# Programmation NQC

## Application à la commande d'un robot RCX 2.0 Lego®

Jean-Philippe Babau

Département Informatique, laboratoire CITI
INSA Lyon

jean-philippe.babau@insa-lyon.fr

Département Informatique INSA de Lyon

## Plan

- Caractéristiques du robot RCX 2.0 de Lego®
- Programmation du robot RCX 2.0
  - Gestion des entrées/sorties
- Not Quite C : pas exactement du C
  - Langage de programmation proche du langage C
  - Programmation des briques RCX de Lego ®
- Programmation multi-tâches en NQC
  - Temps, tâches et synchronisation

jean-philippe.babau@insa-lyon.fr

## Objectifs du cours

- Commande d'un robot RCX 2.0 (Lego Mindstorms ® )
  - trois capteurs (2 capteurs de contact, un capteur de luminosité)
  - trois actionneurs (2 moteurs)
  - un générateur de son
  - un écran LCD
  - un medium infra-rouge
- Structure d'un programme NQC
  - main, données, algorithmes, fonctions, subroutines
- Structure d'un programme multi-tâches NQC
  - tâches, synchronisation

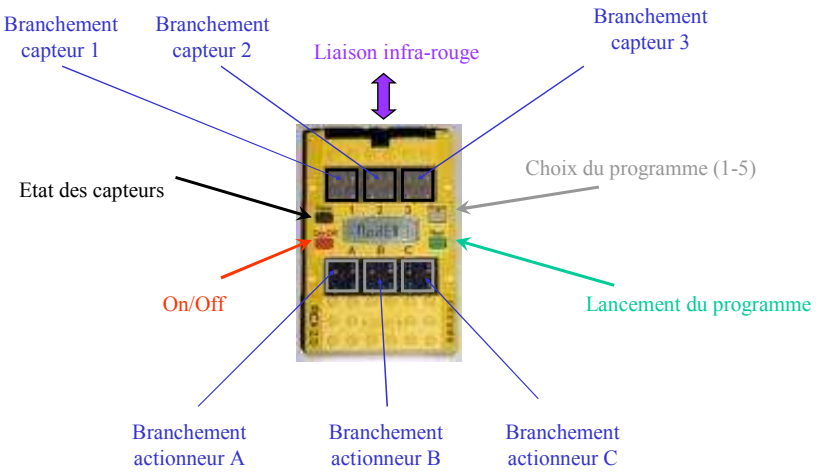




## Plan

- Caractéristiques du robot RCX 2.0 de Lego®
- Programmation du robot RCX 2.0
  - Gestion des entrées/sorties
- Not Quite C : pas exactement du C
  - Langage de programmation proche du langage C
  - Programmation des briques RCX de Lego ®
- Programmation multi-tâches en NQC
  - Temps, tâches et synchronisation

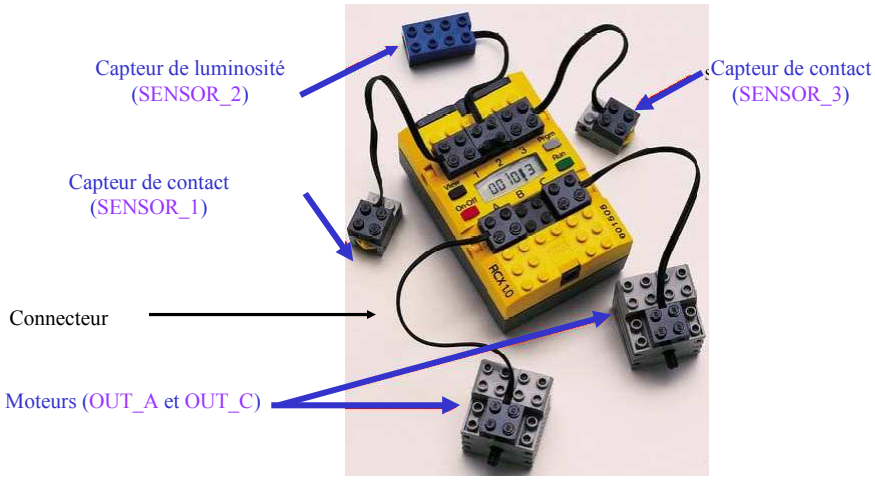
# RCX 2.0




jean-philippe.babau@insa-lyon.fr



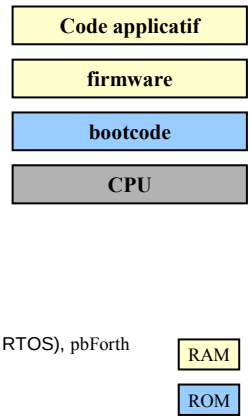
# RCX 2.0 avec ses capteurs et ses actionneurs




jean-philippe.babau@insa-lyon.fr

# Architecture matérielle et logicielle

- Matériel (CPU)
  - micro-contrôleur Hitachi H8300
    - 16 Mhz, 16 ko ROM, 32 ko de RAM
    - 6 ko de RAM disponible pour l'application
- Logiciels
  - Bootcode
    - Initialisation de la carte
    - Gestion des entrées, sorties
    - Communication IR
  - Firmware
    - API du lego
    - Services d'accès aux entrées et aux sorties
    - Exécutif multitâches
    - Par défaut : code interprété (fichier FIRM0328.LGO)
    - Autres firmwares : leJOS(JAVA), BrickOS ou legOS(C/C++ et RTOS), pbForth
- Mémoires
  - ROM : mémoire persistante
  - RAM : mémoire volatile (persistante si alimentée)




jean-philippe.babau@insa-lyon.fr



# Programmation

- Développement croisé
  - Machine hôte (PC, Mac) : développement et mise au point
  - Machine cible (RCX 2.0) : exécution d'un programme
  - Liaison infra-rouge : téléchargement et communications
- Environnement de développement (*Bricx Command Center* ou *BricxCC*)
  - Éditeur de langage NQC
  - Compilation (*Compile*) et téléchargement (*Download*)
  - Suivi des variables (*Watch*)
  - Diagnostic de la carte
  - C:\Program Files\BricxCC\BricxCC.exe
  - Plus d'informations sur http://bricxcc.sourceforge.net/
- Simulation
  - Simulateur web SIMBOT
  - Émulateur (emulegOS basé sur legOS)

jean-philippe.babau@insa-lyon.fr

## Quelques problèmes …

- Le robot n'est pas visible par le logiciel
  - Penser à utiliser le bon port USB
  - Allumer le robot !
- Plus de piles
  - Penser à re-télécharger le firmware
  - Program Files\Lego Mindstorms\meca\FIRM0328.LGO
- Lumière artificielle
  - Interférences avec la liaison IR
- Téléphones, PDA, …
  - Interférences avec la liaison IR
- Interférences entre la liaison IR et le capteur de luminosité
  - Placer le capteur à l'opposé de la zone de communication





## Plan

- Caractéristiques du robot RCX 2.0 de Lego®
- Programmation du robot RCX 2.0
  - Gestion des entrées/sorties
- Not Quite C : pas exactement du C
  - Langage de programmation proche du langage C
  - Programmation des briques RCX de Lego ®
- Programmation multi-tâches en NQC
  - Temps, tâches et synchronisation

# Les capteurs : configuration

- 3 capteurs (numérotés 0,1 et 2)
  **SENSOR_1**, **SENSOR_2**, **SENSOR_3**

- Un type
  - SENSOR_TYPE_NONE — capteur générique passif
  - SENSOR_TYPE_TOUCH — capteur de contact
  - SENSOR_TYPE_TEMPERATURE — capteur de température
  - SENSOR_TYPE_LIGHT — capteur de luminosité
  - SENSOR_TYPE_ROTATION — capteur de rotation

- Un mode
  - SENSOR_MODE_RAW — intervalle entre 0 et 1023
  - SENSOR_MODE_BOOL — booléen (0 ou 1)
  - SENSOR_MODE_EDGE — nombre de tranistions booléennes
  - SENSOR_MODE_PULSE — nombre de périodes booléennes
  - SENSOR_MODE_PERCENT — valeur entre 0 et 100
  - SENSOR_MODE_FAHRENHEIT — degrés Fahrenheit
  - SENSOR_MODE_CELSIUS — degrés Celsius
  - SENSOR_MODE_ROTATION — rotation (16 ticks par tour)

jean-philippe.babau@insa-lyon.fr

Département Informatique INSA de Lyon

# Les capteurs : configuration

- Une configuration : un type + un mode

- **Exemples de Sensor Configuration** — **Type** — **Mode**
  - SENSOR_TOUCH — SENSOR_TYPE_TOUCH — SENSOR_MODE_BOOL
  - SENSOR_LIGHT — SENSOR_TYPE_LIGHT — SENSOR_MODE_PERCENT
  - SENSOR_CELSIUS — SENSOR_TYPE_TEMPERATURE — SENSOR_MODE_CELSIUS
  - SENSOR_FAHRENHEIT — SENSOR_TYPE_TEMPERATURE — SENSOR_MODE_FAHRENHEIT
  - SENSOR_PULSE — SENSOR_TYPE_TOUCH — SENSOR_MODE_PULSE

jean-philippe.babau@insa-lyon.fr

# Les capteurs : les actions

- Initialiser le type, le mode, la configuration
  - **SetSensorType(sensor, type);** SetSensorType(SENSOR_1, SENSOR_TYPE_TOUCH);
  - **SetSensorMode(sensor, mode);** SetSensorMode(SENSOR_1, SENSOR_MODE_RAW);
  - **SetSensor(sensor, configuration);** SetSensor(SENSOR_1, SENSOR_TOUCH);
- Récupérer une information d'un capteur
  - **x = SENSOR_1;** // lit sensor_1 et sauvegarde la valeur dans x
  - **SensorValue(n);** x = SensorValue(0); // lit sensor 1
- Remettre à 0 un capteur cumulatif
  - **ClearSensor(sensor);** ClearSensor(SENSOR_1);

jean-philippe.babau@insa-lyon.fr



# Les actionneurs : configuration

- 3 actionneurs (numérotés 0,1 et 2)
  **OUT_A**, **OUT_B**, **OUT_C**
  **OUT-A+OUT_B**
- Un Mode
  - OUT_OFF actionneur arrêté
  - OUT_ON actionneur commandable
  - OUT_FLOAT
- Une Direction (actif si OUT_ON)
  - OUT_FWD en avant
  - OUT_REV en arrière
  - OUT_TOGGLE changement de direction (avant ->arrière ou arrière->avant)
- Un niveau (POWER), (actif si OUT_ON)
  - 0 (OUT_LOW) à 7 (OUT_FULL)
  - OUT_HALF
- Par défaut
  - OUT_FULL, OUT_FWD

jean-philippe.babau@insa-lyon.fr

# Les actionneurs : les actions

- Configuration
  - **SetDirection(outputs, direction) ;** SetDirection(OUT_A, OUT_FWD); // OUT_A en avant
- Mise au point du niveau
  - **SetPower(outputs, power);** SetPower(OUT_A, OUT_FULL); // OUT_A à 7
  - **OutputStatus(n)** ; x = OutputStatus(0); // x contient le niveau de OUT_A
- Arrêt / marche
  - **SetOutput(outputs, mode);** SetOutput(OUT_A, OUT_ON); // active OUT_A
- Appels de haut niveau
  - **On(outputs)**; **Off(outputs) ;** On(OUT_A); // active OUT_A
  - **Fwd(outputs)** ; **Rev(outputs) ; Toggle(outputs) ;**
  - **OnFwd(outputs); OnRev(outputs) ;** OnFwd(OUT_A); // active OUT_A en avant et au niveau 7
  - **OnFor(outputs, time) ;** // time en unités de 10 ms
  - OnFor(OUT_A, 100) // fait avancer OUT_A pendant 100 x 10 ms

jean-philippe.babau@insa-lyon.fr



# Le son

- 1 générateur de son
- Faire un son
  - **PlaySound(sound);**
  - sound : SOUND_CLICK, SOUND_DOUBLE_BEEP, …
- Une fréquence
  - **PlayTone(frequency, duration);** // duration en unités de 1 ms

  PlayTone(440, 50); // joue do durant 50/100ème de seconde
- Arreter/reprendre
  - **MuteSound(); UnmuteSound();**

jean-philippe.babau@insa-lyon.fr

# L'affichage via l'écran LCD

- 1 mode
  - DISPLAY_WATCH affiche l'heure (par défaut)
  - DISPLAY_SENSOR_1 affiche la valeur du capteur 1
  - DISPLAY_SENSOR_2 affiche la valeur du capteur 2
  - DISPLAY_SENSOR_3 affiche la valeur du capteur 3
  - DISPLAY_OUT_A affiche les paramètres de l'actioneur A
  - DISPLAY_OUT_B affiche les paramètres de l'actioneur B
  - DISPLAY_OUT_C affiche les paramètres de l'actioneur C
  - DISPLAY_USER mode programmable
- Configurer un mode
  - **SelectDisplay(mode);**
- Afficher une valeur (en mode DISPLAY_USER)
  - **SetUserDisplay(value, precision);**
  - **SetUserDisplay(1234, 2); // affiche 12.34**
  - **SetUserDisplay(1234, 0); // affiche 1234**

jean-philippe.babau@insa-lyon.fr



# Le service de trace

- Datalog
- Configurer la zone de sauvegarde
  - **CreateDatalog(100); // zone de stockage de 100 valeurs**
  - On peut aller jusqu'à 1000 données stockées (attention à la politique de stockage)
- Sauvegarde
  - **AddToDatalog(value);**
  - **AddToDatalog(x); // sauvegarde de x**
  - **AddToDatalog(Timer(0)); // sauvegarde de la valeur courante du timer 0**
  - **AddToDatalog(SENSOR_1); // sauvegarde de la valeur courante du capteur 0**
- Récupération des données sur la machine de développement
  - Cf. logiciel BricxCC
    - Histogramme temporel
    - Extraction d'un tableau de valeurs pour tableur

jean-philippe.babau@insa-lyon.fr

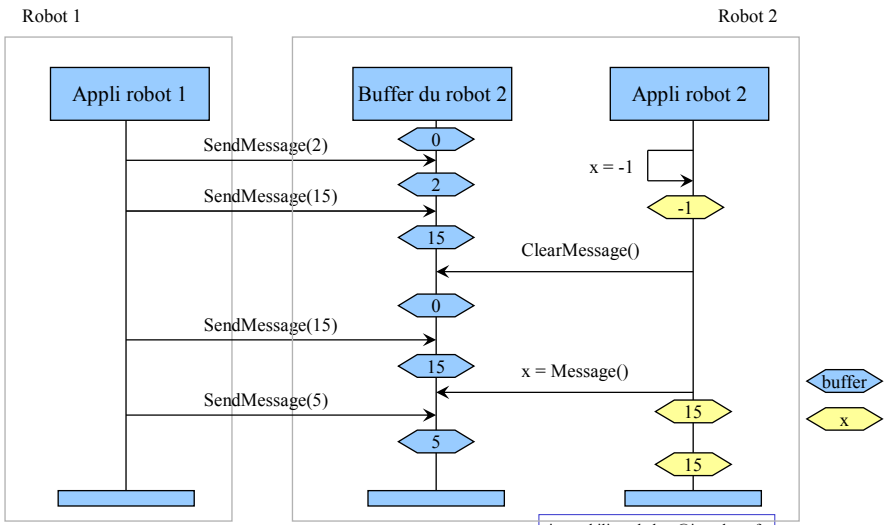
# Le service de communication

- Messagerie
  - Communication d'entiers entre 0 et 255
  - 0 réservé pour détecter l'absence de nouveau message
- Emission
  - **SendMessage(value);**
  - **SendMessage(3); // envoie de 3**
  - **SendMessage(259); // envoie de 3**
- Réception
  - **x = Message();**
- Effacement du buffer de réception
  - **ClearMessage();**
- Exemple
  - **ClearMessage();**
  - **until (Message() > 0); // attente prochain message (différent de 0)**

jean-philippe.babau@insa-lyon.fr



# Exemple de communications




jean-philippe.babau@insa-lyon.fr

# Autres fonctions

- Générateur aléatoire
  - **Random(n); // renvoie une valeur entre 0 et n**
  - **n = Random(10); // renvoie une valeur entre 0 et 10**
- Niveau de batterie
  - **BatteryLevel(); // valeur en** millivolts
  - **x = BatteryLevel();**
- Autres services
  - SelectProgram(3); // lancement du 4ème programme
  - SetWatch(16,0); // mise à l'heure 16h00
  - h = Watch(); // récupération de l'heure en minutes

jean-philippe.babau@insa-lyon.fr

Département Informatique INSA de Lyon

# Programme de commandes

- Une tâche principale
  - task main
  - Séquence de commandes

```
task main ()
{
    /* séquence de commandes */
    PlaySound(SOUND_CLICK);
}
```

jean-philippe.babau@insa-lyon.fr

## Plan

- Caractéristiques du robot RCX 2.0 de Lego®
- Programmation du robot RCX 2.0
  - Gestion des entrées/sorties
- Not Quite C : pas exactement du C
  - Langage de programmation proche du langage C
  - Programmation des briques RCX de Lego ®
- Programmation multi-tâches en NQC
  - Temps, tâches et synchronisation

jean-philippe.babau@insa-lyon.fr



## Programme NQC

- Déclarations de données
  - Initialisation des données
  - Définition de constantes
- Tâche : séquences d'instructions
  - Commandes robot
  - Opérations sur des données
  - Appels de fonctions
  - Structures algorithmiques

jean-philippe.babau@insa-lyon.fr

# Introduction à NQC

- Not Quite C : c'est presque du C
-
- Langage de programmation proche du langage C
  - Syntaxe C-like
- Programmation des briques RCX de Lego ®
  - Gestion des entrées / sorties
- Programmation multi-tâches

jean-philippe.babau@insa-lyon.fr



# Eléments syntaxiques

- Différence entre les minuscules et les majuscules
  - max et Max sont différents
- Liste de mots clés réservés
  - Cf. suite
- Identificateur
  - suite de lettres, chiffres, _ ne commençant pas par un chiffre

| | | | | |
|---|---|---|---|---|
| Ne_le | x1 | _12 | IF | : ok |
| 2_A 2000 | 3A | | | : non ok |

- Commentaires
  - /* commentaire a la C */
  - // commentaire a la C++

jean-philippe.babau@insa-lyon.fr

# Les données

- Tout information manipulée par le programme
  - Constantes, variables
- Caractérisé par
  - un identificateur : nom pertinent
  - **un commentaire**
- Deux types
  - entier signé sur 16 bits (2 octets, [-32768,32767]) : **int**
  - pointeur d'entier : **int ***
- Tableaux
  - Tableaux d'entiers
  - Doit être initialisé explicitement
  - Ne peux pas être passé en argument d'une fonction





# Les données

- Valeurs
  - décimal : 126 - 33
  - hexadécimal : 0x10
  - Booléen : 0 correspond à false, différent de 0 correspond à true
- Déclaration de données

```
int nombreTour; // nombre de tours effectués
int missionTerminee = 0;
int a, b ;
```

- Déclaration de *constantes* (macro subsitution)

```
#define Annee 2007 // année courante à modifier selon la date
```

- Déclaration de tableaux

```
int lesPositions[10]; // ensemble des positions à atteindre
```

Déclaration de lesPositions[0], …, lesPositions[9]

# Liste des mots clés

| | | | | | |
|---|---|---|---|---|---|
| **__event_src** | | **__res** | | **__taskid** | **abs** |
| **__nolist** | **__sensor** | **__type** | **acquire** | | **asm** |
| **do** | | **int** | **sub** | | **break** |
| **else** | | **monitor** | **switch** | | |
| **case** | | **false** | **repeat** | | **task** |
| **catch** | | **for** | **return** | | **true** |
| **const** | | **goto** | **sign** | | **void** |
| **continue** | | **if** | **start** | | **while** |
| **default** | **inline** | **stop** | | | |





# Les opérateurs

- Liste d'opérateurs

  \+ - * / (division entière) % (modulo)
  ++ (incrémentation) -- (décrémentation)
  = (affectation)
  == (comparaison) != < > <= >=
  ! (négation logique) && (et logique) || (ou logique)
  <<, >> (décalages) & (et bit à bit) | (ou bit à bit)

- exemples

```
5 / 4                    5 % 4                    32767 + 1

5 = = 4                  7 <= 19

! 5                      5 && 2
5 & 2                    3 || 0

int a = 5 ; a++ ;        int b ; b = a++; a = ++ b;

a += 4;                  a <<=2 ;
```

## Priorité des opérateurs

- ordre, sens
  - 3 || 0 && 1  5 + 4 % 3  a + b * c
- parenthésage
  - Pas d'ambiguïté
  - Aide à la relecture du code





## Les structures algorithmiques

- condition
  - alternative : **if (** *expression* **)** *instructions* [ **else** *instructions* ]
  - cas_parmi : **switch ( ***exp_entiere* **)**
    **{ case** *valeur1* **:** *suite_instructions* **; break;**
    **case** *valeur2* **:** *suite_instructions* **; break ;**
    **default :** *suite_instructions* **; }**
- répétition
  - tant_que : **while (***expression* **)** *instructions*
  - Jusqu'à : **do** *instructions* **while (***expression* **) ;**
  - Répéter plusieurs fois : **repeat (***expression* **)** *instructions*
- instruction
  - *instructions* **:** *instruction_simple* | *bloc*
  - *bloc* : **{** *suite_instructions* **}**
  - *instruction_simple* : *instruction* **;**
  - *suite_instructions : instruction_simple suite_instructions*

# La structure alternative

- entier # booléen : 0 : faux, différent de 0 : vrai
- la condition est une expression

```
#define dureeTour 400 // durée de rotation

if (SENSOR_1)
// le capteur est appuyé
{
      OnFwd(OUT_A); // tourne à droite
      Wait(dureeTour );
      Off(OUT_A); // arrêt
}
```

```
if (SENSOR_1)
// le capteur est appuyé
{
    OnFwd(OUT_A); // tourne à droite
    Wait(dureeTour);
    Off(OUT_A); // arrêt
}
else
// le capteur est relâché
{
    OnFwd(OUT_C); // tourne à gauche
    Wait(dureeTour);
    Off(OUT_C); // arrêt
}
```

jean-philippe.babau@insa-lyon.fr



# La structure cas parmi

- pour éviter des conditions en cascade
- l'ordre des valeurs est sans importance, mais les valeurs sont différentes

```
int nb ; // …

switch  (nb)
{
      case 1 : OnFwd(OUT_A);
               SelectDisplay(DISPLAY_OUT_A);
               Wait(dureeMarche);
               Off(OUT_A);
               break;
      case 3 : OnFwd(OUT_C);
               SelectDisplay(DISPLAY_OUT_C);
               Wait(dureeMarche);
               Off(OUT_C);
               break ;
      default : SelectDisplay(DISPLAY_USER);
                SetUserDisplay(0, 2);
}
```

jean-philippe.babau@insa-lyon.fr

## La structure tant que

- pas de booléen, 0 : faux, différent de 0 : vrai
- la condition est une expression
- attention aux { }

```
#define dureeAction 100 // durée pour une action d'avancement
#define nbActionsMax 5 // nombre d'actions à faire
int nbActions; // nombre d'actions en cours

// intialisation des données pour l'expression
nbActions = 0;

while ( nbActions < nbActionsMax)
{
        OnFwd(OUT_A+OUT_C);
        Wait(dureeAction);
        Off(OUT_A+OUT_C);
        nbActions ++;
}
/* il faut s'assurer  que l'action modifie les données
afin que l'expression soit fausse à  un moment donné */
```

jean-philippe.babau@insa-lyon.fr



## Les autres formes de structures de boucle tant que

- Tâches répétitives
  - tâches de contrôle, de suivi

```
while ( SENSOR_1 != valeurRecherchee )
{
        /* traitements */
}
```

- Boucles infinies
  - tâches de contrôle, de suivi

```
while ( 1 )
{
        /* attente ou vérification */

        /* traitements */
}
```

jean-philippe.babau@insa-lyon.fr

# La structure repeat

- Nombre d'itérations connues

```
int nbActionsAFaire = 5; // nombre d'actions à faire

repeat(nbActionsAFaire) // repeat(expression)
{
        OnFwd(OUT_A+OUT_C);
        Wait(dureeAction);
        Off(OUT_A+OUT_C);
        /* il existe un programme équivalent avec un seul appel */
}
```

jean-philippe.babau@insa-lyon.fr



# Les fonctions NQC

- Fonctions « inline »
  - Un appel entraîne la recopie du code de la fonction
  - Attention à la taille finale du code !
- Entrées
  - Par valeur : int
  - Par valeur constante : const int
  - Par référence : int&
  - Par référence, valeur constante : const int &
  - Par pointeur : int *
  - Par pointeur constant : const int *
- Pas de retour en NQC : **void**
- Syntaxe

```
void name(liste_des_arguments)
{
    // instructions
}
```

jean-philippe.babau@insa-lyon.fr

# Les fonctions NQC

Passage par valeur : **int**

```
void foo(int x)
{
        x = 2;
}

task main()
{
        int y = 1;          // y est égal à 1
        foo(y);             // y reste égal à 1
}
```





# Les fonctions NQC

Passage par valeur constante : **const int**

```
void foo(const int x)
{
        PlaySound(x);       // ok
        x = 1;              // erreur car x ne peut pas être modifié (const)
}

task main()
{
        foo(2);             // ok
        foo(4*5);           // ok - l'expression est une constante
        foo(x);             // erreur, x n'est pas une constante
}
```

# Les fonctions NQC

Passage par référence : **int &**

```
void foo(int & x)
{
        x = 2;
}

task main()
{
        int y = 1;     // y est égal à 1
        foo(y);        // y est maintenant égal à 2
        foo(2);        // erreur,  seules les variables ont des références
}
```





# Les subroutines NQC

- Fonctions
  - Un appel : pas de recopie du code de la subroutine
  - Non réentrantes
- Restrictions
  - 8 subroutines au maximum
  - Pas d'arguments
  - Pas d'appel à une autre subroutine
- Syntaxe

```
sub name()
{
    // instructions

    // pas d'appel à une autre subroutine
}
```

# Les directives de compilation

- **#define**
  - Macro substitution avant compilation
  - Définition d'une variable de compilation
  - **#undef** : fin de la définition, fin de la macro substitution
- **#include**
  - Inclusion de fichier

```
#include "foo.nqh" // ok
#include <foo.nqh> // erreur en NQC
```

- **#if, #ifdef, #ifndef, #else, #endif**
  - Directives de compilation
- **#pragma reserve startWord endWord**
  - Réservation d'espace mémoire
  - L'utilisation de 3 compteurs (counter) nécessite l'ajout de la directive :

```
#pragma reserve 0 1 2
```

jean-philippe.babau@insa-lyon.fr



# Inclusion multiples de fichiers

- Directive pour une seule inclusion

```
#ifndef FILENAME
#define FILENAME

    #include " fichiersUtilises "

    // instructions NQC

#endif
```

jean-philippe.babau@insa-lyon.fr

# Plan

- Caractéristiques du robot RCX 2.0 de Lego®
- Programmation du robot RCX 2.0
  - Gestion des entrées/sorties
- Not Quite C : pas exactement du C
  - Langage de programmation proche du langage C
  - Programmation des briques RCX de Lego ®
- Programmation multi-tâches en NQC
  - Temps, tâches et synchronisation

jean-philippe.babau@insa-lyon.fr



# Gestion du temps

- Caractéristiques des timers
  - Le nombre de timers dépend de la cible
  - Précision : 100 ms (10 ticks par seconde)
    - de 0 à 32767 ticks : environ 55 minutes
- Action sur les timers
  - **ClearTimer(n)** // reset du timer n à 0
  - **Timer(n);** **x = Timer(0); // retour de la valeur (en 10ème de secondes,100ms)**
- Attente
  - Wait(nombreDeTicks); // **en 100ème de secondes, 10 ms**
  - Wait(100); // attente d'une seconde

jean-philippe.babau@insa-lyon.fr

# Les tâches

- Toujours une tâche principale
  - task main
  - Lancée au début de l'application
- Nombre de tâches limité par la plateforme (10)

jean-philippe.babau@insa-lyon.fr



# Gestion des tâches

- Lancement/arrêt de tâches
  - start taskName;
  - stop taskName;

```
task main ()
{
    SetSensor(SENSOR_1,SENSOR_TOUCH);
    start controlRobot;
    start readSensor;
}

task readSensor()
{
    while( 1 )
    {
        if ( SENSOR_1 == 1 )
        {
            stop controlRobot;
        }
    }
}
```

```
task controlRobot()
{
    …
}
```

jean-philippe.babau@insa-lyon.fr

# Tâche périodique

- Maîtrise des dates d'activation des actions
  - Scrutation périodique
  - Commande régulière
- Limitation de l'occupation du processeur

```
#define PERIODE 100 // période de scrutation égale à 1 seconde

task readSensor()
{
    while( 1 )
    {
        Wait(PERIODE); // mise en sommeil de la tâche pour PERIODE * 10 ms

        position = SENSOR_1 ; // acquisition
        …
    }
}
```

jean-philippe.babau@insa-lyon.fr



# Partage de données

- Partage d'une variable entière
  - Déclaration globale
- Appels atomiques
  - Uniquement des opérations d'affectation /consultation sur un entier

```
int x ; // variable partagée entre T1 et T2

task T1()
{
   ...
   x = a - b; // opération non atomique
   ....
}

task T2()
{
   ...
   y = x ; // la valeur de x n'est pas garantie
   ....
}
```

```
int x ; // variable partagée entre T1 et T2

task T1()
{ int x1; // variable locale à T1
   ...
   x1 = a - b; // opération locale
   x = x1; // opération atomique
   ....
}

task T2()
{
   ...
   y = x ; // la valeur de x est garantie
   ....
}
```

jean-philippe.babau@insa-lyon.fr

# Problème de synchronisation : redémarrage d'une tâche

```
task readSensor()
{
    int running = 1;
    start controlRobot;
    while( 1 )
    {
        if ( (SENSOR_1 == 1) && (running == 1) )
        {
            running = 0;
            stop controlRobot;
            Wait(200);
        }
        if ( (SENSOR_1 == 1) && (running == 0) )
        {
            running = 1;
            start controlRobot;
            Wait(200);
        }
    }
}
```

```
task main ()
{
    SetSensor(SENSOR_1,SENSOR_TOUCH);
    start readSensor;
}

task controlRobot()
{
    …
}
```

relance la tâche au début

jean-philippe.babau@insa-lyon.fr



# Synchronisation par sémaphore

- Sémaphore : variable entière
  int semTrigger = 0 ; // semaphore de synchronisation
  int semMutex = 1 ; // semaphore de protection
- Déclenchement d'événement
  semTrigger = 1 ;
- Attente d'évenement (non atomique !)
  until(semTrigger = 1) ; semTrigger = 0 ;
- Prise de sémaphore (non atomique!) / … / libération de sémaphore
  until(semMutex = 1) ; semMutex = 0 ; … semMutex = 1 ;

Remarque sur until : #define until(c) while(!(c))

jean-philippe.babau@insa-lyon.fr

# Contrôle d'une tâche

```
task readSensor()
{
    start controlRobot;
    Wait(200);
    running = 1;
    while( 1 )
    {
        if ( (SENSOR_1 == 1) && (running == 1) )
        {
            running = 0;
            Wait(200);
        }
        if ( (SENSOR_1 == 1) && (running == 0) )
        {
            running = 1;
            Wait(200);
        }
    }
}
```

```
int running = 0 ;
task main ()
{
    SetSensor(SENSOR_1,SENSOR_TOUCH);
    start readSensor;
}

task controlRobot()
{
    while(1)
    {
        until(running = 1) ;
        … // actions 1

        until(running = 1) ;
        … // actions 2

        until(running = 1) ;
        …// actions 3
    }
}
```

jean-philippe.babau@insa-lyon.fr



# Les événements : configuration

- 16 événements configurables
  - Numérotés de **0** à **15**
- Configuration : source de l'événement + type d'événement
  **SetEvent(numéro, source, type);**
- Annulation d'un événement
  **ClearEvent(numero); ClearAllEvents();**
- Définition d'intervalles
  **SetLowerLimit(numero, low_limit);**
  **SetUpperLimit(numero, upper_limit);**
- Sources de l'événement
  - Capteur, message buffer, timer, counter
- Types d'événement : critère d'émission de l'événement
  - Capteurs : EVENT_TYPE_PRESSED, EVENT_TYPE_RELEASED, …
  - Intervalles : EVENT_TYPE_LOW, EVENT_TYPE_NORMAL, EVENT_TYPE_HIGH
  - Message : EVENT_TYPE_MESSAGE

jean-philippe.babau@insa-lyon.fr

# Les événements : configuration

- Exemples
- Evénement capteur

```
#define EVENT_A 0
SetSensor(SENSOR_1, SENSOR_TOUCH);
SetEvent(EVENT_A, SENSOR_1, EVENT_TYPE_PRESSED);
```

- Événement timer
  - Toutes les secondes

```
#define EVENT_B 1

ClearTimer(0);
SetEvent(EVENT_B, Timer(0), EVENT_TYPE_HIGH);
SetUpperLimit(EVENT_B, 10);
```

jean-philippe.babau@insa-lyon.fr



# Les événements : les actions

- Suivi

```
monitor(masque)
    // instructions
catch(masque)
    // instructions
```

```
monitor( EVENT_MASK(EVENT_A) | EVENT_MASK(EVENT_B) | EVENT_MASK(4) )
{
        Wait(1000);
}
catch( EVENT_MASK(EVENT_B) )
{
        PlaySound(SOUND_DOWN);          // l'événement event_B arrive
        ClearTimer(0);                  // remise à 0 du timer 0
}
catch
{
        PlaySound(SOUND_UP) ;           // l'événement event_A ou l'événement 4 arrive
}
```

jean-philippe.babau@insa-lyon.fr

## Les événements : les actions

- Exemple

```
#define EVENT_A 0

task main()
{
    SetSensor(SENSOR_1, SENSOR_TOUCH);
    SetEvent(EVENT_A, SENSOR_1, EVENT_TYPE_PRESSED);

    while(1)
    {
        monitor(EVENT_MASK(EVENT_A))
        {
            while(1); ou while(1) Wait(100);
        }

        catch(EVENT_MASK(EVENT_A))
        {
            PlaySound(SOUND_CLICK);
        }
    }
}
```

jean-philippe.babau@insa-lyon.fr



## Ordonnancement des tâches

- Non prédictible
  - pas de priorités
- Comportement du moteur A ? 
- Attention à ne pas multiplier les tâches
  - Partage et cohérence de données
  - Synchronisation des actions

```
int go = 1 ; // déclenchement des moteurs

task T1()
{
  if (go = = 1)
  {
    OnFwd(OUT_A);
  }
}

task T2()
{
  if (go = = 0)
  {  Off(OUT_A);
  }
}

task T3()
{
  go = 0 ;
}
```

jean-philippe.babau@insa-lyon.fr

## Mise en place de tâches

- Conception
  - Tâche de démarrage
    - Initialisations et activations des autres tâches
  - Tâches de réaction à un événement externe ou un événement programmé
    - Un nouveau message
    - Une nouvelle mesure (capteurs)
    - Une alarme programmée sur un dépassement de timeout
  - Tâches régulières
    - Scrutation de capteurs
    - Gestion des actionneurs
    - Suivi du fonctionnement (*monitoring*)
- Programmation
  - Main
  - Tâches périodiques
    - while(1) { Wait(Periode); *Actions* } ou while(1) {*Actions* Wait(Periode);}
  - Tâches « en continu »
    - while(1) { *Actions* }
  - Tâches activées par une autre tâche
    - start *taskName*;
  - Programmation événementielle
    - Un événement -> une tâche

jean-philippe.babau@insa-lyon.fr



## Conclusion

- Robot
  - Langage de programmation de haut niveau
    - Langage interprété
  - Capteurs, moteurs, écran LCD, son, mémoire de stockage, batterie, communication IR
    - Capteurs : l'utilisation dépend de la configuration
    - Moteurs : commande de « -7 » à « 7 »
    - Mémoire limitée à 1000 entiers
    - Communication série half duplex, une trame = un entier
- Programmation
  - NQC
    - Limité aux entiers, pas de fonctions classiques (plutôt des macros)
  - Présentation du code
    - Commentaires, indentation, choix des noms de variables, définition des constantes
- Multitâches
  - Précision du temps : 10ms
  - Mise en place des tâches (10 maximum)
    - main, tâches périodiques, tâches continues
  - Attention au partage des données et à la synchronisation

jean-philippe.babau@insa-lyon.fr

# Références utilisées pour faire ce cours

« NQC Programmer's Guide, Version 3.1 r5 » by Dave Baum & John Hansen
http://bricxcc.sourceforge.net/nqc/

« Programming Lego Robots » by Mark Overmars
http://people.cs.uu.nl/markov/lego/

« Introduction to Computers and Programming » by Prof. I. K. Lundqvist
www.mit.edu/~kristina/Lego/UnifiedF05/Lectures/Lecture1.pdf

jean-philippe.babau@insa-lyon.fr