.reflectedRawRgb();

rgb_values.r; // RED value
rgb_values.g; // GREEN value
rgb_values.b; // BLUE value
```

ambient()

```
int ambient();
```

Vrací naměřenou intenzitu světla dopadajícího na senzor, ale **bez přisvětlení vlastními světly**. Vhodné např. pro kalibraci senzoru pro různá osvětlení. Více informací v poznámce u metody `reflected()`.

Rozsah výstupních hodnot je od 0 do 100.

color()

```
colorid_t color();
```

Vrací rozpoznanou barvu povrchu z výčtového typu `enum colorid_t`.

Hodnoty v typu `colorid_t`:

- `COLOR_NONE` - barva nerozpoznána
- `COLOR_BLACK` - černá barva
- `COLOR_BLUE` - modrá barva
- `COLOR_GREEN` - zelená barva
- `COLOR_YELLOW` - žlutá barva
- `COLOR_RED` - červená barva
- `COLOR_WHITE` - bílá barva
- `COLOR_BROWN` - hnědá barva

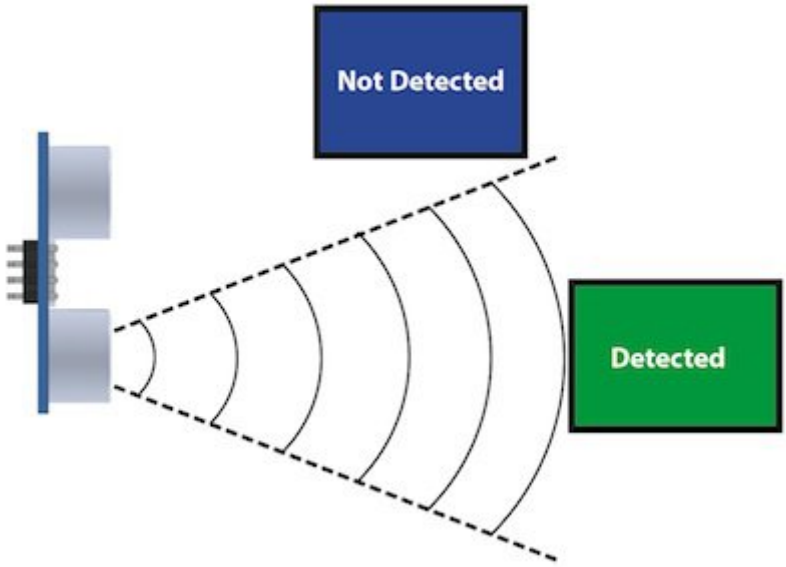
Příklad:

```
colorid_t color_value;
color_value = colorS.color();

if (color_value == COLOR_BLACK)
{
     // sensor on black color
}
```

UltrasonicSensor

Ultrazvukový senzor je primárně určen na měření vzdálenosti. Můžeme jej využít pro detekci překážky, určení vzdálenosti od mantinelu nebo i pro korekci jízdy.

Poznámka: Šíření ultrazvukových vln v prostoru



Obr. 3.1: Ultrazvukové vlny se od vysílače šíří v kuželu. To znamená, že s rostoucí vzdáleností od senzoru pokrývají větší plochu. Zároveň s tím ale klesá rozlišovací schopnost, proto senzor při větších vzdálenostech nedokáže zachytit předměty, které na blízko zachytí. To je podstatný rozdíl v porovnání s infra senzorem, jehož paprsky se šíří prakticky přímo (s mnohem menším rozptylem do stran).

Zdroj obrázku: http://arcbotics.com/products/sparki/parts/ultrasonic-range-finder/

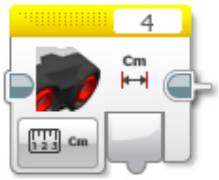
Ultrazvuk v EV3CXX poskytuje tyto metody:

- `centimeters()` - vrací naměřenou vzdálenost v centimetrech
- `millimeters()` - vrací naměřenou vzdálenost v milimetrech
- `inches()` - vrací naměřenou vzdálenost v palcích
- `inchesLine()` - vrací naměřenou vzdálenost v line (1/12 palce)
- `listen()` - vrací zda přijímá signál z jiného ultrazvukového vysílače

Varování: Ultrazvuk v EV3 umí měřit v rozsahu od 3 do 255 centimetrů. Pokud se budete pohybovat na hranici 3 centimetrů, může se stát, že ultrazvuk nedokáže danou vzdálenost změřit a místo hodnoty blízké 3 cm vrátí hodnotu rovnou maximální vzdálenosti => 255 cm.

Pamatujte na tuto vlastnost při návrhu a programování vašich robotů. Nejbezpečnějším řešením je umístit ultrazvuk tak, aby samotná konstrukce nedovolila menší vzdálenost než 4 a více centimetrů.

centimeters()




```
int centimeters();
```

Vrací naměřenou vzdálenost v centimetrech.

Rozsah měření je od 3 do 255.

Varování: Na rozdíl od LEGO Softwaru, v EV3CXX tato metoda pracuje v celých číslech. Pokud chcete vyšší přesnost použijte metodu `millimeters()`.

millimeters()

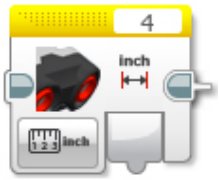
```
int millimeters();
```

Vrací naměřenou vzdálenost v milimetrech.

Rozsah měření je od 30 do 2550.

Poznámka: Tato metoda nemá odpovídající blok v LEGO Softwaru. Jelikož ultrazvuk v EV3 má rozlišení na milimetry a v LEGO Softwaru to řeší pomocí desetinných čísel, je v EV3CXX implementována tato metoda.

inches()




```
int inches();
```

Vrací naměřenou vzdálenost v palcích (1 palec = 2,54 cm).

Rozsah měření je od 1 do 100.

Varování: Na rozdíl od LEGO Softwaru, v EV3CXX tato metoda pracuje v celých číslech. Pokud chcete vyšší přesnost použijte metodu `inchesLine()`.

inchesLine()

```
int inchesLine();
```

Vrací naměřenou vzdálenost v linech (1 line = 1/12 palce).

Rozsah měření je od 10 do 1200.

Poznámka: Tato metoda nemá odpovídající blok v LEGO Softwaru. Jelikož ultrazvuk v EV3 má rozlišení na milimetry a v LEGO Softwaru to řeší pomocí desetinných čísel, je v EV3CXX implementována tato metoda.

---

listen()


```
int listen();
```

Senzor poslouchá a pokud zachytí ultrazvukový signál, od jiného vysílače, vrací `true`, jinak `false`.

GyroSensor

Gyroskop umožňuje změřit o kolik stupňů se robot otočil nebo jak rychle se otáčí. EV3 obsahuje jednoosý gyroskop a tak si při stavbě musíte vybrat v jaké rovině chcete měřit.

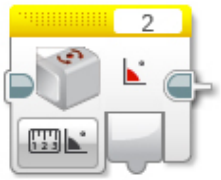
Gyroskop v EV3CXX poskytuje tyto metody:

- `angle()` - vrací aktuální úhel natočení ve stupních
- `rate()` - vrací aktuální rychlost otáčení ve stupních za vteřinu
- `reset()` - nastavuje počáteční polohu gyroskopu
- `resetHard()` - provádí úplný restart senzoru

Varování: Gyroskop v EV3 se občas zasekne a začne měnit aktuální úhel (ujíždět), i když se robot nehýbe. Většinou nepomůže standardní `reset()` a proto je v těchto případech potřeba provést `resetHard()`.

Při každém vytváření objektu ze třídy GyroSensor se provádí `resetHard()`. V tento moment se nesmí gyroskop pohybovat, jinak bude měřit špatně.

angle()




```
int angle();
```

Vrací aktuální úhel natočení vůči počáteční pozici (odchylku od počáteční pozice). Počáteční pozice se nastavuje při vytváření objektu nebo pomocí metody `reset()`.

Rozsah měření je od -32768 do 32767. Při překročení maximální nebo minimální hodnoty (např. 32767 + 1) začne gyroskop vracet hodnotu z druhého konce rozsahu (=> -32768).

Varování: Při použití metody `rate()` dochází k restartu počáteční polohy u metody `angle()`. Ta pak ukazuje opět od nuly.

rate()


```
int rate();
```

Vrací aktuální rychlost změny polohy ve stupních za sekundu.

reset()


```
void reset();
```

Nastavuje počáteční polohu gyroskopu pro metodu `angle()` a také kalibruje senzor. Při volání metody `reset()` by se Gyro senzor neměl vůbec hýbat. Jinak bude špatně měřit. Dejte pozor na vibrace a dojezdy setrvačností.

Metoda může v některých případech odstranit *ujíždění* aktuálního úhlu gyroskopu pro metodu `angle()`, ale ne vždy funguje.

resetHard()

```
void resetHard();
```

Provádí úplný restart senzoru. Měl by odstranit problém s ujížděním, kdy ačkoliv se Gyro senzor vůbec nehýbe, jeho poloha má konstantní přírůstek. V tento moment gyroskop nelze využívat a je potřeba jej restartovat.

Poznámka: Tato metoda nemá odpovídající blok v LEGO Softwaru.

Varování: Počítejte s tím, že `resetHard()` může trvat i několik sekund a po tuto dobu bude zastaven běh programu. Je tedy potřeba provádět úplný reset jen v nutných případech a na místech v programu, kde tato prodleva nebude vadit.

Během restartu se nesmi gyroskop pohybovat, jinak nebude měřit správně.

Poznámka: Implementace úplného restartu je v celku jednoduchá. Aby došlo k restartu, je potřeba přepnout gyroskop mezi režimy v jakých pracuje. První režim je měří úhel natočení (`angle()`) a druhý režim měří rychlost otáčení (`rate()`). Při přepínání mezi těmito režimy dochází k úplnému restartu gyroskopu. Pro stoprocentní funkčnost se v metodě `resetHard()` provádí vícenásobné přepínání (`angle()` => `reset()` => `rate()` => `reset()` => `angle()`).

Zdroj 1: https://bricks.stackexchange.com/questions/7115/how-can-ev3-gyro-sensor-drift-be-handled

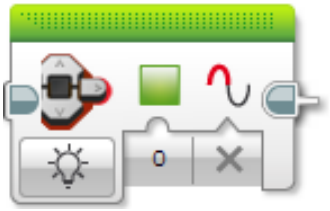
Zdroj 2: https://www.us.lego.com/en-us/mindstorms/community/robot?projectid=96894a3a-45db-48f9-9544-abf66f481b32

KAPITOLA 4

EV3 Brick

EV3 Brick obsahuje několik komponent, které má uživatel k dispozici. V rámci C++ API lze využívat:

- `BrickButton` - tlačítka
- `statusLight` - stavová kontrolka (dvoubarevná LED)
- `display` - LCD displej



Inicializace

Pro využití stavové kontrolky a displej není potřeba inicializace.

Inicializace se provádí jen pro tlačítka:

```
ev3cxx::BrickButton btnEnter(ev3cxx::BrickButtons::ENTER);
```

Vytvořili jsme objekt `btnEnter`, který reprezentuje prostřední tlačítka na EV3 Bricku (`ENTER`).

BrickButton

EV3 Brick má šest tlačítek, které lze využít v uživatelském programu, pro různé nastavování a řízení průběhu programu.

Seznam tlačítek na EV3 Bricku:

- `LEFT` - tlačítko doleva
- `RIGHT` - tlačítko doprava
- `UP` - tlačítko nahoru
- `DOWN` - tlačítko dolů
- `ENTER` - tlačítko uprostřed
- `BACK` - tlačítko zpět

Metody dostupné ve třídě `BrickButton`:

- `isPressed()` - vrací `true` pokud je tlačítko stisknuté
- `waitForPress()` - čekání, dokud se tlačítko nestiskne
- `waitForRelease()` - čekání, dokud se tlačítko neuvolní
- `waitForClick()` - čekání na stisknutí a uvolnění tlačítka

isPressed()

```
int isPressed();
```

Vrací `true` v případě, že je tlačítko stisknuté, jinak `false`.

Příklad: `btnEnter.isPressed();`

waitForPress()


```
void waitForPress();
```

Program je pozastaven, dokud nebude tlačítko stisknuté.

Příklad: `btnEnter.waitForPress();`

waitForRelease()


```
void waitForRelease();
```

Program je pozastaven, dokud nebude tlačítko uvolněno.

Varování: Nezapomínejte, že v běžném stavu může být tlačítko uvolněno. Volání této metody program pozastaví pouze pokud je v daný okamžik tlačítko stisknuté.

waitForClick()


```
void waitForClick();
```

Program je pozastaven, dokud neproběhne stisknutí a uvolnění tlačítka.

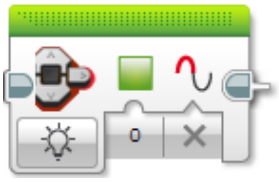
statusLight

Stavová kontrolka slouží běžně k indikaci stavu EV3 Bricku. Systém si sám nastavuje její stav. V době spuštění programu je možné tento stav měnit. Lze se přepínat mezi červenou, zelenou a oranžovou barvou. Pro prací s kontrolkou je k dispozici funkce `setColor()`, která nastavuje stav kontrolky.

- `OFF` - vypnuto

- `RED` - červená barva
- `GREEN` - zelená barvu
- `ORANGE` - oranžová barva

setColor()



```
void setColor(StatusLightColor color);
```

Nastavuje barvu stavové kontrolky. Lze se přepínat mezi červeno, zelenou a oranžovou barvou.

Příklad: `ev3cxx::statusLight.setColor(ev3cxx::StatusLightColor::GREEN);`

display

Objekt `display` slouží k vypisování textu na displej bricku. Pro výpis se dají použít dvě velikosti písma: `SMALL` a `MEDIUM`. Na displej se vejde 8 řádků psaných fontem `MEDIUM`, nebo 13 řádků fontem `SMALL`.

format()

```
format(char const *pattern);
format(int8_t line, char const *pattern);
```

pro výpis textu na displej slouží metoda `format`. Parametr `pattern` je text, který se má vypsat. Volitelný parametr `line` říká, na který řádek displeje se má text vypsat. Text se vypíše vždy od začátku displeje a vybraný řádek je před výpisem smazán.

Příklad: `ev3cxx::display.format("Hello world");`

Vypíše na displej text “Hello world”.

Ve vypisovaném textu se mohou vyskytovat speciální znaky pro ovládání pozice kurzoru:

- `\n` - přesune kurzor na začátek dalšího řádku
- `\r` - přesune kurzor na začátek tohoto řádku

Příklad: `ev3cxx::display.format("Ahoj svete! \nJak se mas?");`

Na prvním řádku displeje bude po provedení tohoto příkazu text “Ahoj svete!”. Na druhém řádku pak “Jak se mas?”.

`format` také umožňuje vypisovat na displej hodnotu proměnných pomocí znaku `%`. Lze vypisovat proměnné a řetězce.

Varování: Pozor: znak `%` pro výpis proměnné vždy "sežere" znak, následující bezprostředně za ním! Z toho důvodu jsou v prvním příkladu s hodinami za znakem `%` napsány dvě mezery. První mezeru "sežere" `%`, druhá se vypíše. Mít za `%` mezeru je nezbytně nutné i v případě, že je `%` posledním znakem v řetězci. Pokud za `%` následuje hned konec řetězce, je chování nedefinované.

Příklad: `ev3cxx::display.format("%") % 1;` číslo "1" se na displeji nezobrazí => za % není mezera `ev3cxx::display.format("% ") % 1;` při tomto zápisu formátovacího řetězce se již číslo "1" zobrazí normálně na displej

Příklad:

```
int hours = 19;
int minutes = 42;
ev3cxx::display.format("Je %  hodin a %  minut.") % hours % minutes;
```

Vypíše na displej text "Je 19 hodin a 42 minut.".

Také se dá specifikovat zarovnání výpisu proměnné doprava na daný počet znaků. Například při výpisu hodin je dobré, když jsou hodiny i minuty vždy na stejném místě, bez ohledu na to, jestli jsou zrovna reprezentovány jednomístným, nebo dvoumístným číslem.

Poznámka: Zarovnání znaků je podporováno jen u celočíselných typů (`int`, `long`, ...). Nelze jej nastavovat u čísel s plovoucí čárkou (`float` a `double`). U nich bude uživatelem zadané zarovnání ignorováno a použije se vždy výchozí nastavení (`float` => `%g` a `double` => `%f` dle standardní specifikace formátování čísel v C).

Příklad:

```
int hours = 8;
int minutes = 42;
ev3cxx::display.format("Je %2 hodin a %2 minut.") % hours % minutes;
```

Vypíše na displej text `Je 8 hodin a 42 minut.`. Všimněte si dvou mezer mezi "je" a "8", ale jen jedné mezery mezi "a" a "42". 8 hodin je pouze jednomístné číslo a tudíž ho formát sám doplnil zleva mezerou, aby zabíralo stejně místa, jako dvoumístné číslo a nedocházelo k posunu následujícího textu. Zarovnání doplňuje zleva mezery, pokud je to potřeba, ale nebrání ve výpisu delších čísel.

Dále je možné vypisovat čísla v dvojkové, nebo šestnáctkové soustavě.

Příklad: `ev3cxx::display.format("hex: %x4\nbin: %b8\ndec: %3") % 42 % 42 % 42;`

Vypíše na displej text:

`hex: 002A bin: 00101010 dec: 42`

Šestnáctková a dvojková čísla se při zarovnání doplňují číslicí 0, zatím co desítková mezerami.

Chcete-li mít ve výsledném textu znak %, použijte kombinaci `%%`:

Příklad: `ev3cxx::display.format("10%%");`

Vypíše na displej text "10%".

resetScreen()

```
void resetScreen(bool_t color = white);
```

Tato metoda smaže celý displej. Podle parametru `color` může být poté celý displej buď bílý, nebo černý. Bez parametru je výchozí stav bílý displej.

Příklad: `ev3cxx::display.resetScreen();`

setFont()

```
void setFont(lcdfont_t font);
```

Tato metoda umožňuje nastavit font, kterým se bude na displej psát. Jsou k dispozici dva fonty:

- EV3_FONT_SMALL - malý font, 13 řádků na displej
- EV3_FONT_MEDIUM - střední font, 8 řádků na displej

Příklad: `ev3cxx::display.setFont(EV3_FONT_MEDIUM);`

KAPITOLA 5

Čas

Pro práci s časem je v EV3CXX k dispozici:

- `wait()` - čekání v milisekundách
- `StopWatch` - třída pro práci se stopkami, umožňuje měření a časování

Čekání

Pro čekání v programu (pozastavení vykonávání) je v EV3CXX k dispozici funkce `wait()`.

void wait()

```
void wait(unsigned int time_ms);
```

Funkce pozastaví vykonávání programu na zadaný počet milisekund.

Poznámka: Nezapomeňte u funkce uvést namespace `ev3cxx`.

Příklad čekání jednu sekundu: `ev3cxx::wait(1000);`

Měření a časování

Pro odměřování času v EV3CXX slouží třída `StopWatch`.

```
StopWatch stopky;
```

Poznámka: Po vytvoření objektu `StopWatch` se automaticky spouští stopky/měření. Pokud si chcete odstartovat měření sami až v průběhu programu, stáčí předat při vytváření objektu hodnotu `false`.

Příklad: `StopWatch stopky(false);`

---

isRunning()

```
bool isRunning();
```

Vrátí `true` pokud časovač běží.

Příklad: `stopky.isRunning();`

reset()

```
void reset(bool start = true);
```

Resetuje stopky. Parametr `start` určuje, zda se stopky hned po restartu rozběhnou. Výchozí hodnota je `true`.

Příklad: `stopky.reset();`

getMs()

```
time_type getMs();
```

Vrací čas v milisekundách.

Příklad: `stopky.getMs();`

getUs()

```
time_type getUs();
```

Vrací čas v mikrosekundách.

Příklad: `stopky.getUs();`

KAPITOLA 6

Úvod

01 - trojúhelník

Prvním tvým úkolem bude naučit robota jezdit. Abychom nejezdili jen tak, zkusíme robota naučit jezdit ve tvaru trojúhelník.

Pro rozpohybování robota musíš použít motor-tank-class.

Je potřeba poskládat příkazy pro jízdu rovně a otočení, tak aby výsledný tvar, který robot projede tvořil trojúhelník.

Aby jsi mohl použít motory v režimu tank je potřeba si vytvořit objekt ze třídy `MotorTank`.

To lze provést následovně:

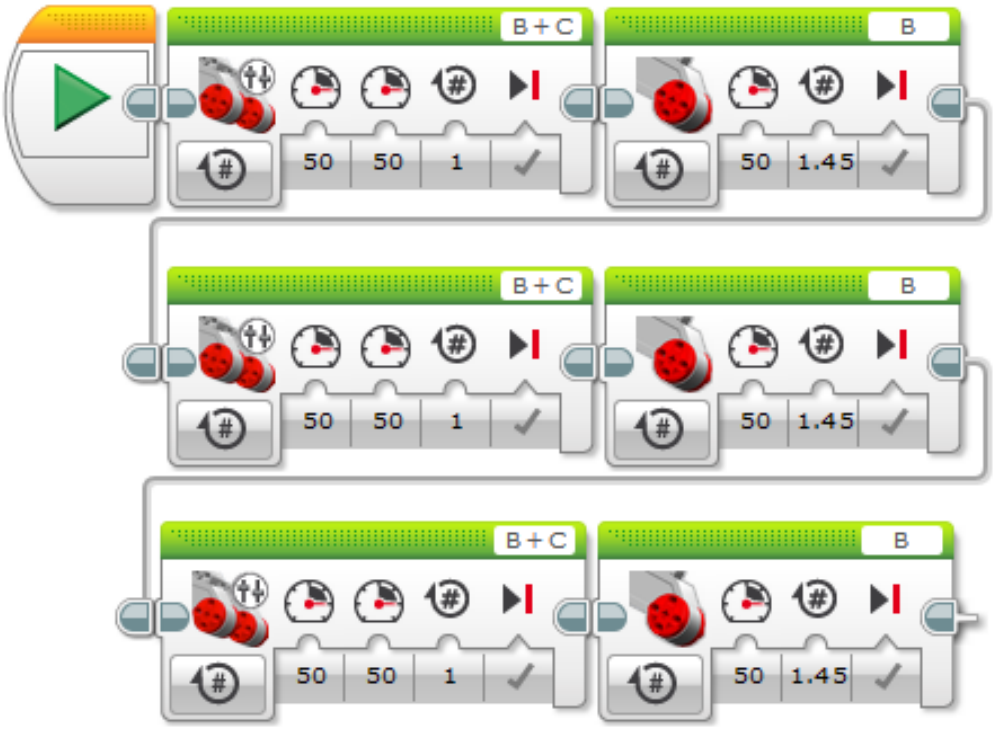
```
ev3cxx::MotorTank motors(ev3cxx::MotorPort::B, ev3cxx::MotorPort::C);
```

Pro jízdu rovně zavolej funkci `motors.onForRotations(50, 50, 1);`. Funkce je volána se stejnou rychlostí levého a pravého motoru (hodnota 50) a proto pojede robot rovně. Vzdálenost kterou ujede je 1 otáčka hřídele motoru a u našeho robota i 1 otáčka kola.

Když se budeš chtít otočit, použiješ stejnou funkci, ale nastavíš jiné parametry. S robotem se chceš otočit na místě, tak aby jedno kolo stálo a druhé jelo. Tím pádem musí být rychlost motoru na jedné straně 0 a na druhé straně třeba 50. Poslední parametr, který musíme upravit je počet otáček. Je potřeba najít optimální číslo pro otáčku, tak abychom měli pěkný trojúhelník. To již ale nechám na tobě. Vyzkoušej si různá čísla a vyber to nejvhodnější. Jen nezapomeň, že můžeš zadávat i desetinná čísla (s desetinnou tečkou: *1.5*).

Pokud máš tyto dva příkazy napsané, stačí už je jen dvakrát rozkopírovat a odzkoušet. Jestli nebudeš mít úplně přesný trojúhelník, nic se neděje. Zkus si ještě pohrát s počtem otáček.

A to je v tomto úkolu vše. Pro inspiraci se můžeš podívat jak by program vypadal v LEGO Softwaru a níže najdeš ukázkovou verzi programu.

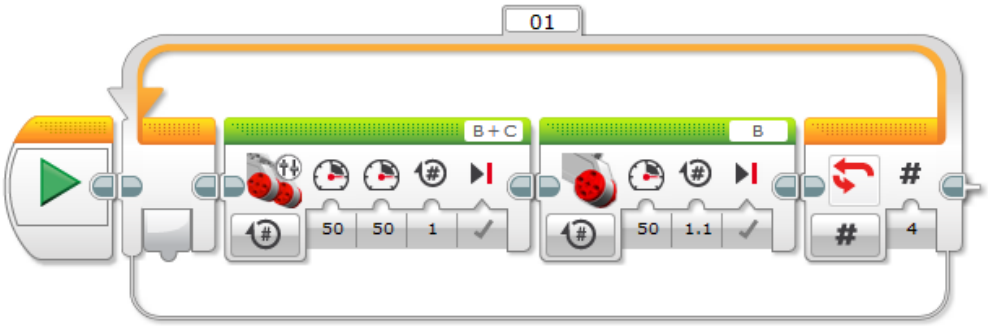



```
/**
 * This is sample program for drive a simple two-wheel robot along a triangle.
 *
 * Author: Jaroslav Páral (jarekparal)
 */

#include "ev3cxx.h"
#include "app.h"

void main_task(intptr_t unused) {
    ev3cxx::MotorTank motors(ev3cxx::MotorPort::B, ev3cxx::MotorPort::C);

    motors.onForRotations(50, 50, 1);
    motors.onForRotations(50,  0, 1.45);

    motors.onForRotations(50, 50, 1);
    motors.onForRotations(50,  0, 1.45);

    motors.onForRotations(50, 50, 1);
    motors.onForRotations(50,  0, 1.45);
}
```

02 - čtverec

Prvním tvým úkolem bylo naučit robota jezdit ve tvaru trojúhelníků. Teď si zkusíme udělat podobný úkol se čtvercem. Říkáš, že je to skoro to samí? Ano, to máš pravdu. My si ale tentokrát práci trochu usnadníme.

Místo kopírování kódu použijeme cyklus. Cyklus za nás provede opakování kódu sám, bez toho abychom jej museli kamkoliv kopírovat.

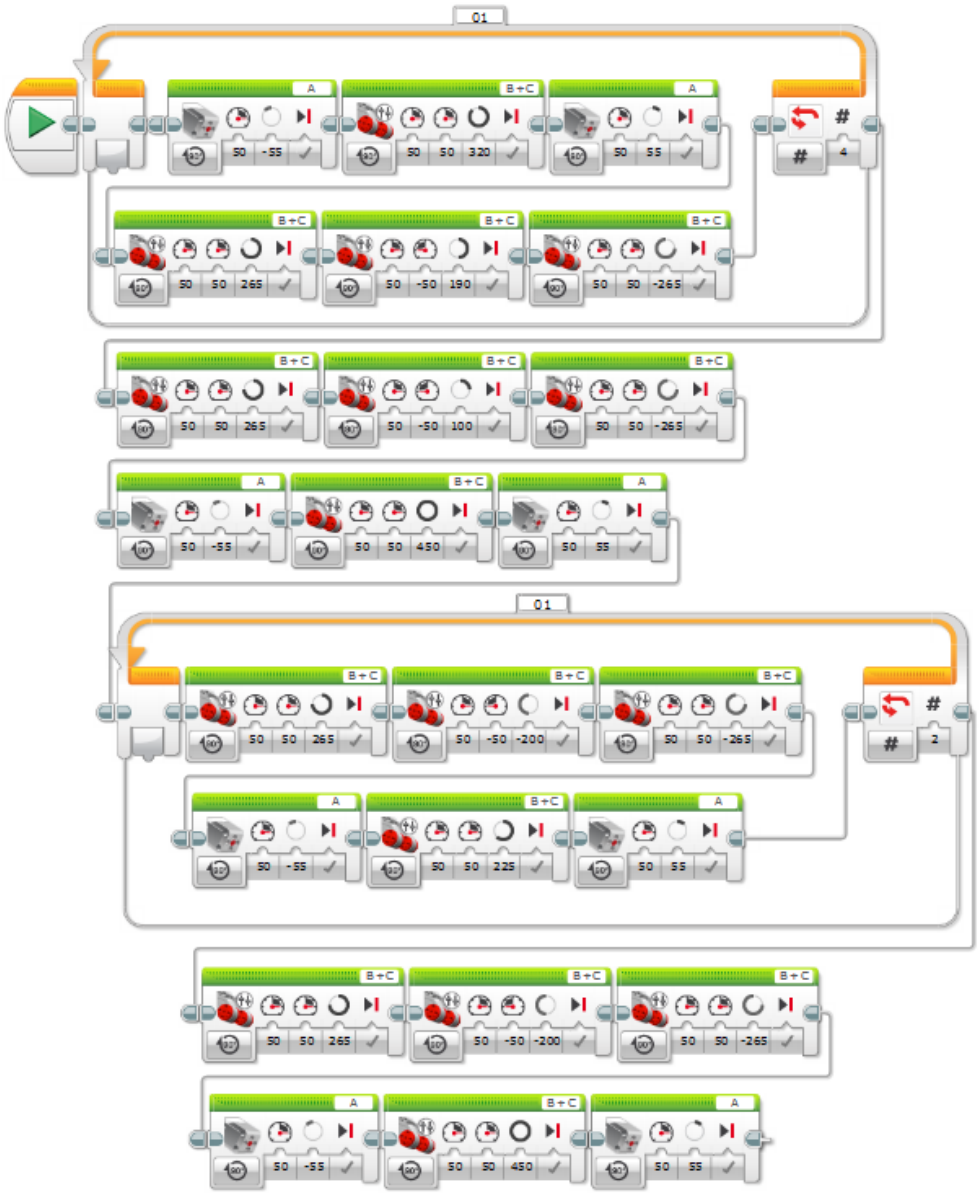



```
/**
 * This is sample program for drive a simple two-wheel robot along a square.
 *
 * Author: Jaroslav Páral (jarekparal)
 */

#include "ev3cxx.h"
#include "app.h"

void main_task(intptr_t unused) {
    ev3cxx::MotorTank motors(ev3cxx::MotorPort::B, ev3cxx::MotorPort::C);

    for (int i = 0; i != 4; ++i) {
        motors.onForRotations(50, 50, 1);
        motors.leftMotor().onForRotations(50, 1.1);
    }
}
```

03 - malujeme dům

Ve třetím úkolu nebudeme jen jezdit, ale zkusíme i kreslit. Takže papír, tužku, izolepu a jdeme kreslit. Zkusíme si robotem nakreslit dům. Pro kreslení je asi nejvhodnější fix a k robotovy ji můžeme přidělat pomocí izolepy na přední packu. Pamatuj na to, že střed robota není střed fixe. S tímto musí váš program počítat.

Tento program je již složitější a rozsáhlejší, tak se zkus inspirovat třeba diagramy z LEGO Softwaru.

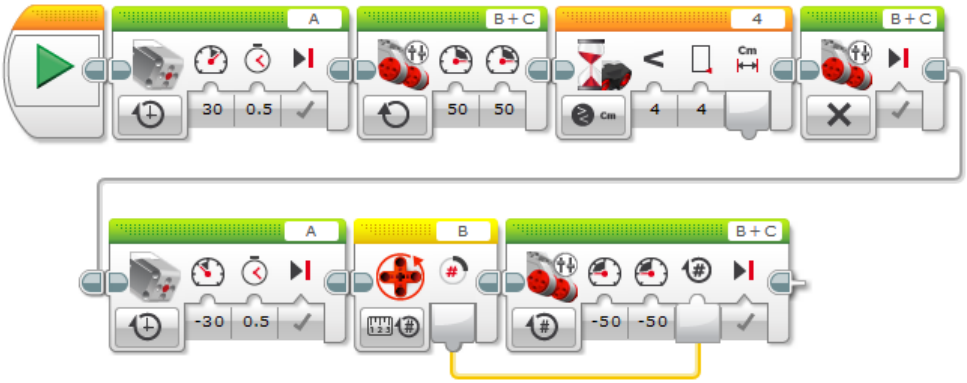


```
/**
 * This is sample program for draw a house by a simple two-wheel robot, holding a pen
 *
 * Don't forget set position of pen_motor UP!
 *
 * Author: Jaroslav Páral (jarekparal)
 */

#include "ev3cxx.h"
#include "app.h"

void main_task(intptr_t unused) {
    ev3cxx::MotorTank motors(ev3cxx::MotorPort::B, ev3cxx::MotorPort::C);
    ev3cxx::Motor pen_motor(ev3cxx::MotorPort::A);

    motors.onForRotations(50, 50, 1);
```

```
    for (int i = 0; i != 4; ++i) {
        pen_motor.onForDegrees(50, -55);
        motors.onForDegrees(50, 50, 320);
        pen_motor.onForDegrees(50, 55);
        motors.onForDegrees(50,  50,  265);
        motors.onForDegrees(50, -50,  190);
        motors.onForDegrees(50,  50, -265);
    }
    motors.onForDegrees(50,  50,  265);
    motors.onForDegrees(50, -50,  100);
    motors.onForDegrees(50,  50, -265);
    pen_motor.onForDegrees(50, -55);
    motors.onForDegrees(50, 50, 450);
    pen_motor.onForDegrees(50, 55);
    for (int i = 0; i != 2; ++i) {
        motors.onForDegrees(50,  50,  265);
        motors.onForDegrees(50, -50, -200);
        motors.onForDegrees(50,  50, -265);
        pen_motor.onForDegrees(50, -55);
        motors.onForDegrees(50, 50, 225);
        pen_motor.onForDegrees(50, 55);
    }
    motors.onForDegrees(50,  50,  265);
    motors.onForDegrees(50, -50, -200);
    motors.onForDegrees(50,  50, -265);
    pen_motor.onForDegrees(50, -55);
    motors.onForDegrees(50, 50, 450);
    pen_motor.onForDegrees(50, 55);
}
```

04 - chytit a vrátit

Ve čtvrtém úkolu zkusíme poprvé použít senzor. Cílem je dojet pro barevnou kostku, kterou jsi si měl složit s Education robotem. Chytit ji pomocí robotické ruky a vrátit se na místo, odkud jsi vyjel. Tento úkol není tak složitý, tak nad ním zkus zapřemýšlet. Potřebuješ použít ultrazvuk, který máš na svém robotovi a vše ostatní již znáš.

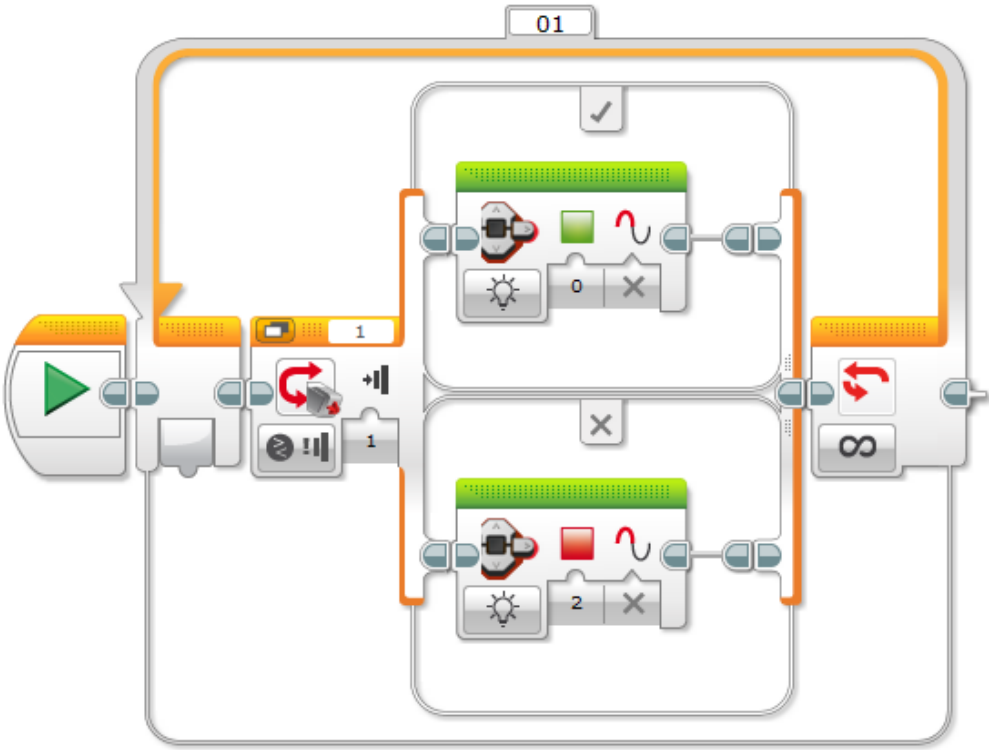



```
/**
 * Go straight until an obstacle is detected by ultrasonic sensor, capture it and␣
↪return to the start position
 *
 * Author: Jaroslav Páral (jarekparal)
```

```
 */

#include "ev3cxx.h"
#include "app.h"

void main_task(intptr_t unused) {
    ev3cxx::UltrasonicSensor ultrasonic(ev3cxx::SensorPort::S4);
    ev3cxx::MotorTank motors(ev3cxx::MotorPort::B, ev3cxx::MotorPort::C);
    ev3cxx::Motor claw_motor(ev3cxx::MotorPort::A);

    claw_motor.onForSeconds(30, 500);
    motors.on(50, 50);

    while (ultrasonic.centimeters() >= 4) {};

    motors.off(true);
    claw_motor.onForSeconds(-30, 500);
    motors.onForDegrees(-50, -50, motors.leftMotor().degrees());
}
```

05 - zmáčkni a sviť

V rámci pátého úkolu si vyzkoušíme pracovat s LED na Bricku a budeme je ovládat pomocí Touch senzoru. Také si poprvé vyzkoušíme poprvé použít podmínku. Již jsme s ní pracovali v rámci cyklu, takže by to neměl být žádný problém.



```
/**
 * Lights green LED if TouchSensor is pressed, otherwise lights red LED.
 *
 * Author: Jaroslav Páral (jarekparal)
 */

#include "ev3cxx.h"
#include "app.h"

void main_task(intptr_t unused) {
    ev3cxx::TouchSensor touch(ev3cxx::SensorPort::S1);

    while (true) {
        if (touch.isPressed())
            ev3cxx::statusLight.setColor(ev3cxx::StatusLightColor::GREEN);
        else
            ev3cxx::statusLight.setColor(ev3cxx::StatusLightColor::RED);
    }
}
```

Vítejte v Robotutoriálu. V následujících několika lekcích tě provedu základními úlohami, které tě naučí pracovat s LEGO MINDSTORMS EV3.

V rámci tutoriálu budeme pracovat s robotem Educator. Návod na jeho složení najdeš na těchto stránkách.

Při skládání postupuj pečlivě a zkontroluj, že jsi všechny motory a senzory zapojil do správných portů. V našem tutoriálu nebudeme potřebovat Gyro senzor a nemusíš jej tedy sestavovat. Barevný senzor nastav tak, aby jsi měl snímačem k zemi (návod strana 52). Nezapomeňte složit i barevnou kostku, budeme ji potřebovat.

Jsi připraven začít? Tak pojďme na to.

Obr. 6.1: Robot Educator

KAPITOLA 7

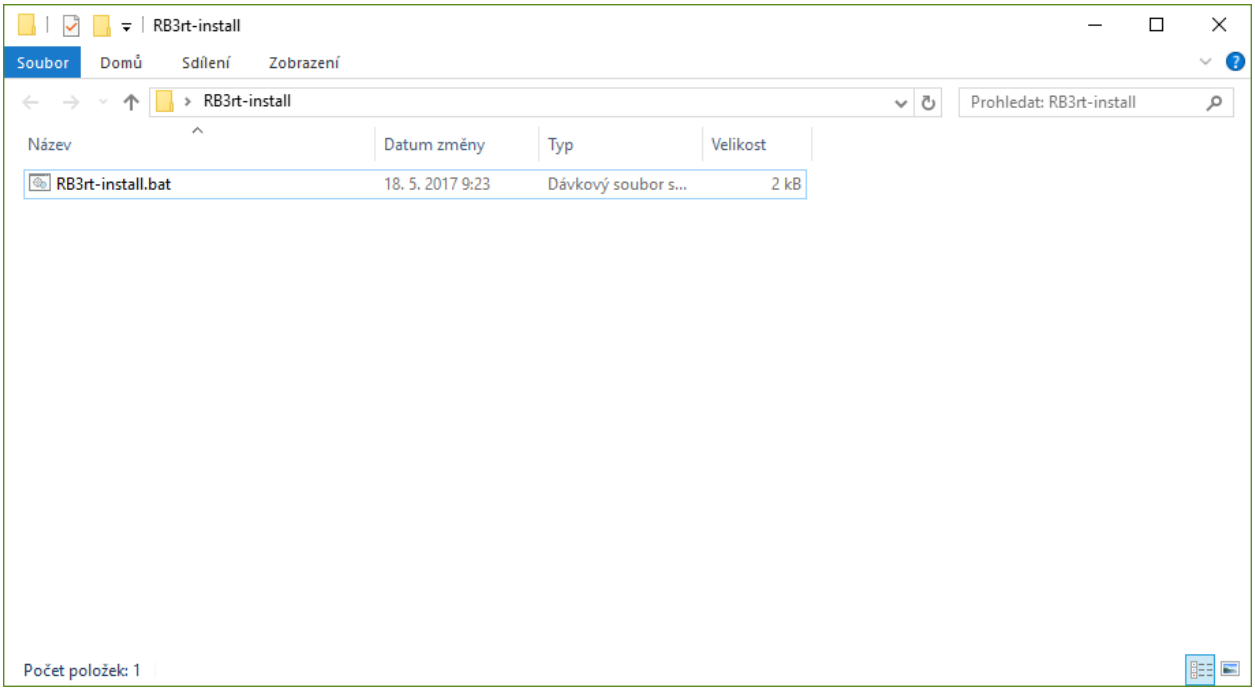
Instalace vývojového prostředí

Pro jednodušší používání C++ knihoven je k dispozici předpřipravené vývojového prostředí.

K instalaci prostředí je připraven skript RB3rt-install.bat.

Veškerý software se nainstaluje na diskový oddíl `C:`.

Po stažení skriptu postupujte podle obrázkového návodu.

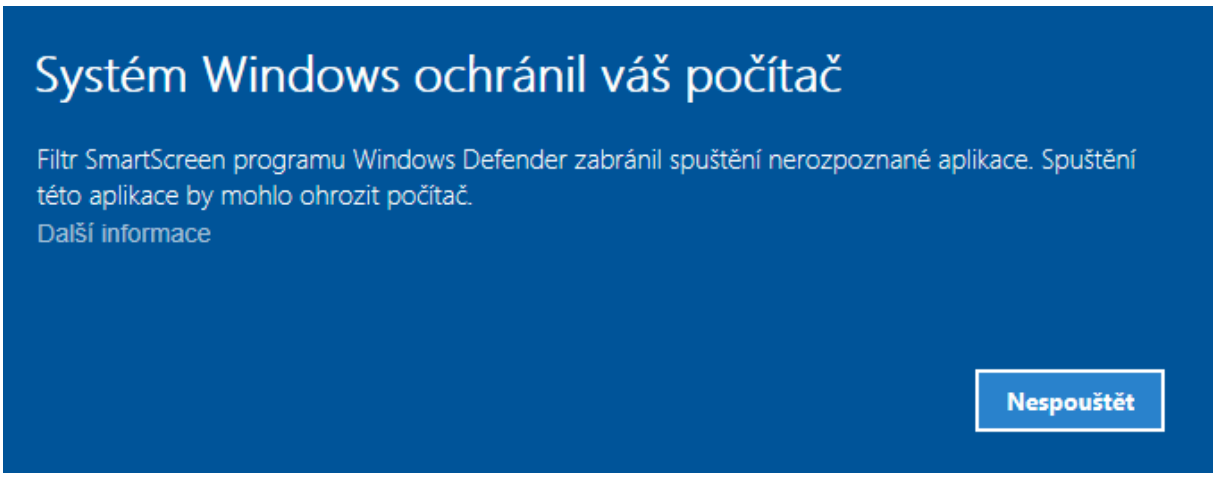


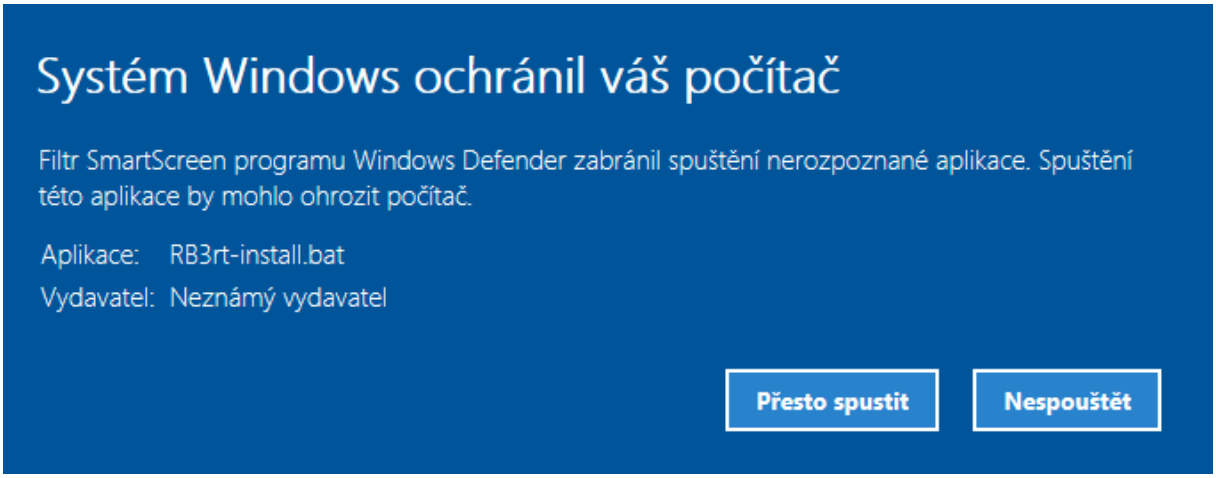
Obr. 7.1: Nejprve je potřeba spustit instalační skript.

Varování: Na novějších verzích Windows (8,10) se zobrazí okno s informací o nerozpoznané aplikaci. Jelikož Windows tento skript nezná, z bezpečnostních důvodů zobrazuje toto okno.

Pro pokračování instalace je potřeba kliknout na `Další informace`

Následně spustit instalaci tlačítkem `Přesto spustit`.

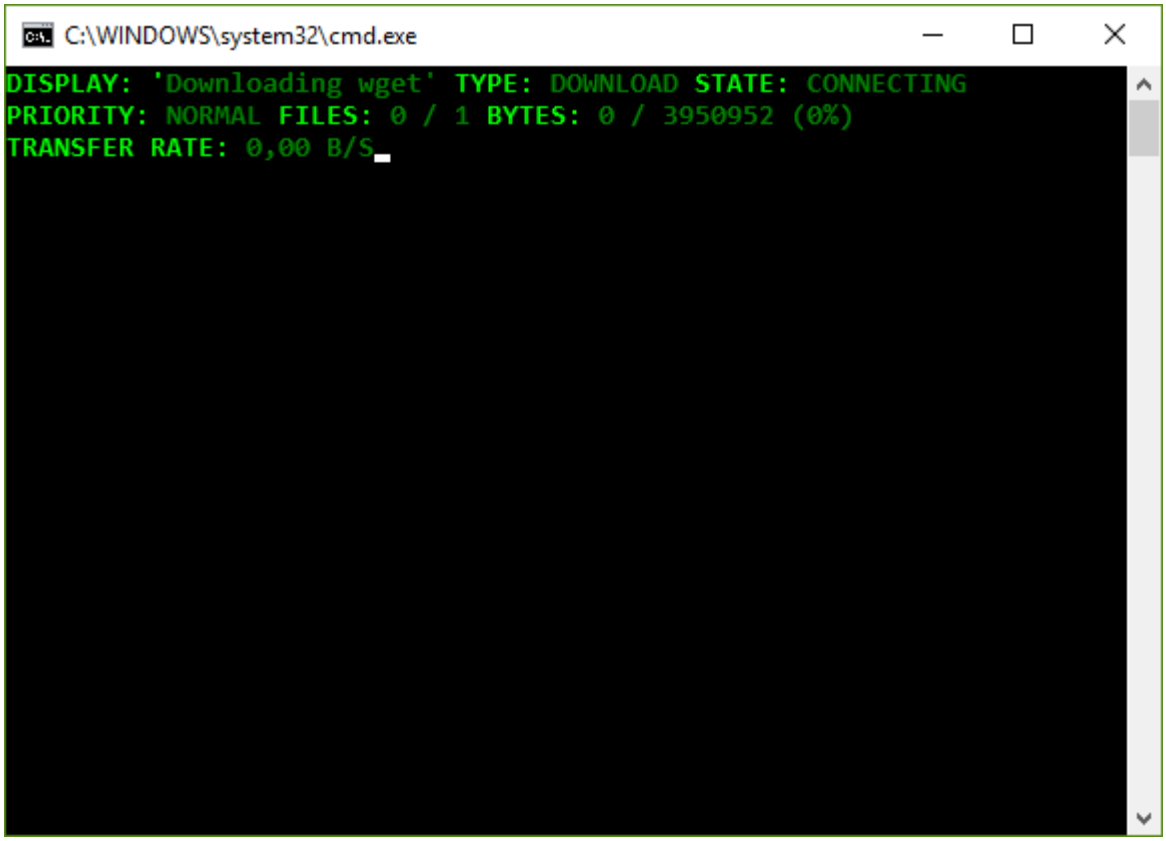




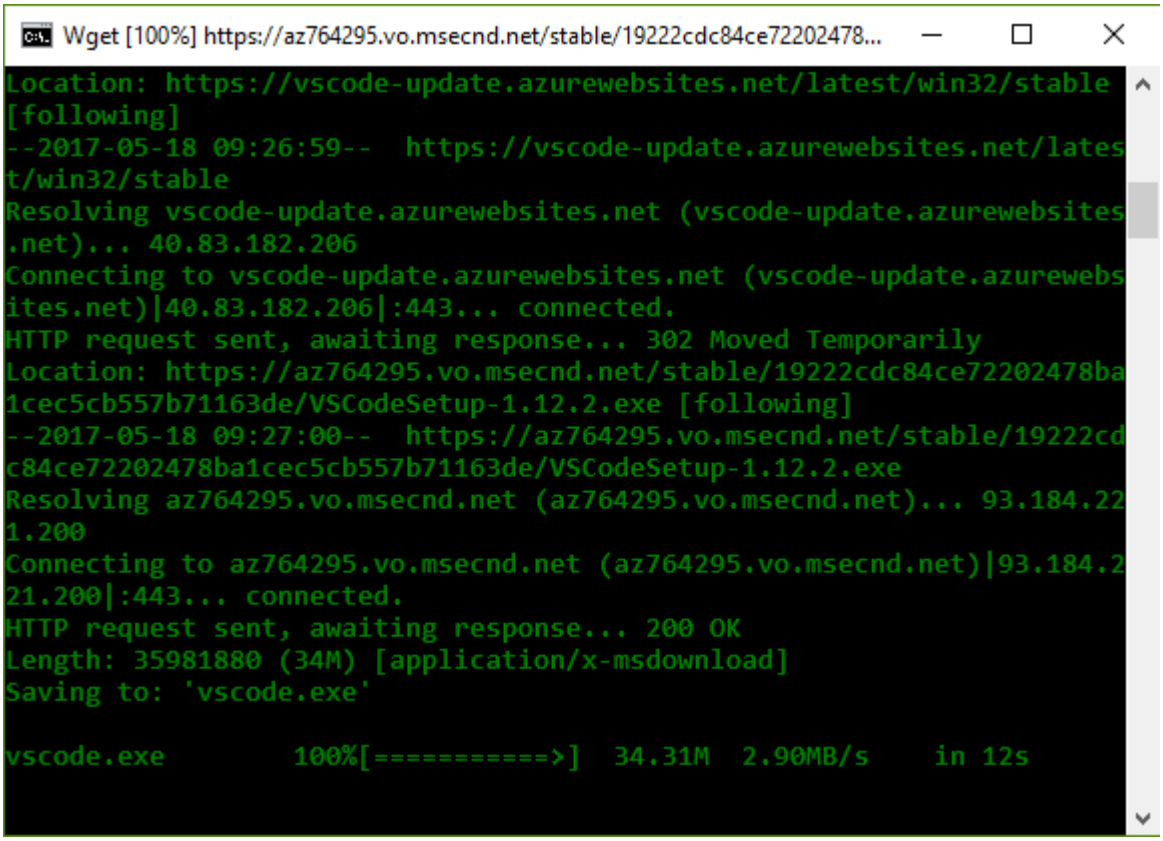
Varování: Nepostupujte na další krok, dokud se okno s instalací Cygwinu nezavře.

Visual Studio Code (VS Code)

Poznámka: Po jeho otevření pravděpodobně proběhne ještě doinstalovaní některých doplňků.

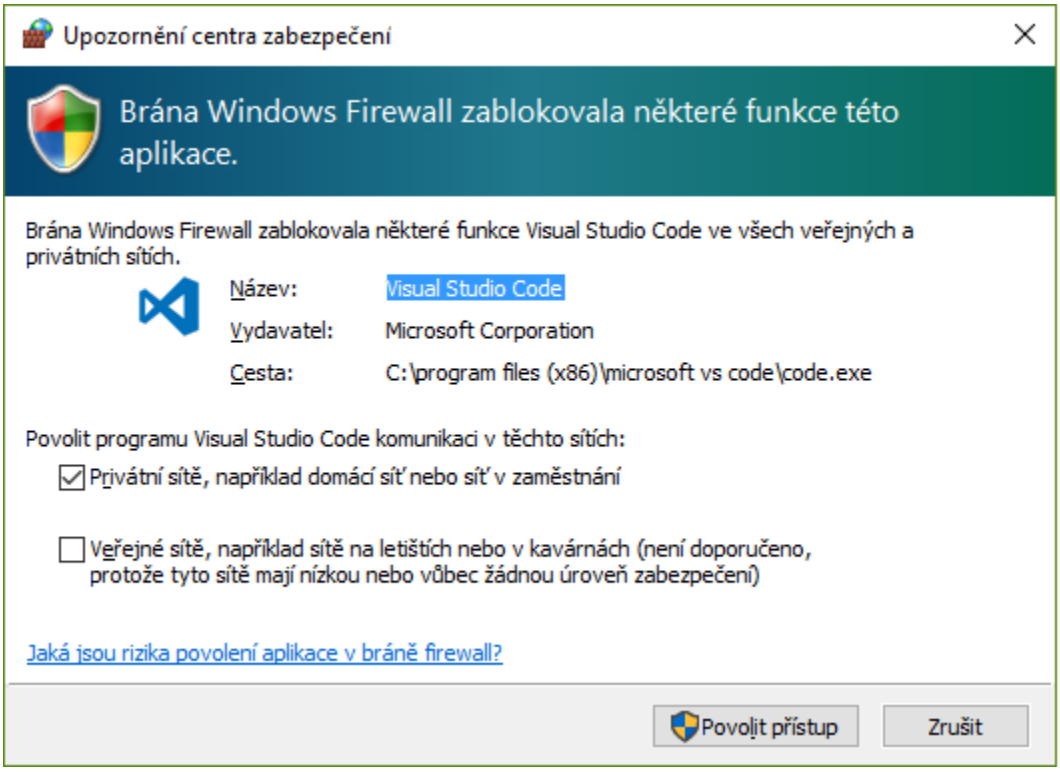


Obr. 7.2: Po spuštění začne skript stahovat potřebný software. V průběhu instalace je nutné potvrzovat práva k instalaci u jednotlivých softwaru.



Obr. 7.3: Skript vždy stáhne požadovaný software a spustí jeho instalaci. Po dokončení instalace pokračuje na další software.

Takto se nainstaluje: 7-Zip, Visual Studio Code, Cygwin, GCC ARM překladač



Obr. 7.4: Po nainstalování Visual Studio Code se program otevře a bude potřeba potvrdit přístup na internet.

Nastavení klávesových zkratek

Přeložení programu

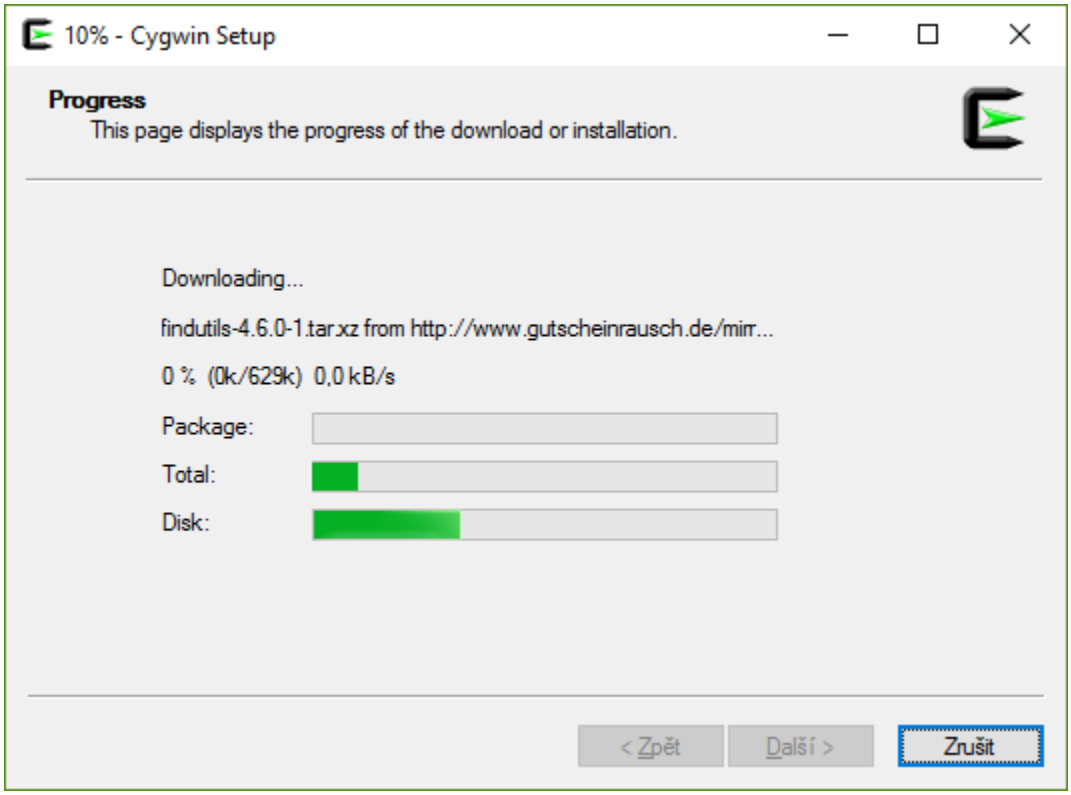
Poznámka: Adresář s ukázkovým programem můžete přemístit kamkoliv na vašem PC. Jeho pozice nemusí být fixní. Všechny ostatní adresáře, které se při instalaci prostředí vytvořili, již ale musí zůstat na stejném místě.

Systém EV3RT

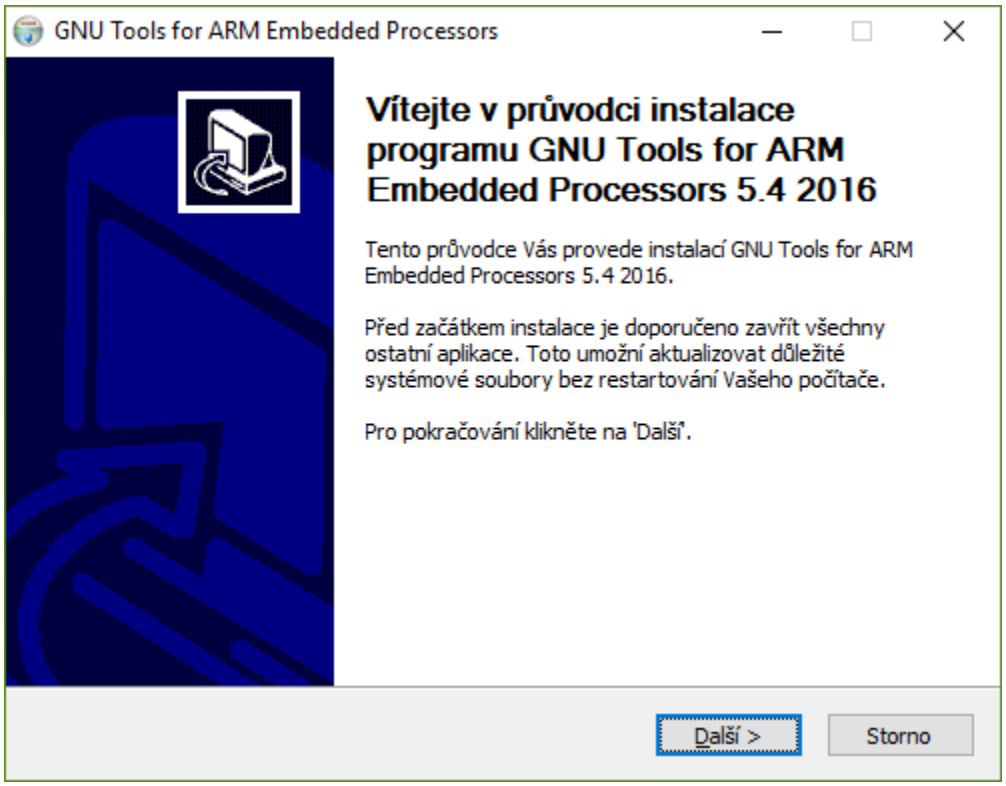
Pro spuštění systému EV3RT na LEGO MINDSTORMS EV3 je potřeba nahrát image systému na micro SDHC kartu.

Image systému po proběhnutí instalačního skriptu, popisovaného v úvodu této kapitoly, k dispozici ve složce `C:\RB3rt-image`. Obsah této složku je potřeba překopírovat na SD kartu a následně ji vložit do EV3 Bricku. Pak již stačí jen spustit Brick.

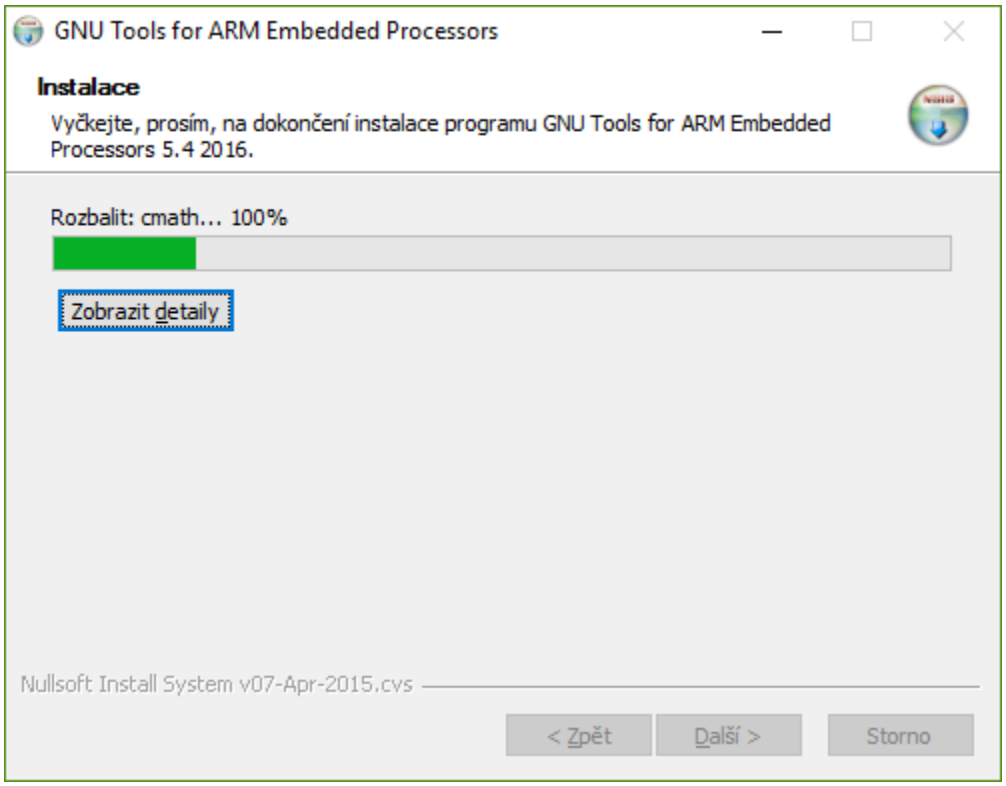
Varování: Systém EV3RT podporuje jen SDHC karty. Neumí pracovat se staršími SD kartami (do 2 GB). Je proto potřeba mít k dispozici kartu alespoň o velikosti 4 GB.



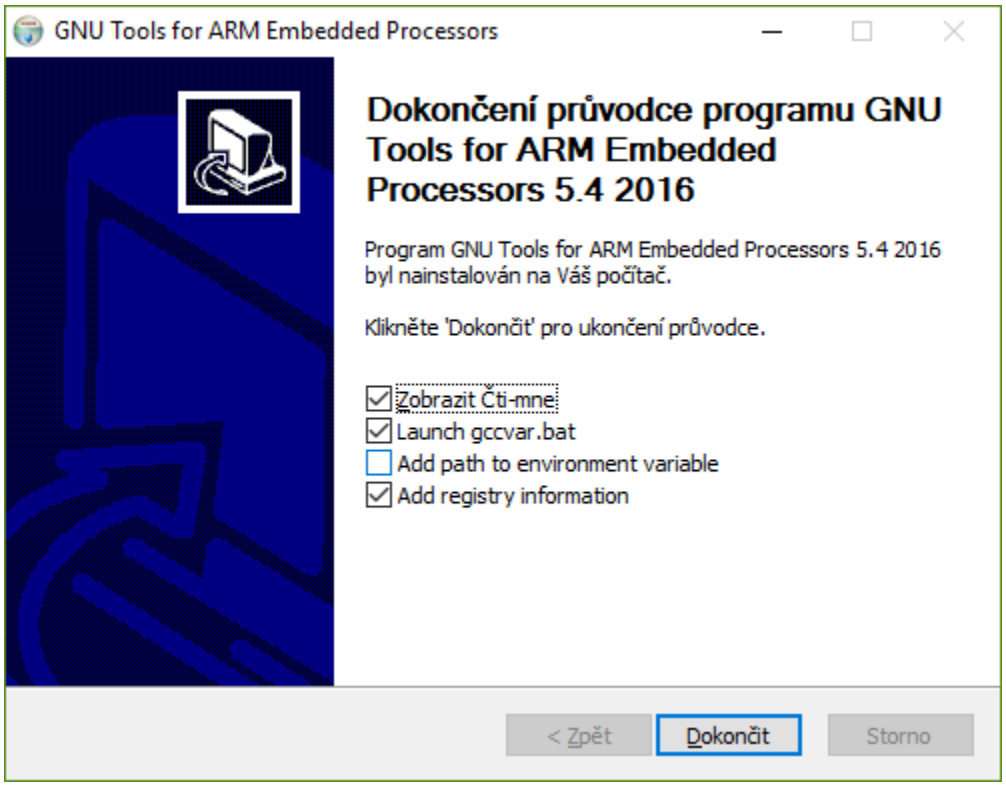
Obr. 7.5: Další okno, které se vám zobrazí bude instalace software Cygwin. Toto okno je jen informativní a není potřeba s ním cokoliv dělat.



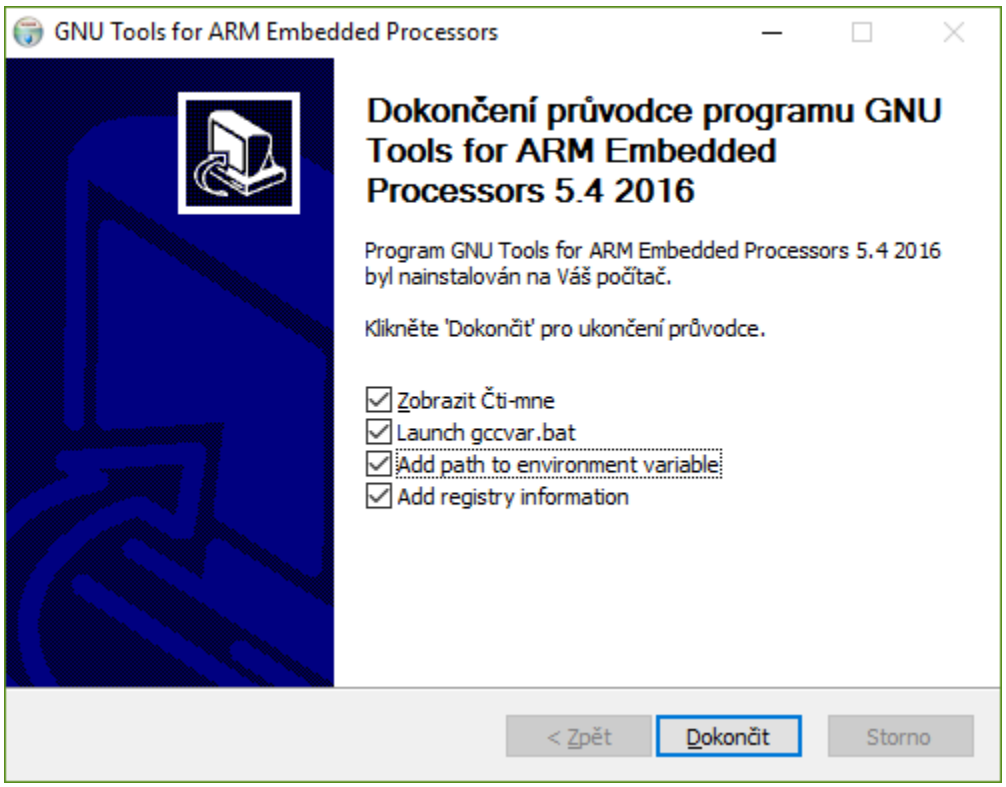
Obr. 7.6: V následujícím kroku se otevře instalace GCC ARM překladače (GNU Tools for ARM ...). Zde je potřeba projít instalačním procesem. Postupně volte vždy krok `Další` a ponechejte výchozí nastavení.



Obr. 7.7: Následně se rozběhne instalace.



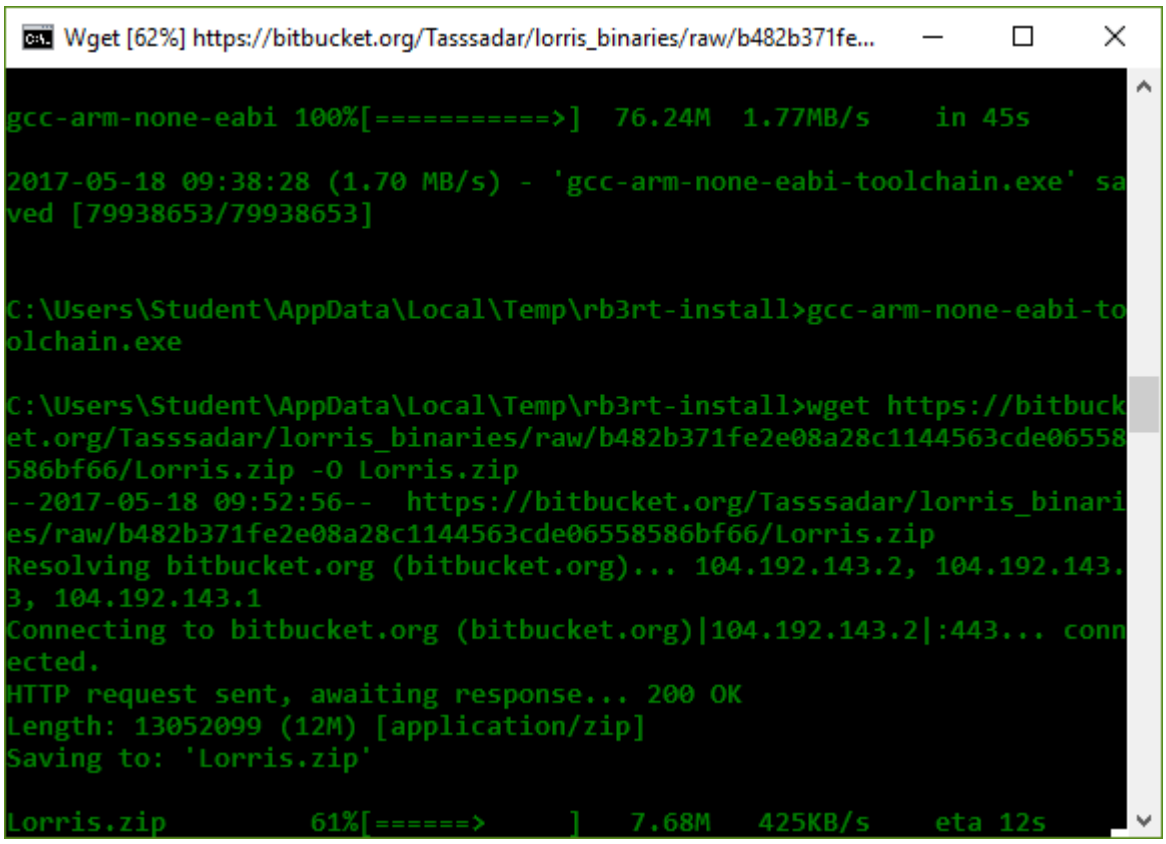
Obr. 7.8: Po dokončení instalace je potřeba vybrat zatržítko `Add path to environment variable`.



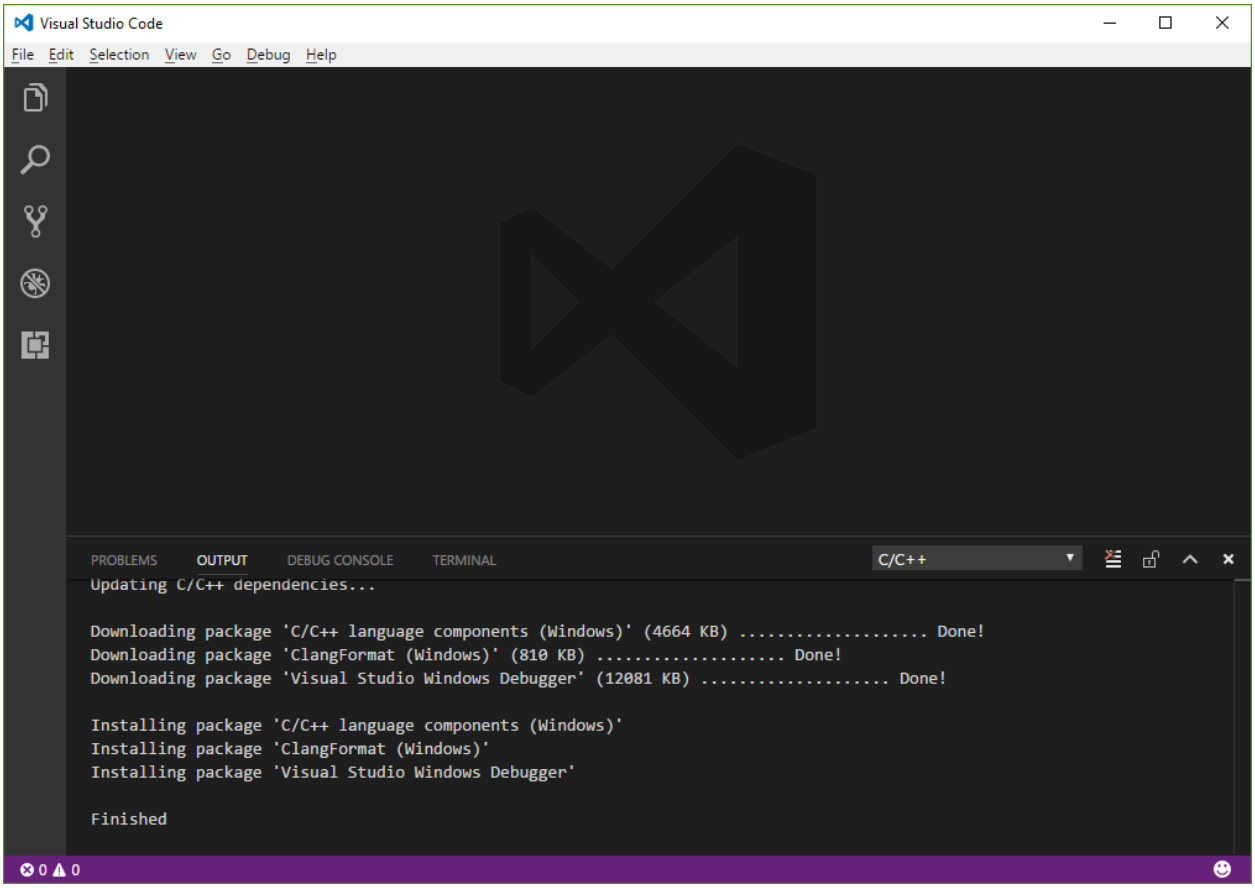
Obr. 7.9: Po vybrání zatržítka `Add path to environment variable`, klikněte na `Dokončit`.

Nahrání programu do EV3RT

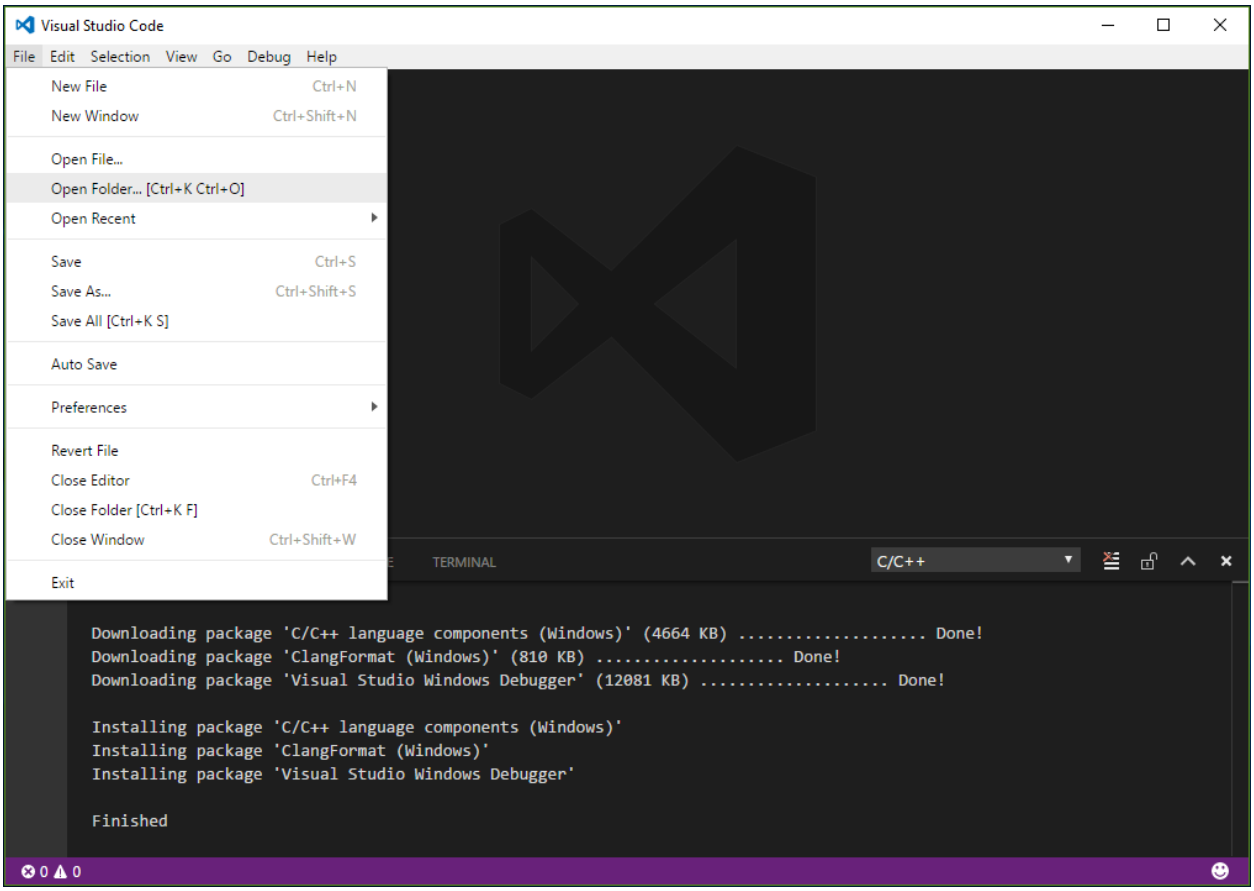
Nahrání programu je velmi jednoduché. Systém EV3RT se při připojení Bricku k PC chová jako standardní Flash disk. Stačí tedy vzít přeložený program (soubor `app`) ze složky s vaším projektem a vložit jej na SD kartu do adresáře `ev3rt\apps\`. Tento adresář je již v image systému vytvořen a obsahuje ukázkový projekt `helloev3`. Projekty na kartě si můžete přejmenovávat jak chcete. Názvy souborů a složek ale nesmí obsahovat diakritiku (háčky, čárky), mezery a nebo speciální znaky ($%^&#@). Pro oddělování slov doporučuji použít pomlčku - nebo podtržítko _.



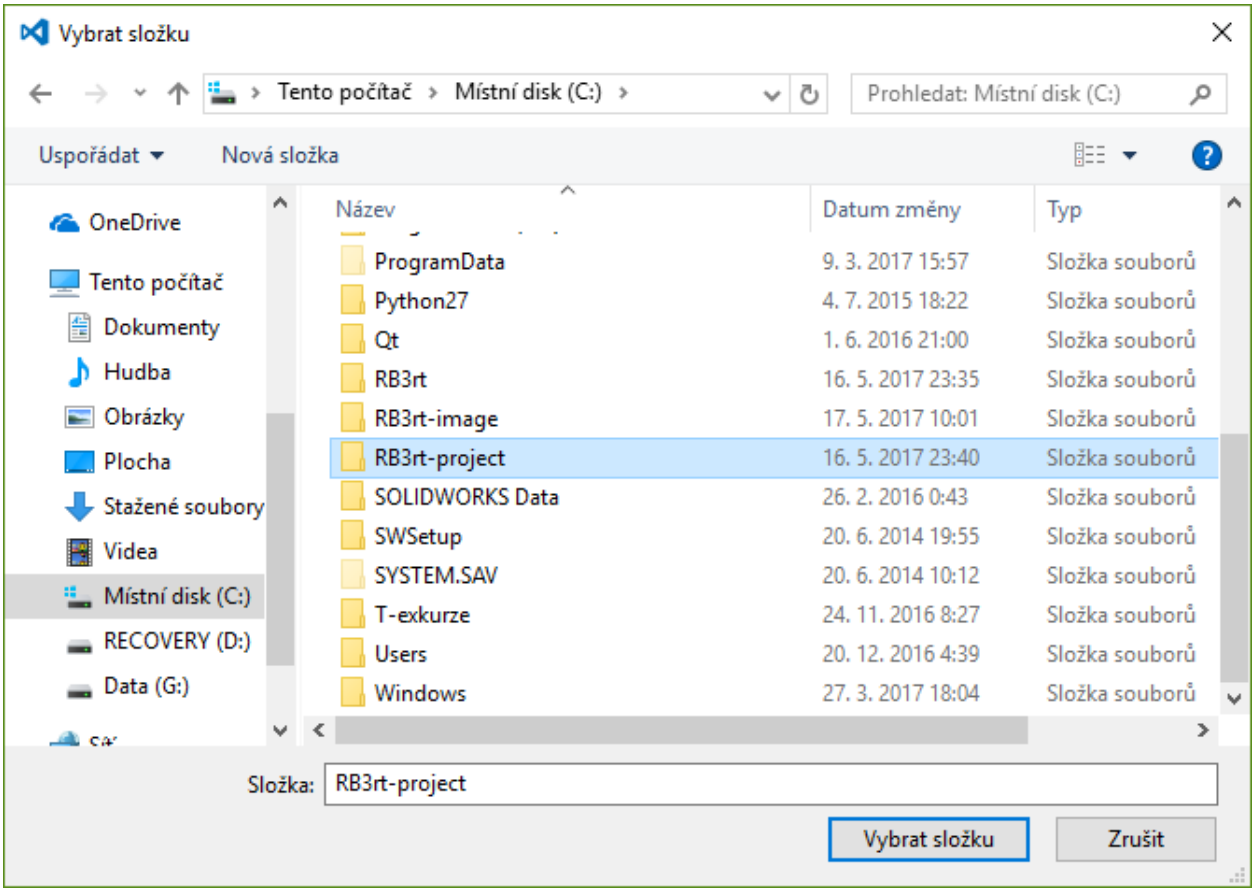
Obr. 7.10: Nyní již automaticky skript dostahuje všechny potřebné soubory a skončí.



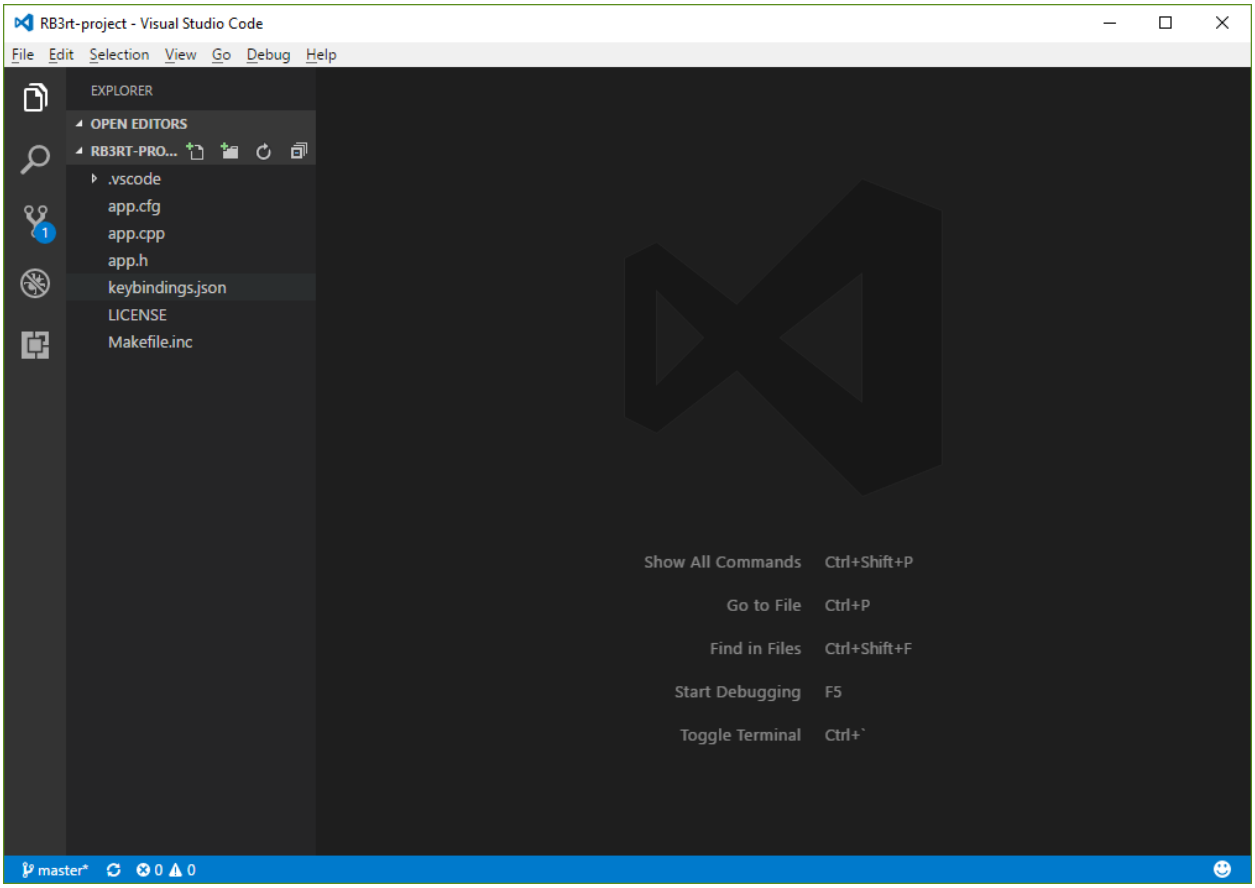
Obr. 7.11: Otevřete vývojový editor Visual Studio Code (najdete jej v nabídce `Start`).



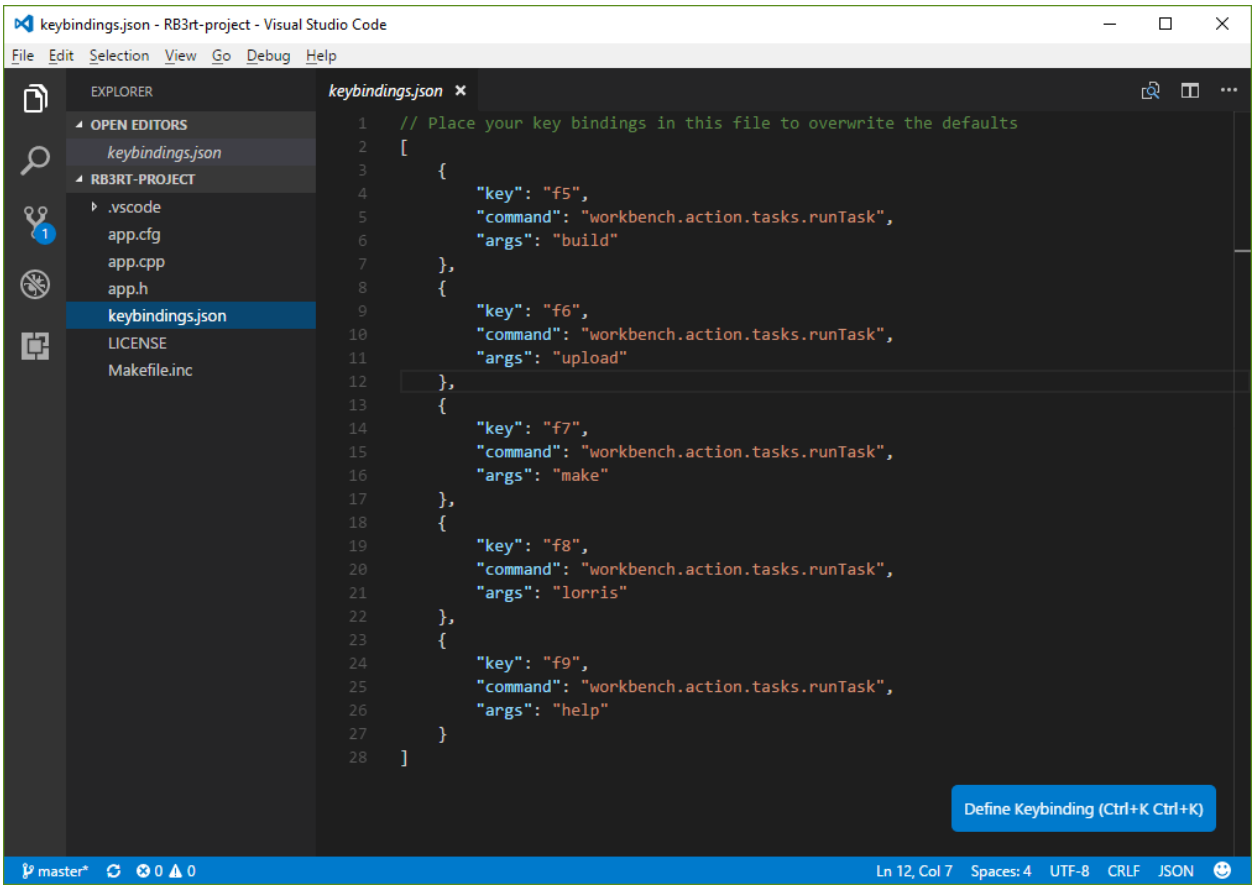
Obr. 7.12: Otevřete nabídku `File` a vyberte volbu `Open Folder...`.



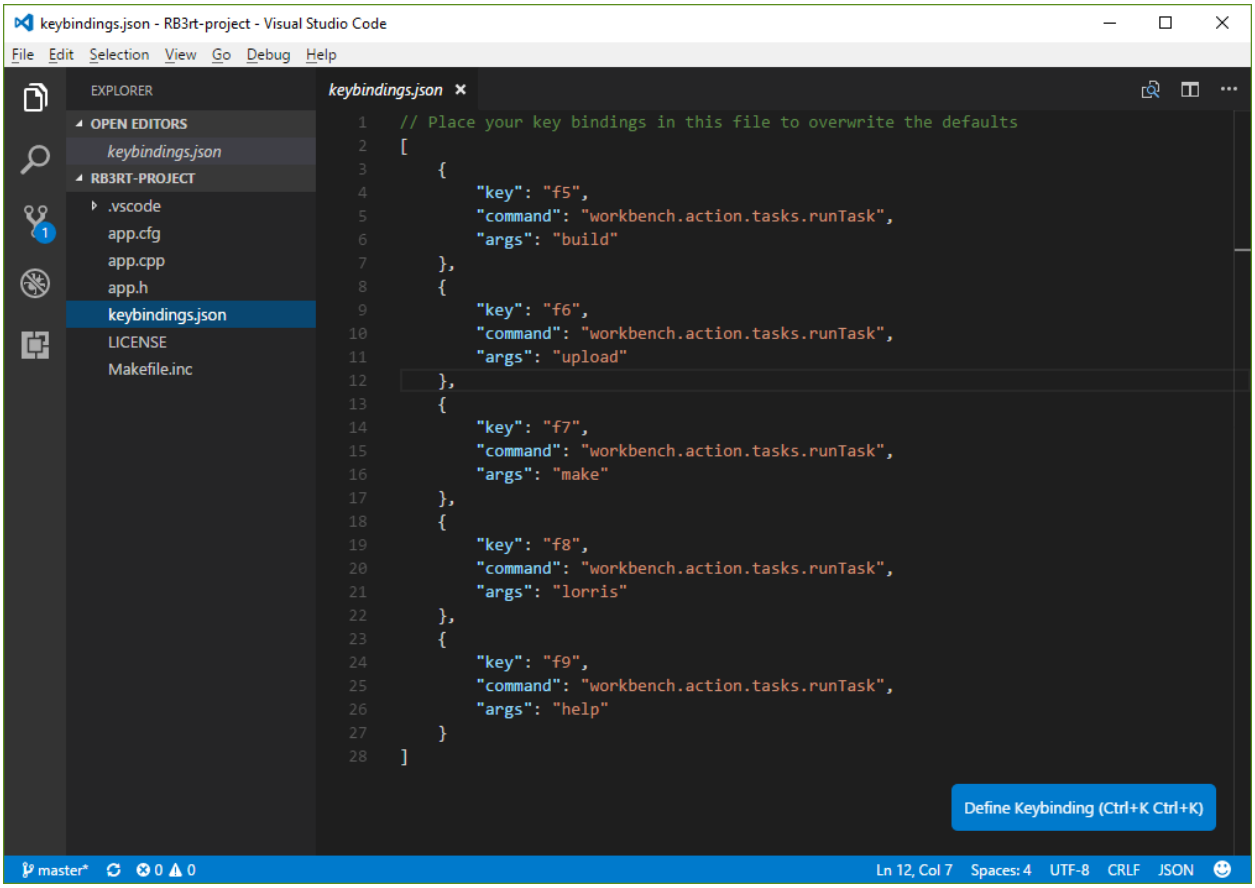
Obr. 7.13: V systémovém oddílu `C:` vyberte složku `RB3rt-project` a potvrďte `Vybrat složku`.



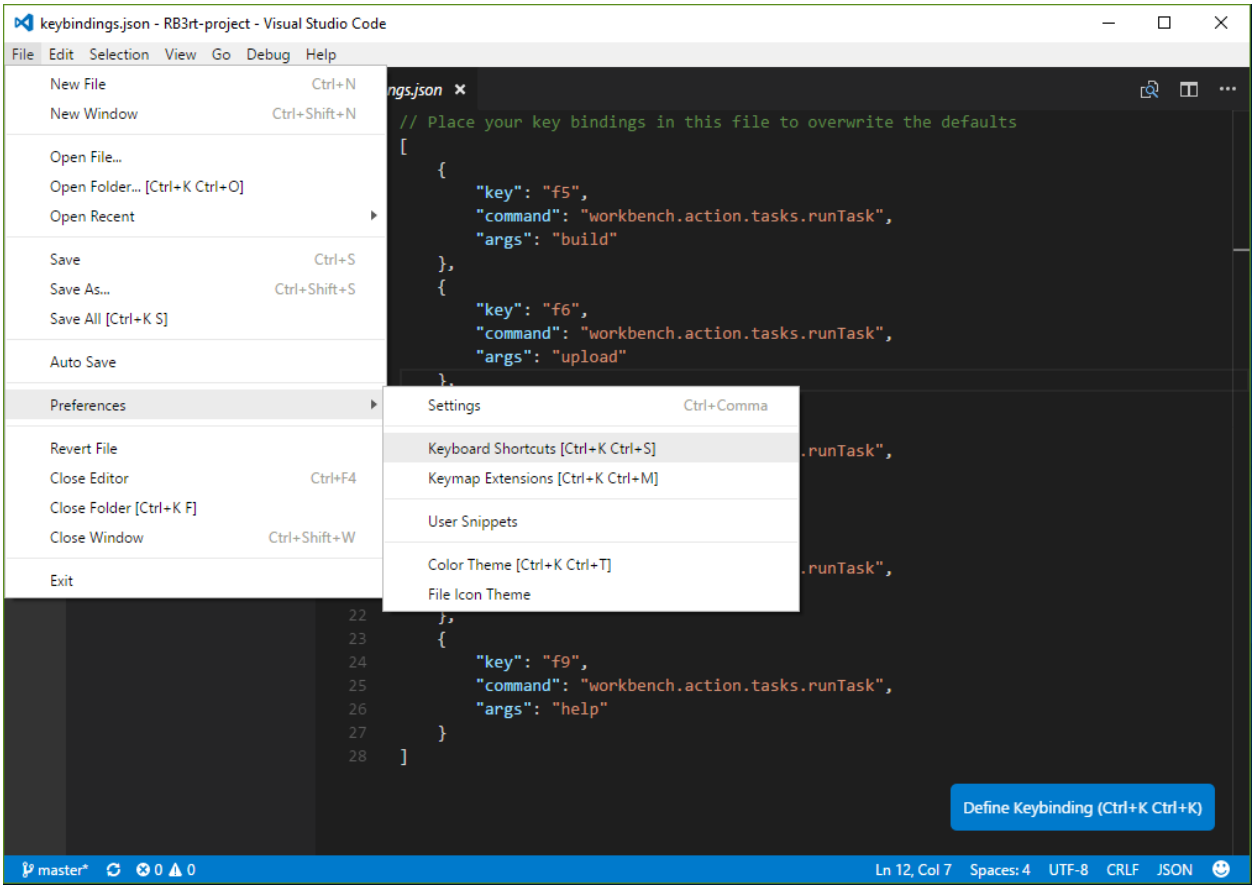
Obr. 7.14: Nyní se otevřel adresář s ukázkovým projektem.



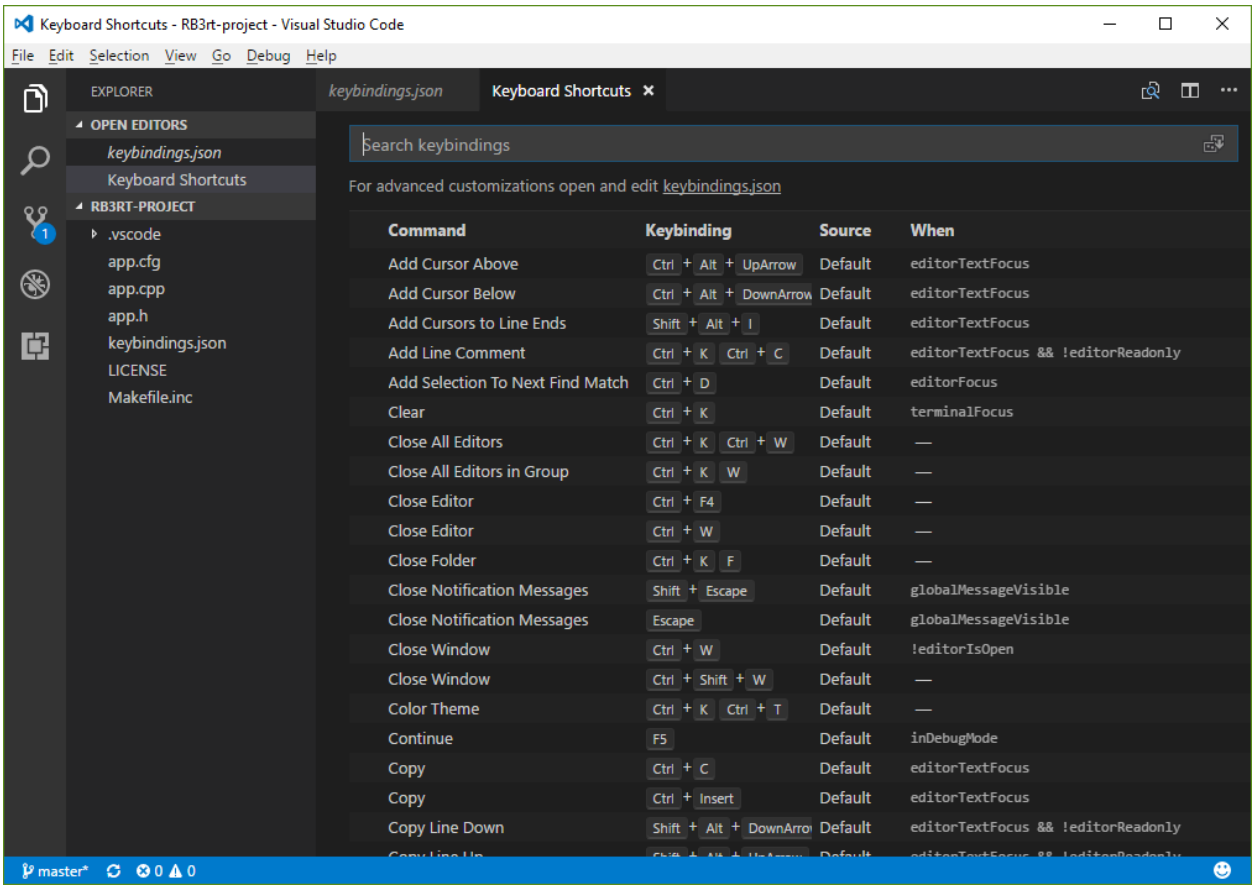
Obr. 7.15: V následujících krocích nastavíme klávesové zkratky. Otevřete soubor `keybindings.json` z levé nabídky (stačí kliknout).



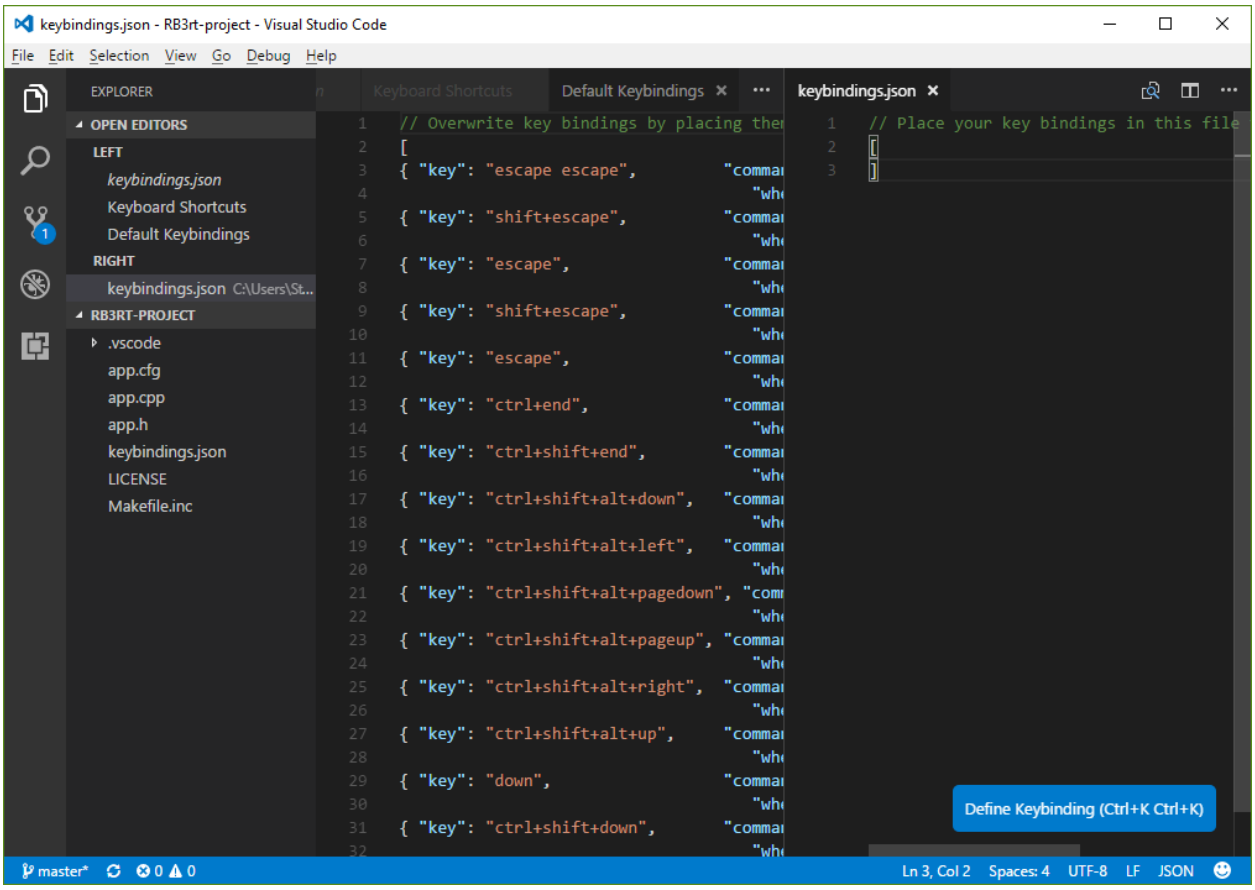
Obr. 7.16: Obsah tohoto souboru budeme za chvíli kopírovat do nastavení VS Code.



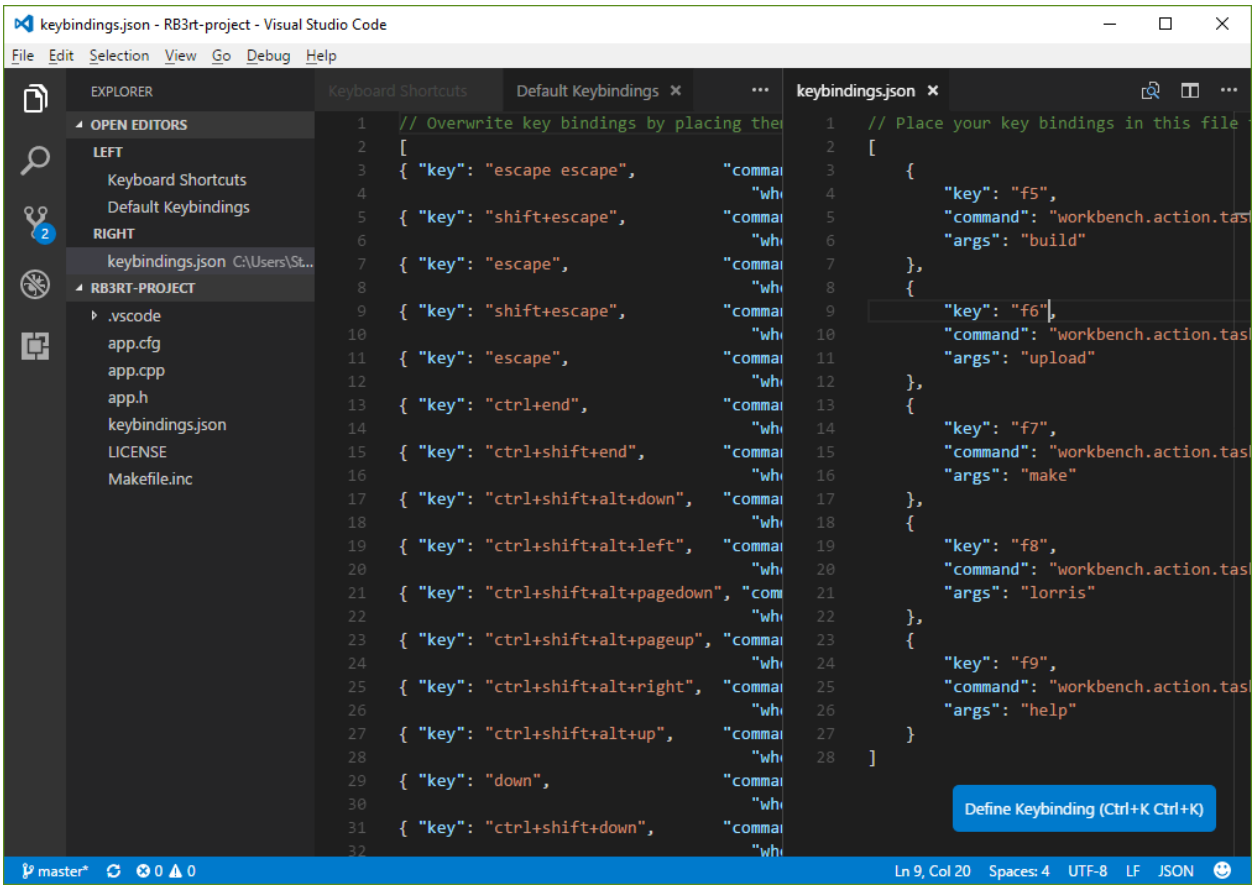
Obr. 7.17: Otevřete nadku `File` => `Preferences` => `Keyboard Shortcuts`.



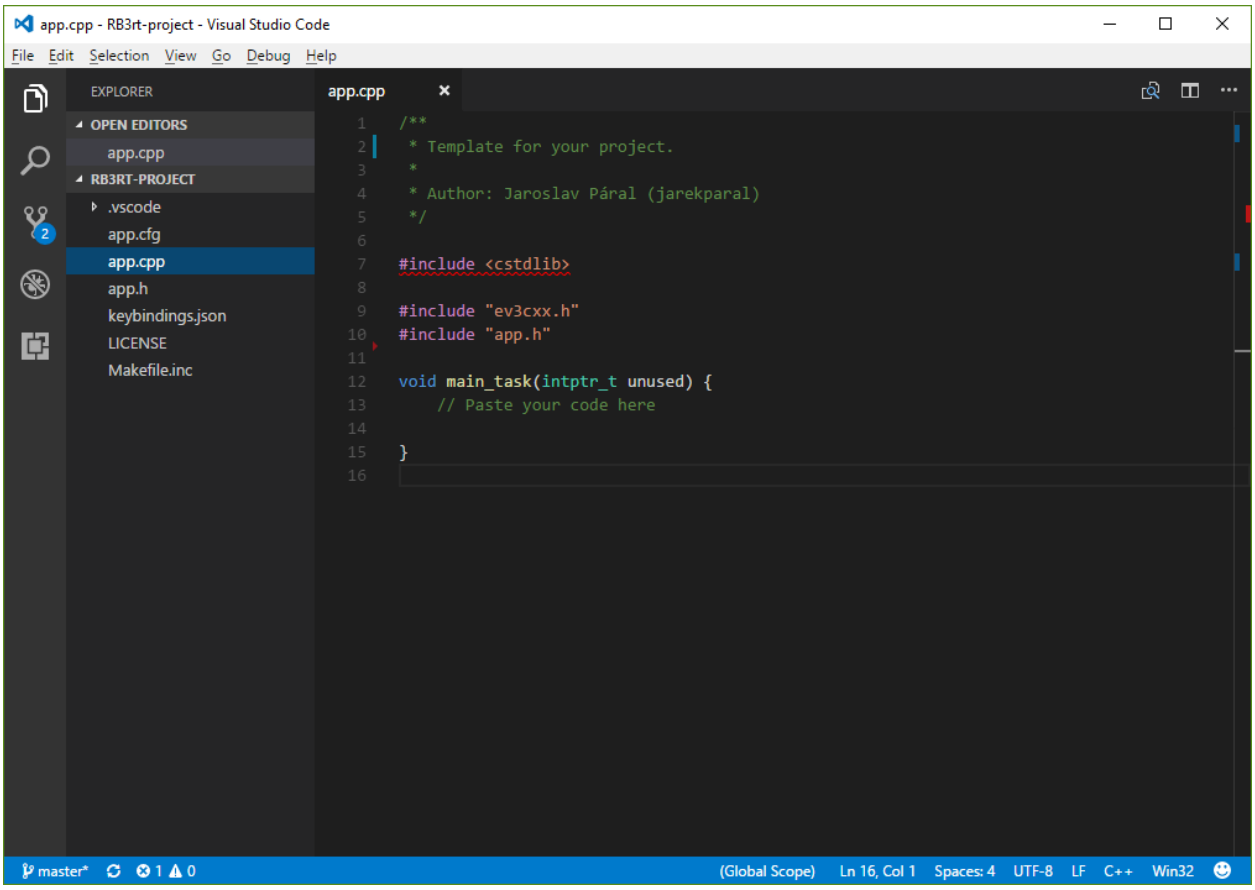
Obr. 7.18: Nyní vidíte všechny klávesové zkratky ve VS Code. My potřebujeme ale nastavit vlastní, a proto klikneme na odkaz `keybindings.json`, který najdete v okně s přehledem zkratek, hned pod vyhledávacím polem a ve větě s textem: For advanced customizations open add edit `keybindings.json`.



Obr. 7.19: V právem okně se otevře soubor `keybindings.json`. Do tohoto souboru je potřeba nakopírovat obsah souboru `keybindings.json`, který jsme otevírali v úvodu.



Obr. 7.20: Po překopírování nastavení již stačí vše uložit a restartovat VS Code. Pak již budou všechny klávesové zkratky fungovat.



Obr. 7.21: Pro přeložení programu, po předchozím nastavení klávesových zkratek, stačí otevřít soubor `app.cpp` v ukázkovém projektu a zmáčknout `F5`.

KAPITOLA 8

FAQ

Zda naleznete soupis chyb a jejich řešení, na které můžete při programování narazit.

Práce s projektem

Vytvoření nového projektu

Pokud chcete založit nový projekt, stačí jen zkopírovat složku s ukázkovým projektem `RB3rt-project` z `C:\RB3rt-project`, přejmenovat jej a umístit kamkoliv budete chtít.

Poznámka: Název složky nesmí obsahovat diakritiku (háčky, čárky), mezery a nebo speciální znaky ($%^&#@). Pro oddělování slov doporučuji použít pomlčku - nebo podtržítko _.

Chyby při překladu programu

Nadpisy podkapitol jsou buď celé nebo zkrácené chybové hlášky, které se mohou během překladu zobrazit.

error: expected unqualified-id before numeric constant

Chyba může nastat, když například špatně zadáte označení portu u senzoru:

```
ev3cxx::TouchSensor touchS(ev3cxx::SensorPort::1);
// spatne oznaceni portu senzory
```

Senzory v EV3CXX jsou označovány: `S1`, `S2`, `S3` a `S4`.

```
ev3cxx::TouchSensor touchS(ev3cxx::SensorPort::S1);
// takhle je to spravne
```

error: ambiguous overload for 'operator%' (operand types are 'ev3cxx::detail::format_impl

Chyba může nastat, když například předáte strukturu `colorid_t` do metody `format()` přes `%`.

Příklad:

```
colorid_t col = colorS.color();
display.format("Color: % \n") % col;
```

Metoda `format()` neumí pracovat se strukturou `colorid_t`. Pokud si chceme zobrazit danou barvu, musíme například použít podmínku:

```
colorid_t col = colorS.color();

if (col == COLOR_BLACK) {
   display.format("Black color\n");
}
```

error: expected '}' at end of input

S touto chybou se můžete setkat, když na konci zdrojového souboru *app.cpp* nenecháte prázdný řádek. Při překladu programů pro EV3RT je potřeba, aby za posledním znakem zdrojového kódu byl prázdný řádek.

Poznámka: Aktuální verze ukázkového projektu je tento problém řeší. Je přidáno nastavení projektu, které zajišťuje automatické přidání prázdného řádku na konec souboru při uložení. Konfigurační soubor pro VS Code si můžete do projektu přidat sami (složka *.vscode* a soubor *settings.json* - viz odkaz na řešení v úvodu poznámky) nebo si stáhněte a používejte aktuální verzi ukázkového projektu.

KAPITOLA 9

Autor

- Jaroslav Páral

Dotazy zasílejte na: paral@robotikabrno.cz

KAPITOLA 10

Indices and tables

- genindex
- modindex
- search