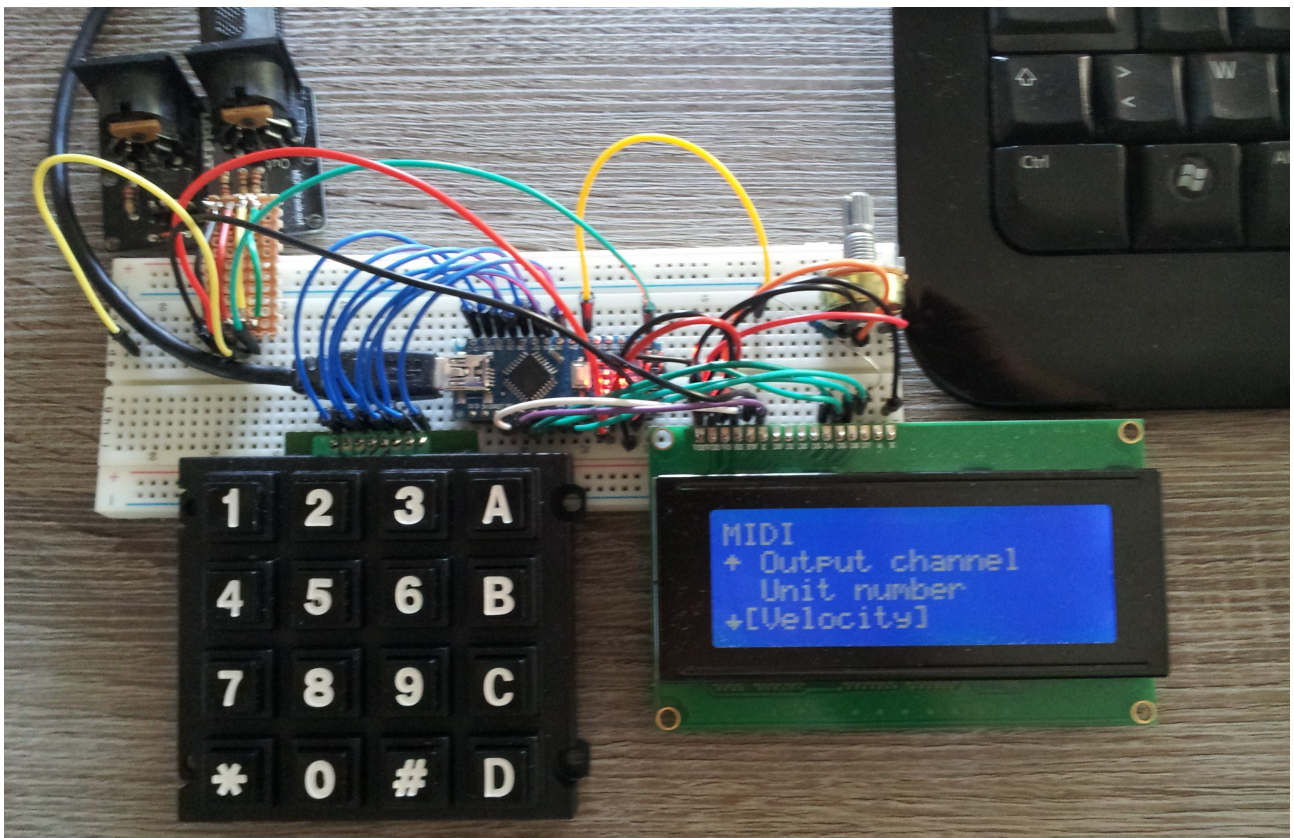


# ArduinoMenu



## Table des matières

Partie menu.....	3
Définition de "Menu".....	3
Représentation informatique d'un menu.....	3
Exemple de menu en XML.....	3
Autre représentation d'un menu.....	4
Fils de .....	5
Tables de navigation.....	5
Table de types d'objets.....	5
Table de labels.....	6
Table de fonctions.....	6
Table des fonctions d'affichage/édition des variables.....	6
Bouts de code généré.....	6
Conclusion.....	14
Partie affichage.....	15
Comment afficher un menu?.....	15
Partie affichage de variable.....	16
Partie affichage de fonction.....	16
Recapitulatif.....	16
Partie saisie.....	17
Conclusion.....	17
Organigramme.....	19

## Partie menu

### Définition de "Menu"

Un menu est une suite d'objets listés sur un écran. On peut s'y déplacer et modifier les objets. Voici une liste non exhaustive des objets que l'on peut rencontrer dans un menu.

- Sous-menu Il contient d'autres listes d'objets. On va également l'appeler *menu*.
- Variable On peut la voir et/ou la modifier.
- Fonction On peut l'exécuter.

Quelque soit son type, un objet a besoin d'un *label* pour être affichable dans le menu. Il faut faire la distinction entre la navigation et les objets. La navigation est le déplacement depuis un objet jusqu'à un autre. L'objet sert parfois à naviguer (cas du sous-menu) mais c'est une exception. Si on sélectionne un sous-menu, on ouvrira la liste de ses objets. Si on sélectionne une variable, c'est bien évidemment pour la lire ou la modifier. Si on sélectionne une fonction, c'est pour l'exécuter. Mais **tous les objets doivent être "navigable"**.

### Représentation informatique d'un menu

Informatiquement parlant, un menu est un arbre dont chaque branche est un objet. La représentation la plus simple d'un arbre est le fichier XML.

Interpréter à la volée un fichier XML est une tâche lourde, c'est pourquoi elle ne sera pas faite dans le microcontrôleur mais en amont, au moment de la compilation, voire avant. Ecrivons donc d'abord sous forme XML le descriptif de ce que l'on voudrait pour notre menu.

- type menu, variable, fonction. Désormais, on mélange menus et sous-menus.
- cname Nom en langage C de l'objet.
- label Texte qui représente l'objet dans le menu.

En XML, cette structure s'impose quasiment d'elle même. Faisons simple, par exemple on ne va pas essayer d'écrire en différentes langues avec des caractères accentués... On se contentera de l'anglais qui a l'avantage de n'utiliser que des caractères ascii parfaitement définis.

### Exemple de menu en XML

```
<?xml version="1.0" encoding="UTF-8"?>
<menuTree>
  <menu cname="globalEdit" label="GLOBAL EDIT">
    <variable cname="channelIn" label="Input channel" />
    <variable cname="channelOut" label="Output channel" />
    <variable cname="programNumber" label="Program number" />
    <variable cname="arpeggiator" label="Arpeggiator" />
    <variable cname="clockIn" label="Midi clock in" />
    <variable cname="clockOut" label="Midi clock out" />
    <variable cname="keyClick" label="Keyboard" />
    <variable cname="audioBeat" label="Metronome" />
    <variable cname="sysEx" label="systemExclusive" />
    <variable cname="transposition" label="Transposition" />
    <function cname="dumpAll" label="Dump all" />
  </menu>
</menuTree>
```

```

    <function cname="loadAll" label="Load all" />
    <function cname="saveAll" label="Save all" />
    <function cname="dumpGlobal" label="Dump global par." />
    <function cname="loadGlobal" label="Load global par." />
    <function cname="saveGlobal" label="Save global par." />
  </menu>
  <menu cname="sequenceEdit" label="SEQUENCE EDIT">
    <variable cname="Groove" label="Groove" />
    <variable cname="gateMode" label="Gate duration" />
    <variable cname="lastStep" label="Last step" />
    <variable cname="ccNum" label="Ctrl chg number" />
    <function cname="initSeq" label="initialize seq." />
    <function cname="swapSeq" label="Swap sequences" />
    <function cname="copySeq" label="copy sequence" />
    <function cname="loadFactory" label="Load factory seq" />
    <function cname="dumpSeq" label="Dump sequence" />
    <function cname="loadSeq" label="Load sequence" />
    <function cname="saveSeq" label="Save sequence" />
  </menu>
  <menu cname="maintenance" label="MAINTENANCE">
    <function cname="factorySettings" label="Factory settings" />
    <function cname="testStepLeds" label="Test step leds" />
    <function cname="testIndLeds" label="Test ind. leds" />
    <function cname="testBeeper" label="Test Beeper" />
    <function cname="displayVersions" label="Versions" />
    <menu cname="submenu" label="TESTS">
      <variable cname="test1" label="test 1" />
      <variable cname="test2" label="test 2" />
      <variable cname="test3" label="test 3" />
      <variable cname="appVersion" label="Version number" />
    </menu>
  </menu>
</menuTree>

```

## Autre représentation d'un menu

Une autre façon de représenter un menu est la liste indentée où chaque niveau de branche se représente par une tabulation supplémentaire.

IDX	LABEL	PARENT	CHILD	NEXT	PREVIOUS
0	GLOBAL EDIT	---	1	17	---
1	Input channel	0	---	2	---
2	Output channel	0	---	3	1
3	Program number	0	---	4	2
4	Arpeggiator	0	---	5	3
5	Midi clock in	0	---	6	4
6	Midi clock out	0	---	7	5
7	Keyboard	0	---	8	6
8	Metronome	0	---	9	7
9	systemExclusive	0	---	10	8
10	Transposition	0	---	11	9
11	Dump all	0	---	12	10
12	Load all	0	---	13	11
13	Save all	0	---	14	12
14	Dump global par.	0	---	15	13
15	Load global par.	0	---	16	14
16	Save global par.	0	---	---	15
17	SEQUENCE EDIT	---	18	29	0
18	Groove	17	---	19	---
19	Gate duration	17	---	20	18

20	Last step	17	---	21	19
21	Ctrl chg number	17	---	22	20
22	initialize seq.	17	---	23	21
23	Swap sequences	17	---	24	22
24	copy sequence	17	---	25	23
25	Load factory seq	17	---	26	24
26	Dump sequence	17	---	27	25
27	Load sequence	17	---	28	26
28	Save sequence	17	---	---	27
29	MAINTENANCE	---	30	---	17
30	Factory settings	29	---	31	---
31	Test step leds	29	---	32	30
32	Test ind. leds	29	---	33	31
33	Test Beeper	29	---	34	32
34	Versions	29	---	35	33
35	TESTS	29	36	---	34
36	test 1	35	---	37	---
37	test 2	35	---	38	36
38	test 3	35	---	39	37
39	Version number	35	---	---	38

## Fils de

*GLOBAL EDIT*, *SEQUENCE EDIT* et *MAINTENANCE* sont frères. *Input channel* est enfant de *GLOBAL EDIT*, *MAINTENANCE* est parent de *Factory settings*. *Arpeggiator* est frère précédent de *Midi clock in* et est également frère suivant de *Program number*. *Input channel* n'a pas de frère précédent.

Cette liste indentée a été obtenue par un programme python qui parcourt (*to parse* en anglais) le fichier XML et extrait les informations, c'est *printLabels.py*.

Vous avez bien sûr compris que l'on cherche une information dans une table grâce à l'index de l'objet, et que cet index a été généré par la position de l'objet dans le fichier XML.

## Tables de navigation

Notre module *arduinoMenu* devra donc gérer les 4 listes PARENT, CHILD, NEXT (pour *prochain frère*) et PREVIOUS (pour *frère précédent*) afin de gérer la navigation dans le menu.

Il faut une valeur NO\_ENTRY (représentée dans la liste précédente par ---) qui signifie *il n'y a pas de parent, il n'y a pas d'enfant, il n'y a pas de frère suivant ou de frère précédent*. Cette valeur ne peut pas être zéro, car le zéro est l'index du premier objet d'un tableau. Si on fait tenir cet index sur un byte, il devrait être la valeur 0xff, c'est à dire la dernière valeur. Notre menu peut ainsi avoir 255 entrées (index de 0 à 254), puisque la 256<sup>ème</sup> (à l'index 255) est réservée pour la valeur NO\_ENTRY.

Le travail ne serait pas fini sans générer quelques autres tables qui ne servent pas à la navigation proprement dite, mais qui sont néanmoins indispensables. Il s'agit des tables de labels, de fonctions, de fonctions d'édition et d'affichage de variables.

## Table de types d'objets

Tant que l'on se contente de naviguer, pas de problème. Mais à partir du moment où on sélectionne un objet, il faut connaître son type pour savoir quelle action engager.

## Table de labels

Pour être affichable dans le menu, chaque objet doit avoir un label. Il doit donc y avoir également une table de labels dans le module *arduinoMenu*. Et pourquoi ne pas prendre le cname comme label? Parce que le cname a beaucoup de contraintes, en particulier la plus gênante : Il ne doit pas contenir d'espace.

S'il faut vraiment écrire des labels en plusieurs langues, la technique classique est de supprimer le label et d'utiliser le cname de l'objet comme une clef de traduction. Au moment d'afficher le texte, on cherche sa traduction dans la table de la langue voulue grâce à la clef. Cette solution est trop lourde pour notre microcontrôleur, on se contentera de l'ascii pur et dur de l'anglais.

Un moteur de menu doit non seulement gérer la navigation, mais aussi l'affichage et le rafraichissement de l'écran, l'affichage et la modification des variables ainsi que l'exécution des fonctions... Sinon, ça ne sert strictement à rien de les faire apparaître dans le menu! Dans le cas de *arduinoMenu*, ce sera fait à base de *callbacks*. Le callback est le seul moyen informatique de séparer les modules ayant une tâche précise. On peut le faire sans callback, mais le prix à payer est que les modules seront imbriqués...On ne pourra pas utiliser le module de menu sans utiliser le module d'affichage ou de saisie! Du coup, ce petit moteur de menu perdrait toute sa portabilité.

## Table de fonctions

Ces fonctions seront du type *void functionName()* qui est la signature la plus courante en C. Quand le menu détectera que l'utilisateur appelle cette fonction, il devra l'exécuter.

## Table des fonctions d'affichage/édition des variables

Ces fonctions seront du type *void functionName(byte action)*. Quand le menu détectera que l'utilisateur appelle cette fonction, il devra l'exécuter. Le paramètre byte pourra prendre les valeurs suivantes sous forme de `#define` :

- |   |                                      |
|---|--------------------------------------|
| • <code>MENU_BROWSER_DATA_INCREASE</code>     | Incrémenter et afficher la variable. |
| • <code>MENU_BROWSER_DATA_DECREASE</code>     | Décrémenter et afficher la variable. |
| • <code>MENU_BROWSER_DATA_JUST_DISPLAY</code> | Juste afficher la variable.          |

Quand le menu détectera que l'utilisateur veut voir ou éditer une variable, il devra l'afficher, scruter l'organe de saisie (clavier, boutons, encodeur incrémental etc...) de l'utilisateur pour modifier la variable et la réafficher si elle est modifiée... Comme notre menu ne contient que le nom de la variable, il va falloir fabriquer le nom de la fonction d'affichage et de la fonction de modification à partir du nom de la variable. Par exemple le nom de variable *lastStep* va servir à fabriquer le nom de fonction *editLastStep*. C'est ainsi que le moteur de menu se contente d'être un moteur de menu et délègue les autres tâches aux parties du programme qui savent le faire.

On se rend compte que fabriquer cet ensemble de tables en écrivant du code C est titanesque, sans parler du risque d'erreurs de saisie ni de déboguage. C'est pourquoi le script python *menuGenerator.py* a été créé. Il fait très exactement la création des tables citées plus haut en analysant le fichier XML décrivant le menu. Il évite ainsi un travail vraiment long et fastidieux, sans parler des risques d'erreurs humaines.

## Bouts de code généré

A partir de l'exemple de fichier XML plus haut, voici les bouts de code généré par le script

**menuGenerator.py** mis bout à bout. Ils peuvent être des tableaux C mais aussi de simples déclarations qui se suivent une à une. Les voici dans leur intégralité, quasiment sans commentaires, mais si vous avez suivi jusqu'ici, ça ne devrait par être un problème pour vous. Un header a été rajouté et le code qui suit est le fichier *menuData.h* dans son intégralité.

```
#ifndef menuData_h
#define menuData_h

/*
 * file : menuData.h
 * Copyright (c) Pfeuh <ze.pfeuh@gmail.com>
 * All rights reserved.
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 */

/*****
 * WARNING!
 * This file is generated by an application,
 * if you edit it, it will be overwritten
 * by the next generation!
 *****/

#include <Arduino.h>

#define MENU_VERSION "1.00"

// Table of parent          00040
// Table of child           00040
// Table of next            00040
// Table of previous        00040
// Table of labels          00512
// Table of pointers to labels 00080
// Table of function callbacks 00080
// Table of edit callbacks  00080
// Table of display callbacks 00080
// Table of types           00040
// -----
// TOTAL                    01032

// functions supposed ready to execute
extern void dumpAll();
extern void loadAll();
extern void saveAll();
extern void dumpGlobal();
extern void loadGlobal();
extern void saveGlobal();
extern void initSeq();
extern void swapSeq();
```

```

extern void copySeq();
extern void loadFactory();
extern void dumpSeq();
extern void loadSeq();
extern void saveSeq();
extern void factorySettings();
extern void testStepLeds();
extern void testIndLeds();
extern void testBeeper();
extern void displayVersions();

// edit & display functions of variables supposed ready to execute
extern void editChannelIn();
extern void displayChannelIn();
extern void editChannelOut();
extern void displayChannelOut();
extern void editProgramNumber();
extern void displayProgramNumber();
extern void editArpeggiator();
extern void displayArpeggiator();
extern void editClockIn();
extern void displayClockIn();
extern void editClockOut();
extern void displayClockOut();
extern void editKeyClick();
extern void displayKeyClick();
extern void editAudioBeat();
extern void displayAudioBeat();
extern void editSysEx();
extern void displaySysEx();
extern void editTransposition();
extern void displayTransposition();
extern void editGroove();
extern void displayGroove();
extern void editGateMode();
extern void displayGateMode();
extern void editLastStep();
extern void displayLastStep();
extern void editCcNum();
extern void displayCcNum();
extern void editTest1();
extern void displayTest1();
extern void editTest2();
extern void displayTest2();
extern void editTest3();
extern void displayTest3();
extern void editAppVersion();
extern void displayAppVersion();

const byte parentTable[40] PROGMEM =
{
    /* 000 */ MENU_NO_ENTRY,
    /* 001 */ 0,
    /* 002 */ 0,
    /* 003 */ 0,
    /* 004 */ 0,
    /* 005 */ 0,
    /* 006 */ 0,
    /* 007 */ 0,
    /* 008 */ 0,
    /* 009 */ 0,

```



```

/* 010 */ 0,
/* 011 */ 0,
/* 012 */ 0,
/* 013 */ 0,
/* 014 */ 0,
/* 015 */ 0,
/* 016 */ 0,
/* 017 */ MENU_NO_ENTRY,
/* 018 */ 17,
/* 019 */ 17,
/* 020 */ 17,
/* 021 */ 17,
/* 022 */ 17,
/* 023 */ 17,
/* 024 */ 17,
/* 025 */ 17,
/* 026 */ 17,
/* 027 */ 17,
/* 028 */ 17,
/* 029 */ MENU_NO_ENTRY,
/* 030 */ 29,
/* 031 */ 29,
/* 032 */ 29,
/* 033 */ 29,
/* 034 */ 29,
/* 035 */ 29,
/* 036 */ 35,
/* 037 */ 35,
/* 038 */ 35,
/* 039 */ 35,
};

const byte childTable[40] PROGMEM =
{
/* 000 */ 1,
/* 001 */ MENU_NO_ENTRY,
/* 002 */ MENU_NO_ENTRY,
/* 003 */ MENU_NO_ENTRY,
/* 004 */ MENU_NO_ENTRY,
/* 005 */ MENU_NO_ENTRY,
/* 006 */ MENU_NO_ENTRY,
/* 007 */ MENU_NO_ENTRY,
/* 008 */ MENU_NO_ENTRY,
/* 009 */ MENU_NO_ENTRY,
/* 010 */ MENU_NO_ENTRY,
/* 011 */ MENU_NO_ENTRY,
/* 012 */ MENU_NO_ENTRY,
/* 013 */ MENU_NO_ENTRY,
/* 014 */ MENU_NO_ENTRY,
/* 015 */ MENU_NO_ENTRY,
/* 016 */ MENU_NO_ENTRY,
/* 017 */ 18,
/* 018 */ MENU_NO_ENTRY,
/* 019 */ MENU_NO_ENTRY,
/* 020 */ MENU_NO_ENTRY,
/* 021 */ MENU_NO_ENTRY,
/* 022 */ MENU_NO_ENTRY,
/* 023 */ MENU_NO_ENTRY,
/* 024 */ MENU_NO_ENTRY,
/* 025 */ MENU_NO_ENTRY,
/* 026 */ MENU_NO_ENTRY,

```

```

/* 027 */ MENU_NO_ENTRY,
/* 028 */ MENU_NO_ENTRY,
/* 029 */ 30,
/* 030 */ MENU_NO_ENTRY,
/* 031 */ MENU_NO_ENTRY,
/* 032 */ MENU_NO_ENTRY,
/* 033 */ MENU_NO_ENTRY,
/* 034 */ MENU_NO_ENTRY,
/* 035 */ 36,
/* 036 */ MENU_NO_ENTRY,
/* 037 */ MENU_NO_ENTRY,
/* 038 */ MENU_NO_ENTRY,
/* 039 */ MENU_NO_ENTRY,
};

```

```

const byte nextTable[40] PROGMEM =
{

```

```

/* 000 */ 17,
/* 001 */ 2,
/* 002 */ 3,
/* 003 */ 4,
/* 004 */ 5,
/* 005 */ 6,
/* 006 */ 7,
/* 007 */ 8,
/* 008 */ 9,
/* 009 */ 10,
/* 010 */ 11,
/* 011 */ 12,
/* 012 */ 13,
/* 013 */ 14,
/* 014 */ 15,
/* 015 */ 16,
/* 016 */ MENU_NO_ENTRY,
/* 017 */ 29,
/* 018 */ 19,
/* 019 */ 20,
/* 020 */ 21,
/* 021 */ 22,
/* 022 */ 23,
/* 023 */ 24,
/* 024 */ 25,
/* 025 */ 26,
/* 026 */ 27,
/* 027 */ 28,
/* 028 */ MENU_NO_ENTRY,
/* 029 */ MENU_NO_ENTRY,
/* 030 */ 31,
/* 031 */ 32,
/* 032 */ 33,
/* 033 */ 34,
/* 034 */ 35,
/* 035 */ MENU_NO_ENTRY,
/* 036 */ 37,
/* 037 */ 38,
/* 038 */ 39,
/* 039 */ MENU_NO_ENTRY,
};

```

```

const byte previousTable[40] PROGMEM =
{

```

```

/* 000 */ MENU_NO_ENTRY,
/* 001 */ MENU_NO_ENTRY,
/* 002 */ 1,
/* 003 */ 2,
/* 004 */ 3,
/* 005 */ 4,
/* 006 */ 5,
/* 007 */ 6,
/* 008 */ 7,
/* 009 */ 8,
/* 010 */ 9,
/* 011 */ 10,
/* 012 */ 11,
/* 013 */ 12,
/* 014 */ 13,
/* 015 */ 14,
/* 016 */ 15,
/* 017 */ 0,
/* 018 */ MENU_NO_ENTRY,
/* 019 */ 18,
/* 020 */ 19,
/* 021 */ 20,
/* 022 */ 21,
/* 023 */ 22,
/* 024 */ 23,
/* 025 */ 24,
/* 026 */ 25,
/* 027 */ 26,
/* 028 */ 27,
/* 029 */ 17,
/* 030 */ MENU_NO_ENTRY,
/* 031 */ 30,
/* 032 */ 31,
/* 033 */ 32,
/* 034 */ 33,
/* 035 */ 34,
/* 036 */ MENU_NO_ENTRY,
/* 037 */ 36,
/* 038 */ 37,
/* 039 */ 38,
};

/* 0 */ const char labelGlobalEdit[] PROGMEM = "GLOBAL EDIT";
/* 1 */ const char labelChannelIn[] PROGMEM = "Input channel";
/* 2 */ const char labelChannelOut[] PROGMEM = "Output channel";
/* 3 */ const char labelProgramNumber[] PROGMEM = "Program number";
/* 4 */ const char labelArpeggiator[] PROGMEM = "Arpeggiator";
/* 5 */ const char labelClockIn[] PROGMEM = "Midi clock in";
/* 6 */ const char labelClockOut[] PROGMEM = "Midi clock out";
/* 7 */ const char labelKeyClick[] PROGMEM = "Keyboard";
/* 8 */ const char labelAudioBeat[] PROGMEM = "Metronome";
/* 9 */ const char labelSysEx[] PROGMEM = "systemExclusive";
/* 10 */ const char labelTransposition[] PROGMEM = "Transposition";
/* 11 */ const char labelDumpAll[] PROGMEM = "Dump all";
/* 12 */ const char labelLoadAll[] PROGMEM = "Load all";
/* 13 */ const char labelSaveAll[] PROGMEM = "Save all";
/* 14 */ const char labelDumpGlobal[] PROGMEM = "Dump global par.";
/* 15 */ const char labelLoadGlobal[] PROGMEM = "Load global par.";
/* 16 */ const char labelSaveGlobal[] PROGMEM = "Save global par.";
/* 17 */ const char labelSequenceEdit[] PROGMEM = "SEQUENCE EDIT";
/* 18 */ const char labelGroove[] PROGMEM = "Groove";

```

```

/* 19 */ const char labelGateMode[] PROGMEM = "Gate duration";
/* 20 */ const char labelLastStep[] PROGMEM = "Last step";
/* 21 */ const char labelCcNum[] PROGMEM = "Ctrl chg number";
/* 22 */ const char labelInitSeq[] PROGMEM = "initialize seq.";
/* 23 */ const char labelSwapSeq[] PROGMEM = "Swap sequences";
/* 24 */ const char labelCopySeq[] PROGMEM = "copy sequence";
/* 25 */ const char labelLoadFactory[] PROGMEM = "Load factory seq";
/* 26 */ const char labelDumpSeq[] PROGMEM = "Dump sequence";
/* 27 */ const char labelLoadSeq[] PROGMEM = "Load sequence";
/* 28 */ const char labelSaveSeq[] PROGMEM = "Save sequence";
/* 29 */ const char labelMaintenance[] PROGMEM = "MAINTENANCE";
/* 30 */ const char labelFactorySettings[] PROGMEM = "Factory settings";
/* 31 */ const char labelTestStepLeds[] PROGMEM = "Test step leds";
/* 32 */ const char labelTestIndLeds[] PROGMEM = "Test ind. leds";
/* 33 */ const char labelTestBeeper[] PROGMEM = "Test Beeper";
/* 34 */ const char labelDisplayVersions[] PROGMEM = "Versions";
/* 35 */ const char labelSubmenu[] PROGMEM = "TESTS";
/* 36 */ const char labelTest1[] PROGMEM = "test 1";
/* 37 */ const char labelTest2[] PROGMEM = "test 2";
/* 38 */ const char labelTest3[] PROGMEM = "test 3";
/* 39 */ const char labelAppVersion[] PROGMEM = "Version number";

```

```
const char *const labelsTable[] PROGMEM =
```

```

{
    /* 0 */ labelGlobalEdit,
    /* 1 */ labelChannelIn,
    /* 2 */ labelChannelOut,
    /* 3 */ labelProgramNumber,
    /* 4 */ labelArpeggiator,
    /* 5 */ labelClockIn,
    /* 6 */ labelClockOut,
    /* 7 */ labelKeyClick,
    /* 8 */ labelAudioBeat,
    /* 9 */ labelSysEx,
    /* 10 */ labelTransposition,
    /* 11 */ labelDumpAll,
    /* 12 */ labelLoadAll,
    /* 13 */ labelSaveAll,
    /* 14 */ labelDumpGlobal,
    /* 15 */ labelLoadGlobal,
    /* 16 */ labelSaveGlobal,
    /* 17 */ labelSequenceEdit,
    /* 18 */ labelGroove,
    /* 19 */ labelGateMode,
    /* 20 */ labelLastStep,
    /* 21 */ labelCcNum,
    /* 22 */ labelInitSeq,
    /* 23 */ labelSwapSeq,
    /* 24 */ labelCopySeq,
    /* 25 */ labelLoadFactory,
    /* 26 */ labelDumpSeq,
    /* 27 */ labelLoadSeq,
    /* 28 */ labelSaveSeq,
    /* 29 */ labelMaintenance,
    /* 30 */ labelFactorySettings,
    /* 31 */ labelTestStepLeds,
    /* 32 */ labelTestIndLeds,
    /* 33 */ labelTestBeeper,
    /* 34 */ labelDisplayVersions,
    /* 35 */ labelSubmenu,
    /* 36 */ labelTest1,

```

```

    /* 37 */ labelTest2,
    /* 38 */ labelTest3,
    /* 39 */ labelAppVersion,
};

const PROGMEM MENU_FUNCTION_PTR execFunctionsTable[18] =
{
    /* 11 0 */ &dumpAll,
    /* 12 1 */ &loadAll,
    /* 13 2 */ &saveAll,
    /* 14 3 */ &dumpGlobal,
    /* 15 4 */ &loadGlobal,
    /* 16 5 */ &saveGlobal,
    /* 22 6 */ &initSeq,
    /* 23 7 */ &swapSeq,
    /* 24 8 */ &copySeq,
    /* 25 9 */ &loadFactory,
    /* 26 10 */ &dumpSeq,
    /* 27 11 */ &loadSeq,
    /* 28 12 */ &saveSeq,
    /* 30 13 */ &factorySettings,
    /* 31 14 */ &testStepLeds,
    /* 32 15 */ &testIndLeds,
    /* 33 16 */ &testBeeper,
    /* 34 17 */ &displayVersions,
};

const PROGMEM MENU_FUNCTION_PTR editFunctionsTable[18] =
{
    /* 1 0 */ &editChannelIn,
    /* 2 1 */ &editChannelOut,
    /* 3 2 */ &editProgramNumber,
    /* 4 3 */ &editArpeggiator,
    /* 5 4 */ &editClockIn,
    /* 6 5 */ &editClockOut,
    /* 7 6 */ &editKeyClick,
    /* 8 7 */ &editAudioBeat,
    /* 9 8 */ &editSysEx,
    /* 10 9 */ &editTransposition,
    /* 18 10 */ &editGroove,
    /* 19 11 */ &editGateMode,
    /* 20 12 */ &editLastStep,
    /* 21 13 */ &editCcNum,
    /* 36 14 */ &editTest1,
    /* 37 15 */ &editTest2,
    /* 38 16 */ &editTest3,
    /* 39 17 */ &editAppVersion,
};

const enum menuOptionType itemTypeTable[40] PROGMEM =
{
    /* 000 */ menuTypeMenu,
    /* 001 */ menuTypeVariable,
    /* 002 */ menuTypeVariable,
    /* 003 */ menuTypeVariable,
    /* 004 */ menuTypeVariable,
    /* 005 */ menuTypeVariable,
    /* 006 */ menuTypeVariable,
    /* 007 */ menuTypeVariable,
    /* 008 */ menuTypeVariable,
    /* 009 */ menuTypeVariable,

```

```

/* 010 */ menuTypeVariable,
/* 011 */ menuTypeFunction,
/* 012 */ menuTypeFunction,
/* 013 */ menuTypeFunction,
/* 014 */ menuTypeFunction,
/* 015 */ menuTypeFunction,
/* 016 */ menuTypeFunction,
/* 017 */ menuTypeMenu,
/* 018 */ menuTypeVariable,
/* 019 */ menuTypeVariable,
/* 020 */ menuTypeVariable,
/* 021 */ menuTypeVariable,
/* 022 */ menuTypeFunction,
/* 023 */ menuTypeFunction,
/* 024 */ menuTypeFunction,
/* 025 */ menuTypeFunction,
/* 026 */ menuTypeFunction,
/* 027 */ menuTypeFunction,
/* 028 */ menuTypeFunction,
/* 029 */ menuTypeMenu,
/* 030 */ menuTypeFunction,
/* 031 */ menuTypeFunction,
/* 032 */ menuTypeFunction,
/* 033 */ menuTypeFunction,
/* 034 */ menuTypeFunction,
/* 035 */ menuTypeMenu,
/* 036 */ menuTypeVariable,
/* 037 */ menuTypeVariable,
/* 038 */ menuTypeVariable,
/* 039 */ menuTypeVariable,
};

#endif

```

## Conclusion

Ca y est, on a le code des données de notre menu. On ne sait pas encore quoi en faire, mais on l'a. On se doute bien qu'il va falloir l'insérer quelque part dans les fichiers .c .cpp ou .h de la future bibliothèque, mais sans plus. Le plus simple est de créer un fichier *menuData.h* qui sera inclus **uniquement** par *menu.cpp* (hé oui, le module *menu* sera un module .cpp).

Il y a dans le générateur une petite partie de code inutilisée jusqu'à présent, c'est la création des squelettes des fonctions. Car ce que le générateur ne peut pas vérifier, c'est l'existence de ces fonctions dans le reste du code. Pour chaque fonction non trouvée, il y a bien sûr une erreur de compilation. Si on appelle cette petite partie, on peut démarrer une application à partir de son menu, ceci pour se concentrer évidemment sur le menu, finies les erreurs de compilation, car même si elles sont vides, les fonctions existent. Pour appeler cette petite partie, il suffit de rajouter `-c` (pour *createEmptyFunction*) à la ligne de commande qui lance **menuGenerator.py**. Travailler sur une partie d'une application alors que les autres parties de l'application n'existent pas encore est un souhait très courant chez les développeurs.

## Partie affichage

Cette partie est très spécifique au côté embarqué de la famille Arduino du fait des périphériques d'affichage utilisés, ce sont souvent des afficheurs 7 segments, des lcd en mode texte, plutôt que des moniteurs VGA, écrans tactiles etc...

Le module *menuDisplay* utilise un lcd 4 lignes de 20 caractères. Niveau hardware, la communication se fait en I2C sur un mode 8 bits. Si vous voulez autre chose... Il vous faudra écrire votre module en vous inspirant de celui là.

### Comment afficher un menu?

La façon la plus simple d'afficher un menu est une liste. On entoure l'objet qui a le focus par une paire de balises ou de crochets. Comme par exemple ici :

```
Program number
<Arpeggiator      >
Midi clock in
Midi clock out
```

On comprend facilement que l'objet qui a le focus est *Arpeggiator*. Par contre, on ne voit pas qu'il y a d'autres objets devant *Program number* et derrière *Midi clock out*. C'est pourquoi on va redéfinir 2 caractères pour représenter une flèche vers le haut et une flèche vers le bas. Utiliser quelques caractères redéfinis fait d'ailleurs partie des fonctionnalités de base des afficheurs lcd de type texte.

```
↑ Program number
<Arpeggiator      >
Midi clock in
↓ Midi clock out
```

On constate que le texte le plus large est celui de l'objet *Arpeggiator*, à cause de la balise fermante. Si on se donne la peine de compter, on arrive à 20 caractères, ce qui est pile la largeur de notre lcd. On a du coup la première constante de notre module d'affichage, c'est 20 – largeur de < - largeur de > - largeur de ↑ = 17, ce qui va être la taille maximum des labels que l'on peut afficher. D'où cette ligne quelque part dans un fichier .h:

```
#define MENU_LABEL_MAX_SIZE 17
```

Il ya encore une autre chose que l'on ne voit pas, c'est la position de notre liste dans l'arborescence. C'est pourquoi on va consacrer la première ligne de l'afficheur au chemin, par exemple comme ceci :

```
GLOBAL EDIT
↑<Arpeggiator      >
Midi clock in
↓ Midi clock out
```

On voit qu'à partir du 2<sup>ème</sup> niveau d'arborescence, on va avoir un problème de place sur notre première ligne, car  $2 \times 17 = 34$ , ce qui est déjà beaucoup plus que ce qui est physiquement faisable. Dans certains menus évolués il y a un bouton "*Were am I?*" qui permet d'afficher momentanément

le chemin complet. On va plutôt écrire un algorithme qui prend le début et la fin du chemin en mettant 3 points au milieu.

Force est de constater que l'affichage des menus en majuscules et des variables/fonctions en normal donne une vision de l'arborescence plus précise, on sait intuitivement qu'il y a d'autres menus derrière les majuscules et une variable/fonction sinon. On peut partir de cette constatation pour écrire une première règle de conception des menus, mais ce n'est absolument pas une obligation.

On arrive tout doucement au bout de la gestion de l'affichage partie navigation dans les menus, il reste la partie affichage des variables et des fonctions. Comment afficher un menu?

## **Partie affichage de variable**

Je rappelle que cette partie est assurée par une partie externe à *arduinoMenu*. Mais rien n'empêche de lui fournir un écran préformaté, une sorte de template, de sorte qu'elle l'utilise quand elle doit afficher quelque chose. En fait il suffit d'afficher le template juste avant de passer la main à la fonction d'affichage et de fournir la fonction d'affichage de la variable, c'est à dire celle qui va poser la valeur de la variable au bon endroit sur l'écran.

A ce stade, il est bon de rappeler que comme cette partie est externe à *arduinoMenu*, elle peut gérer tout ce qui est envisageable comme type de variable, du signé, du flottant, du texte etc... Voici une idée d'écran que l'on pourrait fournir, la dernière ligne servant à afficher la valeur de la variable.

```
GLOBAL EDIT
<Arpeggiator>
Edit value:
123
```

## **Partie affichage de fonction**

Comment, il faut afficher quelque chose quand on exécute une fonction? Hé oui, si on exécutait la fonction sans aucun retour visuel, on aurait toujours un doute : La fonction a t-elle été exécutée ou pas? Donc proposition d'écran (optionnel) pour avant l'exécution:

```
MAINTENANCE/TESTS
<test 1>
Hit button to exec.:
```

Et après exécution :

```
MAINTENANCE/TESTS
<test 1>
Execution done!
```

## **Récapitulatif**

Notre partie affichage sera donc le module *menuDisplay* capable d'effectuer les tâches suivantes:



- `refreshBrowserScreen`      Rafraichissement de l'écran de navigation.
- `showEditVariableScreen`      Afficher l'écran d'édition de variable.
- `showVariable`      Afficher la valeur de la variable au bon endroit de l'écran.
- `showPrefunctionScreen`      Afficher l'écran de prélancement de fonction.
- `showPostfunctionScreen`      Afficher l'écran de postlancement de fonction.
- `userPrint`      L'utilisateur affiche ce qu'il veut où il veut.

Il faut faire la différence entre *showEditVariableScreen* qui affiche l'écran complet et *showVariable* qui n'affiche que la valeur de la variable sur l'écran à la place qui lui a été allouée. Le *userPrint* est un plus, au cas plus que probable où l'utilisateur veut afficher autre chose que du menu.

On peut dire que n'importe quel périphérique de sortie capable d'afficher à un endroit précis d'un écran pourra servir à afficher un menu, il suffit d'implémenter les fonctions listées plus haut.

Si on ne se sert du menu que pour configurer, (très rarement, donc) on peut envisager un terminal PC connecté en USB... Mais il faut un terminal "amélioré", celui de l'environnement de développement Arduino ne peut ni repositionner le curseur ni effacer l'écran.

## Partie saisie

Par saisie, on entend :

- Se déplacer dans le menu.
- Sélectionner une variable.
- Editer une variable.
- Sélectionner une fonction.
- Exécuter une fonction ou annuler.

Notre partie saisie sera donc le module *menuinput* capable d'effectuer ces tâches. Dans le cas de la famille Arduino, le moyen le plus courant est un joystick tout ou rien. Les 4 directions en tout ou rien suffisent, il n'y a même pas besoin de boutons supplémentaires!

- haut, bas      se déplacer, incrémenter/décrémenter une variable.
- droite      entrer dans un sous menu, valider une fonction.
- gauche      sortir d'un sous-menu, annuler une fonction.

Un autre périphérique courant est l'encodeur incrémental. Il est composé d'un axe rotatif et d'un click.

- Tourner droite/gauche      se déplacer, incrémenter/décrémenter une variable.
- click court (normal)      entrer dans un sous menu, valider une fonction.
- click long      sortir d'un sous-menu, annuler une fonction.

Il faut néanmoins être absolument sûr de son encodeur, car s'il a des rebonds aléatoires, son utilisation devient vite un cauchemar, en particulier au niveau des click longs et courts.

## Conclusion

Le module *arduinoMenu* est constitué des 4 modules suivants:

- **Module menuBrowser**            Navigation dans les menus.
  - `menuBrowser.cpp`
  - `menuBrowser.h`
  - `menuData.h`
- **Module menuDisplay**           Affichage des différents écrans.
  - `MenuDisplay.hpp`
  - `MenuDisplay.h`
- **Module menuInput**            Saisie des données utilisateur.
  - `menuInput.cpp`
  - `menuInput.h`
- **Module arduinoMenu**           Fait le lien entre l'application et les 3 autres modules.
  - `arduinoMenu.hpp`
  - `arduinoMenu.h`

Ces 4 modules sont embarqués dans la bibliothèque *arduinoMenu* pour la famille Arduino. Il y a aussi le module python *menuGenerator.py* qui doit être exécuté en amont de la compilation du micro afin de ne pas saisir à la main toutes les données dont le moteur de menu a besoin.

Chaque module est totalement indépendant grâce au système de *callbacks*, le développeur peut personnaliser ce qu'il veut de façon assez simple. Dans l'état, cette bibliothèque est fournie pour:

- Afficheur LCD 20 x 4 text en I2C
- Joystick tout ou rien 4 directions.

Il y a dans le répertoire *examples* un exemple d'utilisation avec le terminal PC en USB, ça peut servir de base pour se construire un moteur de menu personnalisé. Avantage non négligeable, il fonctionne sans aucun hardware additionnel, ce qui est le rêve pour un développeur logiciel.

## Organigramme

Un dessin valant mieux qu'un long discours, voici l'organigramme de l'interaction entre l'application (appelée sketch chez Arduino), *arduinoMenu* et *menuGenerator.py*.

