PŘEHLED FUNKCÍ BRICX

Jan Mareš

Rok: 2012

Obsah

1 Připravené programy **4**
- 1.1 Bricx . . . 4
- 1.2 CodeBlocx . . . 4
- 1.3 Dev c++ . . . 5

2 Stručný přehled **6**
- 2.1 Tabulka probraných funkcí . . . 6
- 2.2 Wait . . . 6
  - 2.2.1 Příklad Wait: . . . 6
- 2.3 TextOut . . . 7
  - 2.3.1 Příklad TextOut: . . . 7
- 2.4 OnFwd . . . 7
  - 2.4.1 Příklad OnFwd: . . . 7
- 2.5 OnRev . . . 8
- 2.6 Off . . . 8
  - 2.6.1 Příklad: pohyb motorů: . . . 8
- 2.7 OnFwdSync . . . 9
  - 2.7.1 Příklad OnFwdSync: . . . 9
- 2.8 RotateMotor . . . 9
  - 2.8.1 Příklad RotateMotor: . . . 9
- 2.9 RotateMotorEx . . . 10
  - 2.9.1 Příklad RotateMotorEx: . . . 10

3 Proměnné **11**
- 3.1 Co jsou proměnné? . . . 11
- 3.2 Počítání s proměnnými . . . 12
  - 3.2.1 Inicializace proměnných ** . . . 14
- 3.3 Vstup od uživatele * . . . 14
- 3.4 Typy proměnných * . . . 15
- 3.5 Zkrácený zápis přiřazení ** . . . 15
- 3.6 Pole ** . . . 15
- 3.7 Vícerozměrná pole *** . . . 15

4 Podmínky **16**
- 4.1 Možné výrazy v podmínce . . . 16
- 4.2 Kombinované podmínky ** . . . 17

5 Cykly **19**
- 5.1 Cyklus repeat . . . 19
- 5.2 Cyklus while * . . . 19
- 5.3 Cyklus for * . . . 20

6 Funkce **21**
6.1 Funkce bez parametrů 21
6.2 Funkce s parametry * 22
6.3 Funkce s návratovou hodnotou ** 22

7 Senzory **24**
7.1 Touch senzor 24
7.2 Light senzor 25
7.3 Sound senzor 25
7.4 Ultarosinic senzor 26

8 Dodatky **27**
8.1 Psaní netradičních znaků 27
8.2 Označení motorů 27

Seznam použitých zdrojů **28**

1 Připravené programy

Pro zrychlení výuky jsou zde uvedeny základní verze programů, které si mžete zkopírovat do odpovídajícího programovacího prostředí.

1.1 Bricx

Tady je to velmi jednoduché:

```
task main()
{

//sem prijdou prikazy

}
```

1.2 CodeBlocx

Tady je to složitější, jelikož jsme v normálním C++:

```
#include <iostream>   //tady jsou funkce cin a cut
#include <math.h>     //matematicke funkce
using namespace std;  //tohle neni treba resit

int main() //hlavni funkce
{

//sem prijdou prikazy

return 0;
}
```

1.3 Dev c++

Zde musíme ještě navíc zařídit, aby se nám po ukončení programu okno hned nezavřelo:

```
#include <iostream>   //tady jsou funkce cin a cut
#include <math.h>     //matematicke funkce
using namespace std;  //tohle neni treba resit

int main() //hlavni funkce
{
int pomocna; //pomocna promenna

//sem prijdou prikazy


cin >> pomocna; //aby se okno po dokonceni programu nezavrelo
return 0;
}
```

2 Stručný přehled

2.1 Tabulka probraných funkcí

V následují tabulce jsou uvedeny funkce jen stručně pro rychlou orientaci.

Obecně	Příklad
Wait(cas);	Wait(5000);
TextOut(poziceX, poziceY, text, nastaveniVypisu);	TextOut(0, 0, "Hello World!", 0);
OnFwd(porty, silaMotoru);	OnFwd(OUT_BC, 75);
OnRev(porty, silaMotoru);	OnRev(OUT_C, 75);
Off(porty);	Off(OUT_C);
OnFwdSync(porty, silaMotoru, pomer);	OnFwdSync(OUT_BC, 75, 50);
RotateMotor(porty, silaMotoru, uhel);	RotateMotor(OUT_AB, 75, 180);
RotateMotorEx(porty, silaMotoru, uhel, pomer, synchronizace, stopka);	RotateMotorEx(OUT_AB, 75, 180, 50, true, true);

Tabulka 1 - Tabulka probraných funkcí.

2.2 Wait

```
void Wait(
    unsigned long  ms
)
```

Zastaví provádění příkazů na určitou dobu danou parametrem **ms**, který se udává v milisekundách (tisícinách sekundy). Funkce ale například nedává žádný příkaz na zastavení motorů. Pokud je tedy zadán robotovi pohyb, pojede stále dál.

- **ms** : doba čekání v milisekundách

2.2.1 Příklad Wait:

```
task main()
{
    TextOut(0,0,"Hello World!",0);
    Wait(5000);
}
```

Vypíše se text a poté program čeká 5 sekund, než se ukončí.

2.3 TextOut

```
char TextOut(
    int  x,
    int  y,
    string  str,
    unsigned long  options = DRAW_OPT_NORMAL
)
```

Vypíše textový řetězec **str** na obrazovku robota a pozici **x**, **y**. (y musí být násobek osmi, tedy 0, 8, 16, ... , 56) Poslední parametr **options** umožňuje další nastavení způsobu výpisu (viz manuál).

- **x** : vzdálenost od levého okraje
- **y** : řádek výpisu
- **str** : text, který bude vypsán
- **options** : další nastavení

2.3.1 Příklad TextOut:

```
task main()
{
TextOut(0,0,"Hello World!",0);
Wait(5000);
}
```

Vypíše se text do levého dolního rohu se základním nastavením a poté program čeká 5 sekund, než se ukončí.

2.4 OnFwd

```
void OnFwd  (
    byte  outputs,
    char  pwr
    )
```

Roztočí motory **outputs** (viz Označení motorů 8.2) rychlostí **pwr** (hodnoty -100 až 100). Motory se bude točit, dokud nedostane další příkaz. Zadáním záporné hodnoty parametru pwr se otočí směr pohybu a zadáním 0 se motory zastaví.

- **outputs** : určení motorů
- **pwr** : síla (rychlost) otáčení

2.4.1 Příklad OnFwd:

```
task main()
```

```
{
    OnFwd(OUT_A, 75);
    OnFwd(OUT_A, -75);
    Wait(2000);
    OnFwd(OUT_A, 0);
}
```

Robot dostane příkaz roztočit motor v portu A dopředu rychlostí 75. Okamžitě však dostane další příkaz na opačný pohyb. Po dvou sekundách otáčení v protisměru je motor zastaven.

2.5 OnRev

```
void OnRev  (
    byte  outputs,
    char  pwr
    )
```

Stejné jako OnFwd, jen točí s motory opačným směrem.

- **outputs** : určení motorů
- **pwr** : síla (rychlost) otáčení

2.6 Off

```
void Off  (
    byte  outputs
    )
```

Vypne motory (vynutí okamžité zastavení).

- **outputs** : určení motorů

2.6.1 Příklad: pohyb motorů:

```
task main()
{
    OnFwd(OUT_A, 75);
    OnRev(OUT_A, 75);
    Wait(2000);
    Off(OUT_A);
}
```

Úplně stejný příklad jako příklad k OnFwd 2.4.1, jen s využitím různých funkcí.

2.7 OnFwdSync

```
void OnFwdSync  (
    byte  outputs,
    char  pwr,
    char  turnpct
 )
```

Roztočí motory **outputs** (viz Označení motorů 8.2) rychlostí **pwr** (hodnoty -100 až 100). Motory se bude točit, dokud nedostane další příkaz. Zadáním záporné hodnoty parametru pwr se otočí směr pohybu a zadáním 0 se motory zastaví. Parametr **turnpct** udává poměr otáčení kol (každé se tedy otáčí různě rychle) a tedy určuje poloměr zatáčení robota. (hodnoty -100 až 100)

- **outputs** : určení motorů
- **pwr** : síla (rychlost) otáčení
- **turnpct** : míra zatáčení

2.7.1 Příklad OnFwdSync:

```
task main()
{
    OnFwdSync(OUT_BC,75,50);
}
```

Robot jede dopředu a potupně zatáčí. ??? jakým směrem ???

2.8 RotateMotor

```
void RotateMotor  (
    byte  outputs,
    char  pwr,
    long  angle
 )
```

Roztočí motory **outputs** (viz Označení motorů 8.2) rychlostí **pwr** (hodnoty -100 až 100). Motory se otočí o úhel **angle**. Zadáním záporné hodnoty parametru pwr se otočí směr pohybu a zadáním 0 se motory zastaví.

- **outputs** : určení motorů
- **pwr** : síla (rychlost) otáčení
- **angle** : úhel otočení

2.8.1 Příklad RotateMotor:

```
task main()
```

```
{
    RotateMotor(OUT_A,75,180);
}
```

otočí motorem A rychlostí 75 o 180 stupňů.

2.9 RotateMotorEx

```
void RotateMotorEx  (
    byte  outputs,
    char  pwr,
    long  angle,
    char  turnpct,
    bool  sync,
    bool  stop
 )
```

Roztočí motory **outputs** (viz Označení motorů 8.2) rychlostí **pwr** (hodnoty -100 až 100). Motory se otočí o úhel **angle**. Zadáním záporné hodnoty parametru pwr se otočí směr pohybu a zadáním 0 se motory zastaví. Parametr **turnpct** udává poměr otáčení kol (každé se tedy otáčí různě rychle) a tedy určuje poloměr zatáčení robota. (hodnoty -100 až 100) ??? který motor se řídí zadaným úhlem ??? Další parametr **sync** určuje, zda se má provádět synchronizace motorů a měl by být nastaven na true, jinak funkce nemám moc smysl... Poslední parametr **stop** udává, zda má robot po dokončení okamžitě nuceně zastavit nebo nechat motory setrvačností dotočit. (Je dobré také dát true, jinak nevíte přesně, kde robot a a baterii snad za každou cenu šetřit nemusíme.)

- **outputs** : určení motorů
- **pwr** : síla (rychlost) otáčení
- **angle** : úhel otočení
- **turnpct** : míra zatáčení
- **sync** : zapnutí/vypnutí synchronizace
- **stop** : zapnutí/vypnutí vynuceného zastavení

2.9.1 Příklad RotateMotorEx:

```
task main()
{
    RotateMotorEx(OUT_AB,75,180,50,true,true);
}
```

otočí motory A a B rychlostí 75 o 180 stupňů ??? který ??? synchronizovaně a na konci vynutí okamžité zastavení.

3 Proměnné

Pro příklady v této kapitole budeme předpokládat spouštění v prostředí s konzolovým výstupem ("černé okno"). Důvod je ten, že zde můžeme použít pohodlný výpis pomocí **cout**. Nyní si připravíme testovací program, ve kterém si můžeme zkoušet níže uvedené příklady:

```
#include <iostream>
using namespace std;

int main()
{

    //sem prijdou prikazy
    //napriklad
    cout << "Ahoj," << endl << "jak se vede?";

    return 0;
}
```

(Řádky za dvěma lomítky jsou komentáře, tedy něco, čeho si počítač nevšímá.) Rovnou jsme zde uvedli příklad použití cout: začíná slovem cout a poté následuje sled věcí, které chceme vypsat oddělených "dvojzobáčky«<. Příkaz **endl** znamená konec řádku, takže celkově program vypíše:

```
Ahoj,
jak se vede?
```

3.1 Co jsou proměnné?

Proměnné nám umožňují pracovat s pamětí počítače. Pro konkrétnost hned uvedeme příklad:

```
int cislo = 5;
```

Tento kód zavádí novou proměnnou, která se jmenuje *cislo*. Slovo **int** znamená, že se jedná o proměnnou pro ukládání **celých** čísel (0, 1, 5, 42, -8, 56854, ...). Hned při vytvoření je do proměnné *cislo* vložena hodnota 5.

Jak už nám sám název napovídá, hodnota proměnné se může při běhu programu měnit. V každém okamžiku je však hodnota uložená v proměnné pevně stanovena. [1] Změna hodnoty proměnné se provádí pomocí -", tedy například:

[1] Srovnání s proměnnými v matematice: V matematice máme například výraz $x = 2y$. S tím můžeme dále pracovat, ale dokud se například nedozvíme, kolik je y, nevíme, kolik je x. Když nám pak někdo y dá můžeme x určit. V programování vytvoříme například opět proměnnou x a náš program na této hodnotě závisí (x určuje otočení motorů). Program pak může fungovat tak, že se hodnota x určí podle okamžité situace, třeba podle vzdálenosti robota od překážky (pomocí ultrasonic senzoru).

```
cislo = 10;
```

3.2 Počítání s proměnnými

Při programování můžeme používat standardní matematické operace: +, -, *, / (plus, mínus, krát, děleno) i bez použití proměnných. Například chceme, aby robot udělal 5 otoček motoru. Nemusíme potřebný úhel počítat dopředu a místo toho můžeme použít příkaz:

```
RotateMotor(OUT_A,75,5*360);
```

Daleko zajímavější je však počítání s proměnnými. Můžeme například novou hodnotu jedné proměnné určit pomocí hodnoty jiné proměnné:

```
int a = 10;
int b = 0;
b = (a+2)*2; // v b je (10+2)*2=24
cout << b;  // vypise b
```

Zde jsme také ukázali, že můžeme normálně pracovat se závorkami (platí standardní přednost operací) a tento program tedy vypíše hodnotu $b = 24$. Ještě zajímavější je, že můžeme použít samotnou proměnnou pro určení její nové hodnoty, tedy například:

```
int a = 10;
a = 40/a+a; // v a je 40/10+10=4+10=14
cout << a;
```

Funguje to tak, že ve výrazu na pravé straně má a všude původní hodnotu, určí se celá hodnota pravé strany a ta se pak uloží do a. Výsledek tedy bude výpis čísla 14. (Poznamenejme, že zde se proměnné v počítači zásadně liší od proměnných v matematice. V matematice například rovnice $a = a + 2$ ani nemá řešení, zato v programování tento příkaz prostě zvýší hodnotu a o dva.)

Zatím jsme se seznámili s proměnnou **typu** int, do které můžeme vkládat celá čísla. V oddíle o typech proměnných si povíme daleko více, ale nyní si pro účely počítání zavedeme ještě proměnné pro ukládání **desetinných** čísel - **float**. Vytvoření může vypadat například takto:

```
float cislo2 = 2.5;
```

Poznamenejme, že při programování vždy používáme desetinnou *tečku*! Také s proměnnými typu float můžeme provádět čtyři základní matematické operace (+, -, *, /). Můžeme i kombinovat proměnné typů int a float. Vše snad osvětlí následující příklad:

```
int a = 10;
int vysledek = 0;
float x = 1.25;

x = 2*x; // v x je 2*1.25=2.5
vysledek = a*x/5;   // ve vysledek je 10*2.5/5=25/5=5

cout << vysledek;
```

Tento program tedy na konci vypíše *vysledek*, ve kterém je číslo 5.

Pozorný čtenář si už jistě všiml, že jsme si ve všech příkladech dali dobrý pozor, abychom do celočíselné proměnné (typu int) neukládali desetinné číslo. V posledním příkladě jsme sice s desetinnými čísly pracovali, ale nakonec vyšlo číslo celé. Pokud se pokusíme uložit desetinné číslo do proměnné typu int, číslo se *zaokrouhlí dolů*! Tedy například

```
int a = 10;
int b = 3;
int c = 0;
float x = 1.25;

c = x;              // do c se ulozi 1
cout << c << endl;  // vypise c a ukonci radek

c = a/b;            // 10/3=3.3333 -> do c se ulozi 3
cout << c << endl;  // vypise c a ukonci radek

c = b/a;            // 3/10=0.3 -> do c se ulozi 0
cout << c << endl;  // vypise c a ukonci radek

c = b/5;            // 3/5=0.6 -> do c se ulozi 0
cout << c << endl;  // vypise c a ukonci radek
```

Na toto je třeba dávat velký pozor. Ještě záludnější je to, že program takto pracuje s celými čísly, která nejsou přiřazena proměnný. Představme si, že máme zadáno otočení motoru ve stupních a chceme spočítat, kolik je to otoček. Použijeme program:

```
float x = 0; //x je DESETINNE cislo

x = 540/360;   // 540/360=1.5 -> do x se ulozi 1!
cout << x;
```

Problém je v tom, že se celý výpočet provede s celými čísly, zaokrouhlí a teprve poté se vloží do x. Program se dá spravit tím, že počítači naznačíme, že má počítat s desetinnými čísly například takto:

```
float x = 0;

x = 540.0/360;   // .0 udělá z 540 desetinné číslo
                    -> do x se ulozi 1.5
cout << x;
```

Jistě se vám někdy stalo, že nějaký program prostě z ničeho nic přestal pracovat a jediná možnost byla ho vypnout. Nyní si ukážeme, jak se to může podařit. Jed o dělení nulou. Na to je třeba dávat pozor, protože pokus o dělení nulou způsobí kolaps programu. Můžete si spustit tento příklad:

```
int a = 0;
```

```
int b = 5;

b = b/a; // tady program spadne!

cout << b;
```

3.2.1 Inicializace proměnných **

Možná jste si všimli, že pokud proměnnou chceme použít jen k uložení nějakého výpočtu, dáváme do ní hodnotu 0. Mohli bychom tam dávat i libovolnou jinou hodnotu, stejně ji pak přepíšeme. Dokonce tam nemusíme dávat nic, tedy je možné proměnnou definovat takto:

```
int a; //vytrori promennou a
```

Navíc si můžeme ušetřit práci při definování více proměnných zápisem:

```
int a,b,c; /vytvori 3 promenne a, b a c
```

Co se ale stane, když bychom nyní nechali tuto proměnnou vypsat nebo dokonce s ní počítat? Můžeme to zkusit:

```
int a;      //vytrori promennou a
int b = 5;

cout << a << endl;   //vypiseme a

cout << a/b; //vypiseme a/5
```

Nyní se však nedá říct, co vlastně program vypíše. Jelikož jsme proměnné a neurčili žádnou hodnotu, tedy jsme ji **neinicializovali**, nevíme, co v ní je. Program však proběhne a něco vypíše a na druhém řádku skutečně vypíše pětinu toho, co vypsal na prvním. Funguje to tak, že při vytvoření proměnné si počítač prostě vezme nějaký kus paměti (u int jsou to 4B) a nyní je v proměnné uloženo to, co zrovna v daném kousku paměti zbylo z běhu jiných programů. Z našeho pohledu tedy vlastně zcela náhodné číslo (kladné i záporné a často dost velké).

3.3 Vstup od uživatele *

Někdy potřebuje, aby náš program komunikoval s uživatelem. Tedy aby ho požádal o vložení nějakých dat a na základě těchto dat mu poskytl odpověď. K tomu se používá příkaz **cin**. Opet začneme příkladem:

```
int a = 0;

cout << "Zadejte a: "; //jen vypise vyzvu

cin >> a;  //do a ulozi to, co uzivatel zadal

cout << "Dlojnasobek a je:" << 2*a; //vypise hodnotu a
```

Tento program si vytvoří proměnnou a, zobrazí výzvu k zadání její hodnoty, pomocí příkazu cin tuto hodnotu uloží do a (všimněte si obrácených zobáčků) a nakonec vypíše dvojnásobek zadané hodnoty.

3.4 Typy proměnných *

S některými typy proměnných jsme se již setkali, nyní se podíváme ne kompletnější (ne kompletní) seznam.

Typ	Co se do něj ukládá
bool	Logucká hodnota false/true, respektive 0/1
int	Celá čísla v rozmezí přibližně ± 2 000 000 000
float	Desetinná čísla (obrovský rozsah a velká přesnost)
char	Jeden znak (písmenu), respektive číslo 0/256
string	Řetězec znaků (slovo, věta,...)

Typ bool **

Typ int

Typ float

Typ char

Typ string

3.5 Zkrácený zápis přiřazení **

3.6 Pole **

3.7 Vícerozměrná pole ***

4 Podmínky

Jednou ze základních možností programu je provádět dvě rozdílné operace na základě splnění/nesplnění určité podmínky. K tomu se používá konstrukce s příkazem **if**. Klasicky uvedeme příklad:

```
float x = 4.6;                    // vytovrime promennou x

if (x>5) {                        // zacatek podminky
cout << "x je velke" << endl;     // prikazy provedene pri splneni
} else {
cout << "x je male" << endl;      // prikazy provedene pri nesplneni
}
```

Nyní se pokusíme v příkladu zorientovat.

Nejprve si vytvoříme proměnnou x a vložíme do ní hodnotu 4.6 (viz kapitola proměnné).

Nyní následuje příkaz *if* (známe z anglických podmínkových vět) a za ním **podmínka** uzavřená v obyčejných závorkách. K tomu, jak může vypadat podmínka se vrátíme později. V našem případě je podmínka splněna, pokud je hodnota x větší než 5, což není.

Následují příkazy, které by se provedly, kdyby podmínka splněna byla. Jsou uzavřeny ve složených závorkách (začíná za podmínkou a končí před "else"). Pokud by tedy bylo x vetší než 5, program by vypsal text "x je velke".

Dále máme příkaz **else** (tedy anglicky "jinak"), které říká, že následují příkazy provedené při nesplnění podmínky.

Příkazy provedené při nesplnění podmínky jsou opět mezi složenými závorkami a program tedy v tomto případě (x není větší než 5) vypíše text "x je malé".

Do lidské řeči (a češtiny) by se tedy dal následující příklad napsat takto:

```
x je 4.6

Když (x je větší než 5) {
Vypiš "x je velke"
} jinak {
Vypiš "x je male".
}
```

4.1 Možné výrazy v podmínce

K určení splnění/nespnění podmínky můžeme požít operátory:

- $<$ (menší než)

- > (větší než)
- <= (menší nebo rovno)
- >= (větší nebo rovno)
- == (je rovno) POZOR! Je potřeba požít dvě rovnítka! (Jedno = znamená přiřazení hodnoty)
- != (není rovno)

Pro lepší představu uvedeme ještě jeden zajímavější příklad (**).

```
int vek,cena;
cout << "Kolik Vam je let?" << endl;
cin >> vek;

//do patnacti let je sleva
if (vek<15) {
    cena = 60;
} else {
    cena = 100;
}

cout << "Cena pro vas je: " << cena << endl;
```

Zde se zavedou proměnné *vek* a *cena*. Program se nyní zeptá, kolik je uživateli let a zadanou hodnotu uloží do proměnné *vek*. Na základě toho, zda je věk uživatele větší, nebo menší než 15 let pak program určí odpovídající cenu a tu následně vypíše.

4.2 Kombinované podmínky **

Někdy potřebujeme udělat složitější podmínku závisející na více okolnostech, k tomu použijeme operátory **and** (tedy "a") a **or** ("nebo"), které se zapisují jako && (dva ampersandy) respektive || (dvě "ořítka", proto se tomu také tak říká :-)). Jako ukázku použití těchto operátorů trochu upravíme předchozí příklad. Slevu nyní dostanou i lidé starší než 60 let.

```
int vek,cena;
cout << "Kolik Vam je let?" << endl;
cin >> vek;

//do patnacti let je sleva
if (vek<15 || vek>60) {                // tady je zmena
    cena = 60;
} else {
    cena = 100;
}

cout << "Cena pro vas je: " << cena << endl;
```

Použitá podmínka je tedy splněna pokud je věk menší než patnáct let *nebo* větší než 60 let.

Pro ilustraci podmínky && použije obdobný příklad. Představme si bazén, kde dávají slevu dětem do patnácti let, ale jen v případě, že chtějí vstupenku alespoň na 3 hodiny. Pak bychom mohli požít program:

```
int vek,cena;
int doba;                                   // nova promenna doba
cout << "Kolik Vam je let?" << endl;
cin >> vek;
cout << "Na jak dlouho jdete?" << endl;    // zadani doby
cin >> doba;

//do patnacti let je sleva
if (vek<15 && doba >= 3) {                  // tady je zmena
    cena = 60;
} else {
    cena = 100;
}

cout << "Cena pro vas je: " << cena << endl;
```

Podmínka je tedy splněna jen pokud je věk menší než 15 let *a* doba delší nebo rovna 3 hodiny.

5 Cykly

Cykly se používají v případě, že chceme nějaký příkaz mnohokrát opakovat. Existuje několik druhů cyklů.

5.1 Cyklus repeat

Nejjednodušším cyklem v Bricks je **repeat**. POZOR: cyklus repeat není v normálním C/C++, takže vám nebude fungovat například v CodeBllocks, ale pouze v Bricx! Uveďme si jednoduchý příklad:

```
repeat (5)
{
    RotateMotor(OUT_B,50,100);
    Wait(1000);
}
```

Je téměř zřejmé, jak příkaz funguje. Anglické slovo *repeat* znamená opakuj a parametr v závorce udává počet opakování. Následují složené závorky, které ohraničují příkazy, které chceme opakovat. (Stejně jako u podmínky může být mezi složenými závorkami příkazů více.) Tento program tedy udělá to, že 5-krát pootočí motorem v portu A a vždy sekundu počká.

Úkol: (cyklus repeat, pohyb) Vytvořte program, který popojede s robotem třikrát dopředu a dozadu s použitím cyklu repeat.

5.2 Cyklus while *

Zajímavější je cyklus **while**. Ten se liší tím, že počet opakování není předen dán. Cyklus na začátku každého opakování kontroluje podmínku a pokud je splněna, pokračuje.

```
int max_rychlost = 90;
int rychlost=1;
while (rychlost<max_rychlost)
{
    OnFwd(OUT_BC,rychlost);
    Wait(50);
    rychlost=rychlost+1;
}
```

Nejprve si zvolíme maximální rychlost (90) a počáteční hodnotu rychlosti (1). Potom program vstoupí do cyklu. Podmínka je splněna (1<90) a robot se rozjede rychlostí rychlost (1), jede touto rychlostí 5 setin vteřiny a zvedne rychlost o 1 (tedy na 2). Cyklus se stále opakuje a robot jede

čím dál větší rychlostí. Jakmile však rychlost dosáhne hodnoty *max_rychlost*, cyklus se ukončí a následně i celý program.

5.3 Cyklus for *

Jedná se o cyklus, u kterého je předem dáno, kolikrát se bude opakovat. Může vypadat například takto:

```
for (int i=0;i<100;i++) {
cout << "Uz nikdy nebudu rozebirat roboty jinych skupin." << endl;
}
```

Zkusme nyní program postupně rozluštit. Slovo *for* pochopitelně značí, že následuje for cyklus. Dále jsou zde obyčejné závorky obsahující tři příkazy oddělené středníkem.

1. int i=0 - Zavede proměnnou i, do které uloží hodnotu 0. Tato proměnná nám říká, v kolikátém běhu cyklu se právě nacházíme.
2. i<100 - Podmínka, která určuje, zda má cyklus běžet znovu. Cyklus běží, dokud je podmínka splněna. V našem případě tedy dokud i (počet již proběhlých opakování) je menší něž 100. Cyklus tedy poběží právě 100-krát.
3. i++ - Příkaz, který při každém opakování zvýší i o jedničku.

Pro základní použití těmto příkazům vlastně vůbec není třeba rozumět. Prostě jen číslo 100 nahradíme počtem cyklů, které potřebujeme.

Následují příkazy provedené v každém běhu cyklu opět obalené ve složených závorkách. Tento program tedy udělá to, že 100-krát napíše větu: "Uz nikdy nebudu rozebirat roboty jinych skupin."

6 Funkce

Počítače fungují tak, že máme nějaké základní operace, které můžeme provádět a z nich postupně vytváříme příkazy (funkce) složitější. Tím se vlastně počítač "učí"nové věci. Nyní se tedy naučíme, jak si připravovat vlastní složitější funkce z funkcí jednodušších a jak tyto nové funkce používat.

6.1 Funkce bez parametrů

Nejjednodušší varianta funkce je taková, která dělá vždy to samé. není tedy třeba jí při použití dávat nějaké další údaje. Tu můžeme vytvořit například takto:

```
#include <iostream>
using namespace std;

int tamAZpet()                  // zde se vytvori nova funkce
{
    RotateMotor(OUT_BC,75,180);   // prikazy, ktere funkce dela
    RotateMotor(OUT_A,75,90);
    RotateMotor(OUT_BC,75,-180);
    return 0;
}

int main()
{
    tamAZpet();             // pouziti funkce
    return 0;
}
```

Zde jsme uvedli celý program, aby bylo vidět, kam definice nové funkce patří: mimo funkci *main*! a to konkrétně nad ni. nejprve tedy určíme, jak se bude nová funkce jmenovat, tedy *tamAZpet*. Slova *int* před názvem nové funkce si zatím nebudeme všímat je jen ho tam prostě dáme (vysvětlení níže). I prázdné závorky je třeba napsat!

Následuje klasický blok příkazů obalený ve složených závorkách. Zde se jedná o pohyb robota dopředu, otočení třetím motorem a návrat robota zpět.

Příkaz *return 0* opět zatím jen opíšeme.

Nakonec je ve funkci *main* nová funkce *tamAZpet* použita. (Pozor na správné psaní velkých a malých písmen. Zde jsme použili konvenci, že každé slovo v názvu kromě prvního začíná velkým písmenem.)

6.2 Funkce s parametry *

Funkce s parametry dobře známe. Jsou to všechny funkce, které jsme zatím používali. Například funkce pro pohyb motorů přijímají parametry pro sílu motoru, počet stupňů otočení, ... I naše funkce mohou mít parametry, které nějak ovlivňují jejich průběh. Zde je příklad takové funkce:

```
int otocka(float rKola, float roztec, int uhel)
{
    float otoceni = roztec*uhel/rKola,100;          // vypocet otoceni
    RotateMotorEx(OUT_AB,100,otoceni,true,true);      // provedeni pohybu
    return 0;
}
```

Tato funkce se jmenuje *otocka* a vstupují do ní dva parametry typu desetinného čísla (*rKola* a *roztec*) a jeden celočíselný parametr (*uhel*). Funkce dělá to, že otočí robota o zadaný úhel (tedy třeba úhel 180° je čelem vzad). Jelikož však můžeme určit jen úhel otočení motorů a ne úhel otočení robota (to je úplně něco jiného), musíme vědět, jak robot vypadá. Známe-li velikost kol a vzdálenost mezi koly (tedy velikost kružnice, kterou kola při otočce opisují), můžeme správné otočení motorů vypočítat. (Odvození výpočtu zde nebudeme rozebírat.) Následně se tedy odpovídající otočení motorů provede.

Tuto funkci bychom mohli použít například s parametry:

```
otocka(2.5, 5, 180);
```

Chceme tedy otočit robota s koly o poloměru 2.5 cm a roztečí kol 5 cm čelem vzad. (Jelikož se rozteč ve výpočtu dělí poloměrem kol, je jedno, v jakých jednotkách tyto veličiny uvedeme. Jednotky jen musí být stejné.)

6.3 Funkce s návratovou hodnotou **

Nyní si řekneme, co znamená to *int* před názvem funkce a *return* na konci. Funkce nemusí jen přímo provádět příkazy. Může například něco vypočítat ze zadaných hodnot a výsledek předt k dalšímu použití. Jako příklad si uvedeme jednoduchou funkci, které zadáme počet otoček a ona nám spočte, kolik je to stupňů.

```
#include <iostream>
using namespace std;

int otockyNaStupne(float otocky)
{
    int stupne = otocky*360;
    return stupne;
}

int main()
{
    int pocetStupnu;
```

```
    pocetStupnu = otockyNaStupne(5.78);
    cout << pocetStupnu;
}
```

Slovo *int* před funkcí tedy říká, že naše funkce bude vracet celočíselný výsledek. Nejprve se ze zadaného počtu otoček vypočte počet stupňů. Dále následu příkaz **return** (anglicky "vrátit"), který právě vrátí vrátí hodnotu proměnné *stupne*. Tím tedy říkáme, že právě hodnota této proměnné je to, co má být výsledkem naší funkce.

Ve funkci main naši funkci použijeme. Do proměnné *pocetStupnu* vložíme výsledek naší funkce *otockyNaStupne* s parametrem 5.78 otočky a výsledek vypíšeme. Tak se dozvíme, že 5.78 otočky je 2080 stupňů.

7 Senzory

Senzory slouží k tomu, aby mohl robot nějakým způsobem reagovat na své okolí.

Na začátku programu je třeba senzory nastavit a přiřadit ke správným portům. Tady je příklad:

```
SetSensorTouch(IN_1);
SetSensorLight(IN_3);
SetSensorSound(IN_2);
SetSensorLowspeed(IN_4); //ultrasonic
```

Samozřejmě nemusíme přiřadit všechny senzory, můžeme je přiřadit na jiné porty nebo můžeme na různé porty přiřadit více senzorů stejného typu.

Každý senzor měří nějakou fyzikální veličinu a my můžeme používat naměřenou hodnotu. To uděláme pomocí "proměnných":

```
SENSOR_1
SENSOR_2
SENSOR_3
SENSOR_4
SensorUS(IN_4) // pro ultrasonic
```

Ve skutečnosti se nejedná o proměnné, jelikož do nich nemůžeme nic ukládat. Nicméně můžeme jejich hodnotu uložit do jiné proměnné, například:

```
SetSensorLight(IN_1);
int x;
x=SENSOR_1;
```

Tento program uloží do proměnné x hodnotu úrovně světla naměřenou pomocí light senzoru na začátku programu.

7.1 Touch senzor

Touch senzor má jen dvě hodnoty: 0 nestlačený, 1 stlačený. Uveďme si příklad (cyklus while):

```
působem reagovat
task main()
{
    SetSensorTouch(IN_1);
```

```
    OnFwd(OUT_BC, 75);
    while (SENSOR_1 == 0);
    Off(OUT_BC);
}
```

(Předpokládáme, že robot má nějaký nárazník napojený na touch senzor.) Nejprve nastavíme touch senzor na port 1. Robot se poté rozjede dopředu. Následující cyklus neprovádí žádné příkazy, jen stále dokola opakuje, zda je hodnota v touch senzoru 0 (tedy senzor není stlačen). Jakmile robot do něčeho narazí, stlačí se senzor, hodnota $SENZOR\_1$ se změní na 1 a cyklus se ukončí. Program konečně pokračuje na další příkaz, který zastaví motory.

7.2 Light senzor

Light senzor měří množství světla odraženého od povrchu (tedy nerozeznává barvy). Jednoduchý příklad:

```
task main()
{
    int hranice=40;
    SetSensorLight(IN_3);
    OnFwd(OUT_BC, 75);
    while (SENSOR_3 < hranice);
    Off(OUT_BC);
}
```

Příklad je zcela obdobný předchozímu příkladu na touch senzor. Jediný rozdíl je v tom, že robot se zastaví, když úroveň světla přesáhne určitou hranici (robot najede na bílou čáru).

7.3 Sound senzor

Sound senzor snímá zvuk a vrací hodnotu jeho hlasitosti. Zase stejný příklad:

```
task main()
{
    int hranice=40;
    SetSensorSound(IN_2);
    OnFwd(OUT_BC, 75);
    while (SENSOR_2 < hranice);
    Off(OUT_BC);
}
```

Stejné jako jako předchozí příklady. Nyní by se robot zastavil například na tlesknutí.

7.4 Ultarosinic senzor

Ultrasonic senzor vysílá ultrazvukový signál a podle doby, jako signál potřebuje na návrat určuje vzdálenost od objektu. Problém může být, když se vyslaný signál vůbec neodrazí k robotovi. V takovém případě robot nic "nevidí". Stává se to celkem často, tak je potřeba na to dávat pozor. Zase stejný příklad:

```
task main()
{
    int hranice=20;
    SetSensorLowspeed(IN_4);
    OnFwd(OUT_BC, 75);
    while (SENSOR_4 < hranice);
    Off(OUT_BC);
}
```

Stejné jako jako předchozí příklady. Nyní by se robot zastavil pokud je už blízko u překážky.

Dalším příkladem může být robot, který se vyhýbá překážkám:

```
task main()
{
    int hranice=50;
    SetSensorLowspeed(IN_4);
    while (true) //nekonecna smycka
    {
     if (SensorUS(IN_4) > hranice) {
         OnFwd(OUT_BC, 75);
     } else {
            otateMotor(OUT_B, -75,200);
        }
}
}
```

V programu se stále opakuje nekonečná smyčka, ve které se kontroluje, zda je v blízkosti překážka. Pokud ne, robot jede stále dopředu. Pokud ano, robot se pootočí jiným směrem.

8 Dodatky

8.1 Psaní netradičních znaků

Často budeme potřebovat psát znaky, které se v normální textu moc nepoužívají. Budeme je psát pomocí klávesových zkratek s **Alt Gr**, což je pravý Alt. (S levým altem to nebude fungovat!)

Znak	**Klávesová zkratka**	**Název**
@	Alt Gr + v	zavináč
&	Alt Gr + c	ampersand
|	Alt Gr + w	svislítko ("ořítko")
[	Alt Gr + f	levá hranatá závorka
]	Alt Gr + g	pravá hranatá závorka
{	Alt Gr + b	levá složená závorka
}	Alt Gr + n	pravá složené závorka
<	Alt Gr + , (bývá to na klávese)	menší
>	Alt Gr + . (bývá to na klávese)	větší
\	Alt Gr + q	zpětné lomítko (backslash)

Tabulka 1 - Netradičních znaků.

8.2 Označení motorů

Pro označení motorů jsou předdefinované konstanty. Vždy je uvedeno textové označení a příslušná hodnota. (Jako parametry funkcí je tedy možné používat přímo čísla.) Je snad zřejmé, jaké motory konstanta označuje :-).

```
OUT_A     0
OUT_B     1
OUT_C     2
OUT_AB    3
OUT_AC    4
OUT_BC    5
OUT_ABC   6
```

Seznam použitých zdrojů

[1] HANSEN J. *NXC Programmer's Guide*. Verze 1.2.1 r4, 2011.