Programovacia príručka k stavebnici LEGO NXT Mindstorm.

Ing. Tomáš Kasanický
tomas.kasanicky@gmail.com

Obsah

Zoznam tabuliek

Zoznam obrázkov

Úvod

Táto programátorská príručka vznikla za podpory európskeho sociálneho fondu v rámci projektu Mobilná trieda.

Je určená pre frekventantov kurzu mobilná trieda. Spôsob a štýl, akým je napísaná, je prevažne určený pre stredné školy. Príručka pojednáva o programovacom jazyku NXC, postavenom na syntaxe jazyka C. Cieľom tejto príručky nie je vytvoriť úplnú referenciu k danému jazyku, ale naopak podať stručné a prehľadné základy daného jazyka tak, aby čitateľ bol schopný po jej preštudovaní samostatne pracovať. Príručka je členená do troch základných častí:

- Operácie a premenné
- Kontrola toku informácií
- Funkcie programovacieho jazyka NXC

Stať Operácie a premenné pojednáva o základných spôsoboch narábania s premennými ako aj primitívnymi operáciami nad nimi.
Kontrola toku informácií sa venuje oboznámeniu čitateľa so spôsobmi, ako sa dá v programe postupovať pri riadení jeho chodu.
Časť Funkcie programovacieho jazyka NXC sa venuje vymenovaniu a predvedeniu základných funkcií pre prácu s legom NXT.

Operácie a premenné

Operácie

V jazyku NXC sú povolené nasledovné operácie nad rôznymi dátovými typmi.

Operátor	Akcia
a = b	Priradenie b do a
a+=b	a = a+b
a-=b	a = a-b
a*=b	a = a*b
a/=b	a = a/b
a%=b	a = a%b
a&=b	Bitový AND a = a&b
a|=b	Bitový OR a = a|b
a^=b	Bitový exlusive OR
a||=b	Do a priradí absolútnu hodnotu b
sign(a)	Znamienko premennej a
abs(a)	Absolútna hodnota z a
a++,b --	Inkrement a+=1 a dekrement b+=1
!	Logická negácia
? :	Podmienená hodnota a==1 ? b : c
||	Logicky OR
&&	Logicky AND
|	Bitový OR
^	Bitový XOR
&	Bitový AND
a==b, a!=b	a je rovné b , a je rôzne od b
a < b, a >b , a <= b, a >=b	a je menšie ako b, a je väčšie ako b a je menšie rovné b, a je väčšie rovné b
<<, >>	Bitový posuv vľavo, vpravo
+, -	Sčítanie, odčítanie
*, /, %	Násobenie, delenie, modulo

Tabuľka 1. Operácie

Premenné

Sú základnými prvkami programovacieho jazyka, slúžia na uchovávanie hodnôt a stavov.

Mená premenných	Reprezentácia
bool	8 bitov bez znamienka
byte	8 bitov bez znamienka
char	8 bitov so znamienkom
unsignet int	16 bitov bez znamienka
short int	16 bitov so znamienkom
unsignet long	32 bitov bez znamienka
long	32 bitov so znamienkom
mutex	Špeciálny typ pre prístup do kódu
string	Pole bajtov
struct	Užívateľom nadefinovaná štruktúra
Arrays	Pole ľubovoľného typu

Tabuľka 2. Premenné

Premenné sú deklarované v programe pomocou kľúčového slova označujúceho typ premennej. Rovnaké typy sú oddeľované čiarkou, deklarácia sa končí bodkočiarkou.

```
int x;              //deklaracia premennej x
int    y,z,w=1;    /*deklaracia    premennej    y,z,w
                   inicializacia w*/
long  Premenna  =  32;   /*deklaracia  a inicalizacia
                         premennej*/
```

Premenné je možno rozdeliť podľa viditeľnosti na globálne a lokálne. Globálne premenné sú viditeľné v celom programe:

```
int x = 0;       /*globalna premenna vidtelna v danej
                 oblasti*/
task main()
  {
   int r = 3;     /*lokalna    premenna    videtelna
                  v ulohe*/
     {
         int z;   /*lokalna    premenna    videtelna
                  v hranicach zatvoriek {} /*
         z = x;
   }
```

```
  r=x;
  z=r;      //CHYBA   premenna   nexistuje   v tejto
casti
}
```

Naproti tomu lokálne platia len v istom priestore.

Štruktúry v jazyku NXC sú deklarované podobným spôsobom ako v jazyku C. Na začiatku je nutné definovať štruktúru a jej obsah (z hľadiska typov):

```
struct obdlznik
{

  int strana_A;
  int strana_B;

};
```

Po deklarácii môžeme zadefinovať samotnú premennú, ktorú budeme používať:

```
obdlznik stvorec;
```

Na samotné naplnenie štruktúry sa používa nasledovný zápis:

```
stvorec.strana_A = 5;
stvorec.strana_B = stvorec.strana_A;
```

Polia sú deklarované rovnakým spôsobom ako ostatné premenné, ale za menom nasleduje dvojica hranatých zátvoriek []:

```
int pole_intov[];       //jednorozmerne pole
int dvojrozmerne_pole[];//dvojrozmerne pole
```

samotná inicializácia môže byť prevedená rovno pri deklarácii:

```
int   pole_cisel[]   =   {1,2,3,4,5}/*pole   o piatich
                                    prvkoch*/
```

avšak ak chceme inicializovať pole inde ako deklarujeme, je nutné použiť funkciu `ArrayInit(meno_pola,obsah,pocet prvkov):`

```
task main()
{

   int pole[];
   int pole_a[];
   int dvojrozmerne_pole[][];
   ArrayInit(pole,0,3);
        //vytvori sa pole 10-tich nul
   ArrayInit(dvojrozmerne_pole,pole,3);
      //vytvori sa dvojrozmerne pole naplnene 0
      //{0,0,0}
      //{0,0,0}
      //{0,0,0}
   ArrayBuild(pole_a,1,2,3,4,5);
      //vytvori sa pole o 5 prvkoch {1,2,3,4,5}

}
```

Kontrola toku informácií

Základnou bunkou tvorenia kontroly nad štruktúrou programu je dvojica krčených zátvoriek `{}`. Krčená { zátvorka označuje začiatok bloku a } zátvorka označuje koniec bloku. Týmto rozčlenením sa riadi aj viditeľnosť premenných, teda ich lokálnosť, či globálnosť.

```
{
   x = 2;
   int y = 2; /*premenna je viditelna len v ramci
               bloku*/
}
```

Podmienky

Jedným z najprimitívnejších príkazov pre riadenia toku je príkaz `goto`. Jedná sa o takzvaný nepodmienený skok. Je ho možné použiť na prechod do inej časti programu. Príklad použitia:

```
//
// telo programu
//
goto navestie;
//
//telo programu ktore chceme preskocit
//
navestie:    /*miesto kam sa presunie vykonavanie
             programu po narazeni na prikaz goto
             navestie;*/
//
//
```

Základným príkazom pre riadenie toku informácií je podmienka. V jazyku NXC je nazývaná `if`. Telo, ktoré ohraničuje príkaz `if` je vykonané, ak je výraz zapísaný v () pravdivý, teda vracia `true`.

```
if(podmienka) dosledok
if(podmienka)
{
   dosledok
 }
```

Prípadne s použitím kľúčového slova `else`, ktoré reprezentuje čo sa má vykonať, ak nie je splnená podmienka:

```
if(podmienka) dosledok else iny_dosledok
if(podmienka)
{
   dosledok
    }
else
{
   iny_dosledok
}
```

Tak ako bolo naznačené, pre rozsiahlejšiu sekvenciu dôsledkov je možné použiť kučeravé zátvorky na ohraničenie pôsobnosti `if` a `else`. Nižšie uvedený príklad vykonáva triedenie dvoch premenných (Jano, Marek) podľa veľkosti :

```
rovnaky=0;
if (Jano > Marek)
{
   vyssi = Jano;
   nizsi = Marek;
}
else
{
   if (Jano == Marek)

   {
        rovnaky = 1;
   }
   else

   {
        vyssi = Marek;
        nizsi = Jano;
   }
}
```

V prípade, že je potrebné ošetriť viacero hodnôt jednej premennej, je vhodné použiť príkaz `switch`. `switch`, teda prepínač, spusti tú časť kódu, ktorá je ohraničená prípadom (`case`), ktorý sa zhoduje s hodnotou, ktorú má premenná vyšetrovaná príkazom `switch`. Použitie príkazu `switch` vyzerá následovne:

```
switch (premenna)
{
   case 5:
        /*nieco sa vykona ak premenna nadobudne
hodnotu 5*/
        break;
   case 1:
   case 2:
        /* nieco sa vykona ak premenna nadobudne
        hodnotu 1 alebo 2 */
        break;
   deafult:
        /* nieco sa vykona ak premenna je rozna od
        1,2,5*/
        break;

}
```

NXC umožňuje použiť v príkaze `switch` premennú typu `string`, teda:

```
string retazec;
switch (retazec)
{
   case ‘tri’:
           premenna = 3;
           break;
   case ‘dva’:
           premenna = 2;
           break;
   default:
           premenna = 100;
}
```

Cykly

Cyklami označujem príkazy, ktoré majú za úlohu vytvárať v programe slučky. Pod slučkou rozumieme opakovaný prechod vyhradenou časťou programu. V NXC je implementovaných viacero spôsobov pre vytváranie cyklov. Na vytvorenie podmienenej slučky je možné použiť príkaz `while(vyraz).` Pokiaľ je výraz v zátvorke pravdivý, teda vracia hodnotu true, do vtedy trvá slučka. V prípade, ak výraz v zátvorke vracia false, slučka je ukončená bez prechodu blokom, ktorý vytyčuje `while(vyraz).` Nižšie uvedený príklad demonštruje vytvorenie slučky, ktorá vykoná 4 opakovania.

```
int vyraz = 1;
while(vyraz < 5)
{
   vyraz += 1;
}
```

V prípade, ak potrebujeme, aby vyhodnocovanie podmienky nebolo na začiatku slučky ale na konci, môžeme použiť iný tvar cyklu `while` a to `do-while`. Uvedený príklad je funkciou totožný s prikladom `while`.

```
int vyraz = 1;
do
{
   vyraz +=1;
}while(vyraz < 5);
```

V jazyku NXT existuje zadefinované makro `until`. Jeho definícia je nasledovná:

```
#define until(c) while(!(c))
```

Teda opakuje, pokiaľ c je false.

Často používaným príkazom pre vytvorenie slučky je príkaz `for(;;).`

```
for(operacia1;podmienka;operacia2)
{
   telo
}
```

Príkaz `for` pracuje tak, že na začiatku vykoná `operaciu1,` následne prekontroluje `podmienku`, pokiaľ je splnená, vykoná telo a potom vykoná `operaciu2.` Takže prepis príkladu `while a do-while` pomocou príkazu `for` vyzerá nasledovne :

```
for(vyraz=1; vyraz < 5; vyraz +=1)
{
}
```

Jednoduchším ekvivalentom príkazu `for` je príkaz `repeat().` Vykoná toľko prechodov cez telo, akú hodnotu dosahuje jeho vstup. Ekvivalent 4 prechodov vyzerá nasledovne:

```
repeat (5)
{
}
```

Funkcie programovacieho jazyka NXC.

Jazyk NXC, ktorý je viazaný na stavebnicu Lego Mindstorm NXT poskytuje veľkú variáciu funkcií, ktoré pracujú s hardwerovými časťami programovateľnej kocky Lega NXT mindstorm. Nasledujúca časť tejto príručky je venovaná popisu základných a najpoužívanejších funkcií pre prácu so samotnou kockou NXT, ako aj jej perifémymi zariadeniami, medzi ktoré patria snímače a motory. Text je rozdelený na dve hlavée časti:

- NXT kocka
- Snímače a motory

Časť nazvaná NXT kocka popisuje jednotlivé funkcie samotnej kocky:

- Časovanie
- Zobrazovanie, práca s obrazovkou
- Zvuky
- Matematické funkcie

Naproti tomu stať Snímače a motory popisuje prácu so snímačmi a motormi.

NXT kocka

Funkcie pre prácu s časom

Pri mnohých úlohách v robotike je nutné použiť presný časový údaj. Od toho údaju často záleží správna funkčnosť celého zariadenia. Medzi základné funkcie jazyka NXT pre prácu s časom môžeme zaradiť príkaz `wait()`. Argumentom tejto funkcie je číslo vyjadrujúce dobu, po ktorú sa pozastaví program. Toto číslo je časový udaj v milisekundách. Teda zápis:

```
wait(1000);          // cakaj 1s
wait(random(1000)); /* cakaj nahodne vybranu hodnotu
                    od 0 po 1s*/
```

znamená, že program sa pozastaví na 1s a následne sa pozastaví na náhodne vybranú hodnotu medzi 0 a 1s. V kocke NXT bežia vnútorné hodiny, ktoré časujú jednotlivé zariadenia. Tieto hodiny bežia neustále po zapnutí kocky.

Ak chceme zistiť ich stav (stav 32 bitového počítadla) po spustení programu, slúži na to príkaz `FirstTick()`.

```
stav_na_zaciatku = FirstTick() ;
```

Naproti tomu, ak chceme zistiť súčasný stav časovača, môžeme tak urobiť pomocou funkcie `CurrentTick()` nasledovným spôsobom:

```
sucasny_stav = CurrentTick();
```

Kocka NXT ma pre ochranu batérií pred vybitím zavedenú funkciu, ktorá kocku bez ohľadu na to, v akom je stave, po istom čase vypne. Pre prácu s touto možnosťou slúžia nasledovné funkcie. Ak sa rozhodneme v prípade nejakej udalosti, ktorá si vyžaduje uviesť kocku do vypnutého stavu, môžeme tak učiniť pomocou príkazu `SleepNow()`.

```
SleepNow(); //uvedie kocku do vypnuteho modu
```

V prípade, ak tak nechceme urobiť okamžite, ale až o nejakú dobu, môžeme použiť príkaz `SetSleepTimer(minuty)`.

```
SetSleepTimer(6) // nastavi hodnotu casovaca na 6min
```

Ak chceme nanovo spustiť odratávanie času do vypnutia, je možné pre tento účel použiť funkciu `ResetSleepTimer()`

```
ResetSleepTimer(); //zacne nanovo odratavanie
```

Správne načasovanie volania tejto funkcie, nám môže zabezpečiť ochranu pred nežiaducim vypnutím kocky. Pokiaľ nevieme koľko času nám ostáva do vypnutia kocky, môžeme to zistiť pomocou funkcie `SleepTime()`

```
Cas_do_vypnutia = SleepTime();
```

Funkcie pre prácu s reťazcami

Dátový typ reťazec označovaný v jazyku NXC ako string slúži na prácu s textom. Reťazec uchováva znaky. V jazyku NXC je implementovaných viacero funkcií, ktoré uľahčujú pracú s reťazcami. Jedná sa o konverziu reťazcov na iné typy, členenie reťazcov, teda ich formátovanie. Jednou

z najčastejších úloh pri práci s reťazcami je ich konverzia na iné dátové typy. Medzi základne konverzie patrí konverzia čísla na reťazec a reťazca na číslo. Predstavme si, že máme reťazec znakov „123“ reprezentujúci číslo 123. Teda nech:

```
string retazec = “123“;  /*vytvorenie retazca a jeho
                           naplnenie*/
int cislica = 0;
cislica = StrToNum(retazec);
```

tak príkaz `StrToNum()` vráti číslicu. Existuje aj opačná konverzia `NumToStr()`. Teda spätná konverzia je :

```
retazec = NumToStr(cislica);
```

Ďalšou často požívanou konverziou je konverzia pola bajtov na reťazec. Pre tento účel slúži funkcia `ByteArrayToStr(pole_bajtov)`.

```
retazec = ByteArrayToStr(pole_bajtov);
```

`pole_bajtov` musí byť jednorozmerné pole typu byte. Pre opačný prípad, kedy potrebujeme z reťazca získať pole bajtov, je možné použiť funkciu `StrToByteArray()`. Funkcia má dva vstupné parametre, jeden tvorí string, ktorý je potom preložený do pola bajtov, ktoré sa uloží do druhého vstupného parametra.

```
StrToByteArray(retazec, pole_bajtov);
```

Za účelom formátovania reťazcov je naprogramovaných viacero funkcií. Často používanou je funkcia, ktorá spája viaceré reťazce do jedného.

```
string retazec,retazec1 = “Jano“,retazce2 =“ je
“,retazceN = “ maly“;
retazec = StrCat(retazec1, retazec2, retazecN);
```

V reťazci `retazec` sa nachádza veta „Jano je maly“. Funkciou, ktorá vráti len špecifickú časť reťazca, je `SubStr(retazec, miesto, pocet_znakov)`.

```
string novy_retazec;
novy_retazec = SubStr(retazec,7,4);
```

Premenná `novy_retazec` bude obsahovať „maly“, pretože sa začína 7 znakov od začiatku reťazca (rátané od 0) a má 4 znaky. V prípade, že nechceme vrátiť časť reťazca, ale len ho premenovať, môžeme použiť príkaz `StrReplace(retazec, miesto, novy_retazec)`. Jeho použitie si môžeme ukázať na príklade, kedy v premennej `retazec` chceme zameniť meno Jano za meno Fero.

```
retazec = StrReplace(retazec, 0, “Fero“) ;
```

Príkaz nájde 0 pozíciu od začiatku a zapíše tam reťazec “Fero”, ktorý prepíše pôvodný reťazec “Jano“. A teda premenná `retazec` bude obsahovať vetu “Fero je maly“. Ak potrebujeme zistiť dĺžku reťazca, môžeme tak urobiť využitím funkcie `StrLen(str)`

```
int dlzka_retazca;
dlzka_retazca = StrLen(retazec);
```

Čiže v premennej `dlzka_retazca` sa nachádza číslo 11.

Matematické funkcie

Jazyk NXC je špecifický tým, že nepodporuje operácie s desatinnými číslami a ani dátové typy reprezentujúce desatinné čísla. Táto skutočnosť spôsobuje, že matematické funkcie ináč pracujúce nad desatinnými číslami sú upravené tak, aby pracovali výhradne s celými číslami. Jedna zo skupín funkcií, ktorých sa to týka, sú aj goniometrické funkcie. Ich výsledné hodnoty, ako vieme, sú desatinné čísla. Tento problém sa vyriešil tak, že goniometrické funkcie vracajú 100 násobok svojej hodnoty.

```
int a,uhol = 90;
a = Sin(uhol); //sinus uhla vrati 100x hodnoty
a = Cos(uhol); //cosinus uhla vrati 100x hodnoty

a = Asin(uhol);  /*Arcus sinus vstupom je 100x
                 hodnoty vrati cislo v rozmedzi +-
                 90 stupnov*/
a  =  Acos(uhol);/*Arcus  cosinus  vstupom  je  100x
                 hodnoty  vrati  cislo  v  rozmedzi  +-
                 90 stupnov*/
```

Ďalšou často používanou matematickou funkciou je odmocnina. Pre jej realizáciu je v jazyky NXC zavedená funkcia `Sqrt()`, ktorá vracia druhú odmocninu z čísla

```
int odmocnina, cislo = 4;
odmocnina = Sqrt(cislo);
```

Pre vygenerovanie náhodných čísel slúži príkaz `Random()`. Vracia číslo v rozsahu, ktorého spodný okraj je 0 a horný je definovaný vstupným argumentom.

```
int cislo,hranica = 10;
cilso = Random(hranica); /*cislo obsahuje hodnotu z
                           intervalu [0,9]*/
```

Zobrazovanie

NXT kocka obsahuje zobrazovaciu monochromatickú jednotku z tekutých kryštálov. Tento displej je možné používať pre zobrazovanie znakov, ako aj grafiky. Usporiadanie bodov na zobrazovacej jednotke je rátané od ľavého dolného rohu, kde sa nachádza súradnicová poloha 0,0. Y-ová zložka smeruje nahor a X-ová do strany. Pre jednoduchšiu pracú s reťazcami je v jazyku NXC zadefinovaných osem riadkov. Tie sú definované pomocou konštánt, ktoré obsahujú už vopred nadefinované hodnoty súradnice y, ktorá prezentuje polohu riadku na zobrazovacej jednotke. Sú to tieto konštanty:

```
LCD_LINE1, LCD_LINE2,
LCD_LINE3, LCD_LINE4,
LCD_LINE5, LCD_LINE6,
LCD_LINE7, LCD_LINE8
```

Pre výpis číslice na obrazovku slúži príkaz `NumOut(x,y,cislo,zmazanie)`, jeho pramatere sú poloha počiatočného bodu x, y, samotné číslo, ktoré chceme vypísať a parameter zmazania, ktorý, ak sa nezadá, je automaticky nastavený na hodnotu `false`.

```
int cislo = 9;
NumOut(10,LCD_LINE3,cislo); /* cislo 9 sa vypise na
                               treti   riadok   desat
                               bodov z kraja*/
NumOut(10,20,15);            /* cislo 15 sa vypise
                               na poziciu 10,20*/
```

Pre výpis textu, respektíve reťazca, je pripravená funkcia `TextOut(x,y,text,zmazanie)`. Vstupné pramatere tejto funkcie sú rovnaké ako parametre `NumOut()`, rozdiel je však v type. `NumOut()` má ako vstup pre výpis typ `int` a `TextOut()` reťazec, teda `string`.

```
string text = "pous";
TextOut(10,LCD_LINE2,text);
TextOut(10,10,"pokus2");
```

NXT umožňuje vykresliť na obrazovku okrem textu a reťazcov aj rôzne geometrické útvary:

- Kruh `CircleOut(x,y,priemer,zmazanie)`
- Čiara `LineOut(xz,yz,xk,yk,zmazanie)`
- Bod `PointOut(x,y,zmazanie)`
- Obdĺžnik `RectOut(x,y,sirka,vyska,zmazanie)`

Príkaz `CircleOut(x,y,priemer,zmazanie)` vykreslí kružnicu, pričom jej poloha je definovaná pomocou `x,y` a diameter pomocou premennej `priemer`.

```
CircleOut(20,20,10);
```

Príkaz `LineOut(xz,yz,xk,yk,zmazanie)` vykreslí čiaru, ktorá začína v bode `xz,yz` a končí v bode `xk,yk`.

```
LineOut(10,10,20,20,true); //zmaze prv ako kresli
```

Príkaz `PointOut(x,y,zmazanie)` vykresli bod o veľkosti jedného pixelu na pozícii `x,y`.

```
PointOut(5,5);
```

Príkaz `RectOut(x,y,sirka,vyska,zmazanie)` vykreslí obdĺžnik so začiatkom v bode `x,y` pričom jeho rozmery definuje `sirka, vyska.`

```
RectOut(10,10,10,10); // vykresli stvorec 10x10
```

Ak chceme vyčistiť obrazovku, môžeme tak urobiť pomocou príkazu `ClearScreen()`.

```
ClearScreen();//zmaze celu obrazovku
```

Akonáhle začneme využívať obrazovku v našom programe, spôsobí to, že štandardná obrazovka, ktorá symbolizuje beh programu, je nahradená našim výstupom. Ak sa chceme vrátiť k pôvodnej obrazovke, slúži na to príkaz `ResetScreen()`.

```
ResetScreen();//vrati obrazovku do povodneho stavu
```

Zvuky

Kocka NXT obsahuje zvukový výstup, ktorý je možné použiť pre signalizáciu, prípadne prezentáciu. V prípade, ak chceme, aby kocka vydala zvuk o špecifickej frekvencii a trvaní, použijeme príkaz `PlayTone(frekvencia,trvanie);`

```
int frekvencia = 440, trvanie = 1000;
PlayTone(frekvencia,trvanie);/*  ton  a na  stupnici,
                                po dobu 1s */
```

Hlasitosť, ktorá je nastaviteľná v štyroch krokoch sa dá zistiť pomocou príkazu `SoundVolume()`.

```
SoundVolume();//vrati hodnotu hlasitosti
```

Ak chceme nastaviť hlasitosť, vieme tak urobiť za pomoci `SetSoundVolume(hlasitost);`

```
int minimalna = 1 , maximalna = 4;
SetSoundVolume(minimalna);
PlayTone(frekvencia,trvanie);
SetSoundVolume(maximalna);
PlayTone(frekvencia,trvanie);
```

Pre okamžité zastavenie prehrávania sa dá použiť príkaz `StopSound()`;

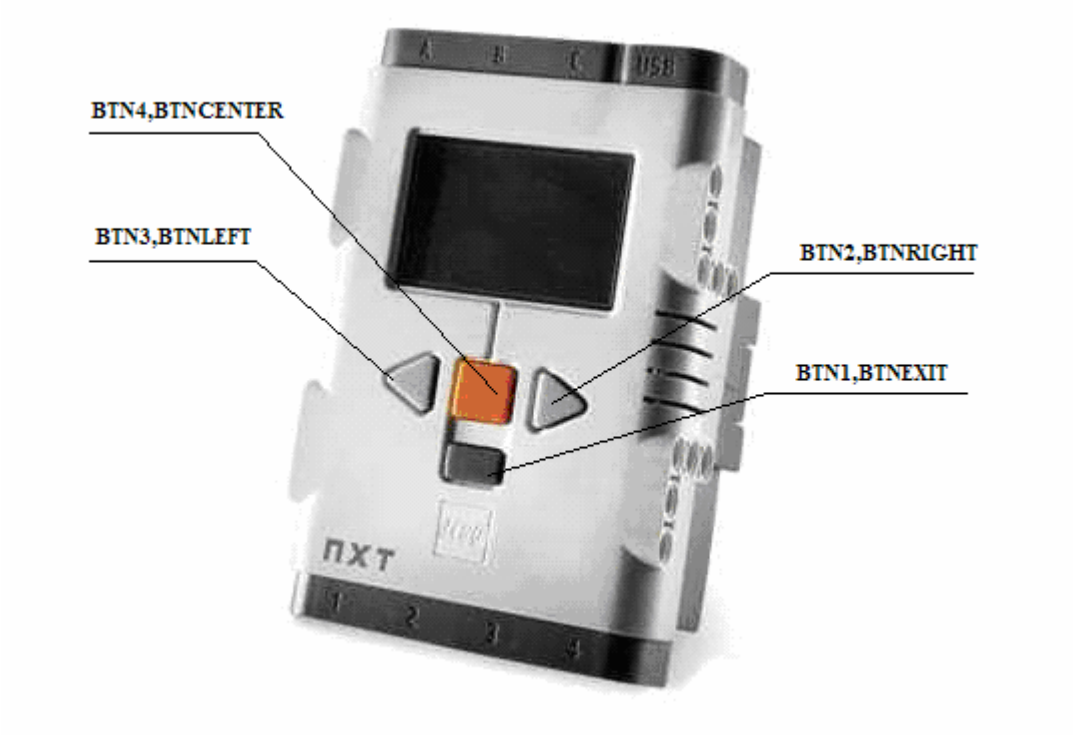
```
PlayTone(frekvencia,trvanie);// ton trva 1s
Wait(500);                    // caka sa 5s
StopSound();                 // vypina sa ton po 5s
```

Tlačidlá

Lego kocka NXT obsahuje štyri tlačidlá, ktoré môžu byť použité ako vstup do programu. Pre prácu s týmito tlačidlami sú zavedené konštanty, ktoré reprezentujú jednotlivé tlačidla v daných príkazoch. V nasledovnej tabuľke sú uvedené tieto konštanty:

Názov premennej	hodnota
BTN1,BTNEXIT	0
BTN2,BTNRIGHT	1
BTN3,BTNLEFT	2
BTN4,BTNCENTER	3
NO_OF_BTNS	4

Tabuľka 3. Tlačidlá



Obrázok 1. Kocka NXT

Ak chceme rátať koľko krát bolo stlačené nejaké tlačidlo, dá sa to uskutočniť pomocou `ButtonCount(tlacidlo,reset)`. Premenná `tlacidlo` reprezentuje hodnotu z tabuľky 3. Premenná `reset` je logického typu a pri jej nastavení na `true` je po odčítaní počítadlo vymazané.

```
int tlacidlo = 1;                /*zodpoveda pravemu
                                    volicu */
ButtonCount(tlacidlo,true);
```

Na zistenie, či tlačidlo je stlačené, sa používa príkaz `ButtonPressed(tlacidlo,reset)`. Parametre tejto funkcie sú zhodné s parametrami funkcie `ButtonCount(tlacidlo,reset)`.

```
ButtonPressed(tlacidlo,reset);
```

Stav kocky

Kocka NXT je prioritne napájaná z batérií. Preto pri písaní programov, ktoré pracujú dlhšiu dobu, je potrebné vedieť zistiť stav, v akom sa batérie nachádzajú. Pre zistenie napäťovej úrovne slúži príkaz `BatteryLevel()`. Ten vracia hodnotu v milivoltoch (tisícina volta).

```
int hodnota_napatia;
hodnota_napatia = BatteryLevel();
```

Snímače a motory

Motory

Kocka NXT obsahuje tri konektory, do ktorých je možné zapojiť súčasne tri motory. Motory obsahujú snímače, ktoré indikujú pootočenie. Motor môže byť použitý aj ako snímač pootočenia či polohy. Funkcie, ktoré pracujú s motormi, majú jeden z argumentov vyhradený pomenovaniu konektoru, na ktorom sú dané motory pripnuté. Tieto pomenovania sú zoradené v tabuľke 4.

Meno konektoru	Hodnota
OUT_A	0
OUT_B	1
OUT_C	2
OUT_AB	3
OUT_AC	4
OUT_BC	5
OUT_ABC	6

Tabuľka 4. Pomenovania konektorov

Na roztočenie motora špecifickým smerom slúži príkaz `OnFwd(meno_konektoru,sila).` Tento príkaz má dva parametre. Jeden určuje konektor a druhý určuje rýchlosť, akou sa má motor otáčať. Rýchlosť je definovaná ako sila a symbolizuje energiu dodávanú motoru. Je definovaná od 0 po 100. Ak zadáme väčšiu hodnotu ako 100, automaticky sa ohraničí na hodnotu 100. Rovnakým príkazom, ale s opačným pôsobením, je príkaz `OnRev(meno_konektoru,sila).`

```
OnFwd(OUT_A,100);
Wait(1000);        //motor A sa toci 1s dopredu
OnFwd(OUT_A,100);
Wait(1000);        //motor A sa toci 1s dozadu
OnFwd(OUT_AB,60);
Wait(1000)         //motory A a B 1S
```

Ak chceme zastaviť motor tak, aby sa ďalej netočil a okamžite zabrzdil, použijeme na to príkaz `Off(meno_konektora).`

```
OnFwd(OUT_A,100);
Wait(500);
Off(OUT_A);       //motor okamzite zastavi (brzda)
```

Na druhej strane, ak požadujeme, aby sa motor po vypnutí napájania ešte samovoľne dotočil, použijeme príkaz `Coast(meno_konektora)`

```
OnFwd(OUT_A,100);
Wait(500);
Coast(OUT_A);     //motor pozvolna zastavi
```

Existujú aplikácie, napríklad rameno podávača, kde je potrebné, aby motor vykonal presne špecifikovaný pohyb. Teda aby vykonal isté pootočenie. Pre tento účel je v jazyku NXT zavedená funkcia `RotateMotor(meno_konektoru,sila,uhol).` Motor pripojený na konektor sa bude otáčať rýchlosťou zodpovedajúcej zadanej sile o uhol.

```
RotateMotor(OUT_A,60,90);
```

Keď si aplikácia vyžaduje presnejšie polohovanie, môžeme použiť PID riadenie.
`RotateMotorPID(meno_konektoru,sila,uhol,p,i,d).` Príkaz má rovnaké vstupy ako `RotateMotor(meno_konektoru,sila,uhol),` až na

parametre p, i, d, ktoré reprezentujú jednotlivé zložky regulátora, a sú zodpovedné za jeho nastavenie.

```
RotateMotorPID(OUT_A,60,30,10,20,100);
```

V dôsledku toho, že motor je riadený silou, ktorá reprezentuje energiu napájania, a táto energia nemusí byť pre každý motor úplné rovnaká (stav jednotlivých článkov batérie je rôzny), je nutná pre aplikácie, kde sa vyžaduje súbežný chod motorov, použiť príkaz `OnFwdSync(meno_konektora,sila,rozdiel)`, `OnRevSync(meno_konektora,sila,rozdiel)`. Vstupný parameter `rozdiel` definuje rozdiel medzi jednotlivým motormi.

```
OnFwdSync(OUT_AB,75,-100);  /*robot by zatacal
                              vpravo */
OnRevSync(OUT_AB,75,-100);  /* robot by zatacal
                              vlavo*/
```

Ako bolo už vyššie uvedené, motor môže pracovať aj ako snímač otáčok. Táto funkcia je výhodná, či už v aplikácii, kde potrebujeme presne vedieť o koľko sa motor otočil, aby sme vedeli vyrátať vzdialenosť, ktorú vozidlo prešlo, alebo v aplikáciách, kde motor slúži ako snímač. Pre zistenie aktuálneho stavu počítadla otočenia motora je možné použiť príkaz `MotorTachoCount(meno_konektoru)`. Pre jeho vymazanie príkaz `ResetTachoCount(meno_konektoru)`

```
int pocidadlo;
ResetTachoCount(OUT_A);
OnFwd(OUT_A);
Wait(1000);
pocitadlo   =   MotorTachoCount(OUT_A);/*v   premennej
                                        pocitadlo   je
                                        ulozeny   udaj
                                        reprezentujuci
                                        pootocenie*/
```

Snímače

Kocka Lego NXT disponuje štyrmi konektormi určenými pre pripojenie snímačov. Snímače, ako aj motory, sú pripojené cez zbernicu I2C. Základná stavebnica Lego Mindstorm NXT obsahuje štyri snímače:

- Ultrasonický diaľkomer
- Dotykový snímač
- Svetelný snímač, svetelný diaľkomer
- Zvukový snímač

Práca a programovanie jednotlivých snímačov si vyžaduje iný prístup pre každý snímač. Preto v nasledujúcej časti upustíme od štandardného postupu vysvetľovania príkazov a nahradíme ho príkladmi, ktoré budú reprezentovať pracú s jednotlivými snímačmi. Konektory, na ktoré je možné pripojiť snímače, sú definované ako:

```
IN_1, IN_2, IN_4, IN_3, IN_4.
```

Ultrasonický diaľkomer
je snímač, ktorý meria vzdialenosť. Je založený na princípe merania času, za aký sa zvukový signál vyslaný k meranému objektu vráti späť do snímača. Tento časový udaj priamo reflektuje vzdialenosť, ktorú musel zvuk uraziť.

```
task main()
{
   int vzdialenost;
   SetSensorLowspeed(IN_4);  //digitalny snimac
   vzdialenost = SensorUS(IN_4);//vzdialenost v cm
}
```

V prvom kroku program nastaví konektor 4 do režimu Lowspeed, ktorý zodpovedá digitálnemu snímaču. Následne sa pomocou príkazu `SensorUS(IN_4)` zmeria vzdialenosť.

Dotykový snímač
je diskrétny snímač, ktorý ma dva stavy. Jeden zodpovedá jeho zatlačeniu a druhý zodpovedá jeho uvoľnenému stavu.

```
task main ()
{
    int stav_senozra;
    SetSensor(IN_1,SENSOR_TOUCH);
    stav_senozra = SENSOR_1;
}
```

Ilustračný program najprv pomocou príkazu `SetSensor(IN_1,SENSOERE_TOUCH)` nastaví premenné tak, že na konektore IN_1 sa nachádza dotykový senzor (`SENSOR_TOUCH`). Následne sa zaznamenáva aktuálny stav senzora do premennej `SENSOR_1,` ktorú sme už priradili k nami zadefinovanej premennej `stav_senzora.`

Svetelný senzor

meria úroveň prijímaného svetelného žiarenia, pričom obsahuje aj zdroj svetla, ktorý toto žiarenie vyžaruje. Na základe jeho intenzity po odraze je možné určovať vzdialenosť, prípadne farby objektu.

```
task main()
{
      int stav_senzora;
      SetSensorLight(IN_2);
      stav_senzora = Sensor(IN_2);
}
```

Príkaz `SetSensorLight(IN_2)` nastaví zariadenie tak, že na konektor IN_2 je pripnutí svetelný senzor. Následne funkciou `Sensor(IN_2),` odčítame hodnotu na snímači a zapíšeme do premennej `stav_senzora`.

Zvukový senzor

je senzor, ktorý sníma intenzitu zvuku. Je možné ho použiť na aplikácie, pri ktorých je potrebné reagovať na zvukové vnemy okolia.

```
task main ()
{
    int stav_senzora;
    SetSensorSound(IN_3);
    stav_senzora = SENSOR_3;
}
```

Príkaz `SetSenosrSound(IN_3)` nastaví konštanty zariadenia tak, aby akceptovali zvukový snímač na konektore `IN_3`. Následne je meraná hodnota ukladaná do premennej `SENSOR_3,` ktorej hodnota sa priradí do premennej `stav_senzora.`