

UTComputer

On souhaite développer l'application `UTCOMPUTER`, un calculateur scientifique permettant de faire des calculs, de stocker et de manipuler des variables et des programmes, et utilisant la notation RPN (Reverse Polish Notation, *i.e.* la notation polonaise inversée ou encore notation post-fixée). La notation polonaise inverse est en fait une méthode de notation mathématique permettant d'écrire une formule arithmétique de façon non ambiguë sans utiliser de parenthèse. Ainsi :

- L'expression en mode infixe `'1+1'` pourra s'écrire `1 1 +` sur une ligne de commande;
- `'2*2+1'` pourra s'écrire `2 2 * 1 +` sur une ligne de commande;
- `'(2+3)*4'` pourra s'écrire `2 3 + 4 *` sur une ligne de commande.

Pour plus d'information, vous pouvez consulter la page [wikipédia](#) dédiée. La notation post-fixée implique l'utilisation d'une *pile* pour stocker les résultats intermédiaires lorsque l'on évalue une formule arithmétique. Pour interagir avec un calculateur utilisant cette notation l'utilisateur entre une suite d'opérandes, une opérande étant soit une littérale soit un opérateur. Cette suite d'opérandes est alors évaluée de la gauche vers la droite de la façon suivante :

- si l'opérande est une littérale, elle est empilée sur la pile;
- si l'opérande est un opérateur d'arité n , elle provoque le dépilement de la pile de n expressions, l'application de l'opérateur sur ces expressions, et l'empilement du résultat.

Un des inconvénients est qu'un opérateur ne peut pas avoir plusieurs arités. En particulier, l'opérateur `-` ne peut pas être à la fois unaire (pour transformer un nombre en nombre négatif) et binaire (pour faire une soustraction). Il faut donc différencier l'opérateur binaire de soustraction (`'10-2'` devient `10 2 -`) de l'opérateur unaire de négation (`'-2'` devient `2 NEG`).

1 Fonctions principales

Attention, les spécifications présentées dans ce document présentent le fonctionnement de l'application demandée. Vous pouvez les personnaliser à condition de ne pas diminuer la complexité du projet. Ces spécifications vous laissent volontairement des choix de conception, tant fonctionnels que conceptuels et technologiques. Quels que soient les choix et adaptations que vous ferez, vous prendrez garde de les exposer et justifier dans le rapport rendu avec le projet. Il peut manquer des spécifications. Dans ce cas, faites un choix en l'exposant clairement dans votre rapport.

1.1 Éléments d'interface

- Dans sa vue principale, l'application possède une partie dédiée à l'affichage de l'état du calculateur (état de la pile, message pour l'utilisateur), une barre de commande affichant la suite d'opérandes en train d'être saisie, et un *clavier cliquable* à la souris comprenant un pavé numérique et les opérateurs les plus utilisés qui seront placés de façon raisonnablement ergonomique. Les X derniers éléments de la pile sont affichés (où X est paramétrable dans l'application).
- Le calculateur peut être utilisé à la souris en utilisant le *clavier cliquable* ou directement en utilisant le clavier de l'ordinateur.
- L'affichage ou non du *clavier cliquable* est une option paramétrable par l'utilisateur.
- L'affichage d'un nouveau message destiné à l'utilisateur peut s'accompagner d'un bip sonore qui peut être désactivé par l'utilisateur.
- Une vue secondaire de l'application sera dédiée à la gestion et l'édition des variables stockées dans l'application.
- Une vue secondaire de l'application sera dédiée à la gestion et l'édition des (mini-)programmes utilisateurs stockés dans l'application.
- Une vue secondaire de l'application sera dédiée à l'édition des paramètres du calculateur.

1.2 Types de littérale

L'application peut manier différents types de littérales. Dans la suite on distinguera la manière dont les littérales sont affichées et la manière dont elles sont obtenues.

- **Littérale entière.** Une littérale entière est affichée sous forme décimale avec suite de chiffres pouvant commencer par le signe "-" dans le cas d'un nombre entier négatif (ex : 34 ou -52). Une littérale entière positive peut s'obtenir en saisissant une suite de chiffres. Une littérale entière négative peut s'obtenir en appliquant l'opérateur **NEG** sur une littérale entière positive.
- **Littérale rationnelle.** Une littérale rationnelle s'affiche avec deux littérales entières séparées par le signe / (ex : 2/3 ou -2/3). La partie à gauche du signe / s'appelle le numérateur. La partie à droite du signe / s'appelle le dénominateur. Dès qu'une littérale rationnelle est traitée, elle est automatiquement simplifiée. Une littérale rationnelle dont le dénominateur est égal à 1 est simplifiée en une littérale entière. Une littérale rationnelle peut s'obtenir en faisant une division entre deux littérales entières.
- **Littérale réelle.** Une littérale réelle s'affiche sous forme de deux suites de chiffres séparées par le symbole "." pour séparer la partie entière de la mantisse. Lors de son affichage, la partie entière est précédée du signe - dans le cas d'un réel négatif. La partie décimale ou la mantisse peuvent ne pas comporter de chiffres signifiant la valeur 0 (mais une des deux parties doit exister (ex : 3.14 ou .56 ou 3.)). Une littérale réelle dont la mantisse est nulle est simplifiée en une littérale entière. Une littérale réelle positive peut s'obtenir en saisissant une suite de chiffres (en utilisant "." pour séparer la partie entière de la mantisse). Une littérale réelle négative peut s'obtenir en appliquant l'opérateur **NEG** sur une littérale réelle positive.
- **Littérale complexe.** Une littérale complexe s'affiche avec deux littérales entières, rationnelles et/ou réelles séparées par le symbole \$ (ex : -3\$4.2 ou 5\$-3/4). Une littérale complexe peut s'obtenir en appliquant l'opérateur binaire \$ sur deux littérales entières, rationnelles et/ou réelles qui constitueront respectivement la partie réelle et la partie imaginaire de la littérale complexe.
- **Littérale atome.** Une littérale atome est une suite de caractères composée de lettres majuscules et de chiffres et commençant par une lettre majuscule (ex: X1 ou TOTO). Une littérale atome pourra éventuellement correspondre à l'identificateur d'une variable, d'un programme ou au nom d'un opérateur prédéfini s'écrivant avec des lettres (comme DUP, SWAP, CLEAR). Une littérale atome peut s'obtenir simplement en saisissant une suite de lettres majuscules et de chiffres.
- **Littérale expression.** Une littérale expression est une suite de caractères entre guillemets, *i.e.* entourée du caractère '. Elle est composée de littérales entières, réelles, rationnelles, complexes ou atomes séparées par des opérateurs numériques. La notation infixe est utilisée pour les opérateurs binaires utilisant un symbole, *i.e.* les opérateurs +, -, *, et / (ex : '3*X+5'). La notation préfixe est utilisée avec le signe - pour former des nombres négatifs (ex : '-2/3+Z'). La notation infixe est utilisée avec le signe \$ pour séparer la partie réelle de la partie imaginaire d'un nombre complexe. La notation préfixe parenthésée est utilisée dans le cas des autres opérateurs (ex : 'SIN(X)' ou 'DIV(6,7)'). Une littérale expression peut comporter des parenthèses pour modifier les priorités entre opérateurs (ex : '3*X1+(3+TOTO)/-2'). Une littérale expression peut s'obtenir en saisissant une suite de caractères entre guillemets. Une littérale expression ne comporte pas d'espace (des espaces peuvent éventuellement apparaître lors de la saisie par l'utilisateur mais ils sont éliminés sans changer la sémantique de la littérale expression).
- **Littérale programme.** Une littérale programme est une suite d'opérandes commençant par le caractère "[" et terminant par le caractère "] ".

Dans la suite, on parlera de **littérale numérique** pour parler indifféremment de littérale entière, rationnelle ou réelle. Sur la pile, chaque littérale devant s'afficher sur une seule ligne, on affichera un nombre maximum de caractères (en fonction de la largeur de l'affichage de la pile) pour une littérale expression ou une littérale programme en suggérant une suite avec les caractères "...". On prendra soin tout de même d'afficher les délimiteurs gauche et droits de la littérale expression ou programme (ex : 'SIN(X*Y2)+25-18*(14+...' ou [DUP 0 < [NEG ...]).

1.3 Évaluation d'une ligne d'opérandes

- L'utilisateur entre une suite d'opérandes (*i.e.* de littérales et d'opérateurs) séparés par des espaces en utilisant le clavier de l'interface ou le clavier de l'utilisateur (comme 36 X1 STO).
- Cette suite d'opérandes apparaît au fur et à mesure de la saisie dans la barre de commande. Tant qu'elle n'est pas évaluée, cette suite peut être rectifiée uniquement de la droite vers la gauche avec la touche **BACKSPACE** du clavier de l'ordinateur ou avec une touche de correction prévue à cet effet sur le clavier cliquable.

- Une ligne d'opérandes est évaluée directement dès que l'utilisateur tape sur les touches **+**, **-**, *****, **/**, **\$** ou **ENTER** du clavier de l'ordinateur ou dès que l'utilisateur clique **sur un opérateur quelconque** ou sur le bouton **ENTER** du clavier cliquable de l'application. Il n'est alors pas nécessaire d'introduire un espace avant l'une de ces actions.
- Les opérandes de la ligne de commande sont traitées une par une, de la gauche vers la droite.
- Le traitement d'une littérale numérique, expression ou programme provoque l'empilement de la littérale sur la pile.
- Le traitement d'une littérale atome dépend des cas :
 - si la littérale atome correspond à l'identificateur d'une variable, le traitement provoque l'empilement de la littérale associée cette variable;
 - si la littérale atome correspond à l'identificateur d'un programme, le traitement provoque l'évaluation du programme associé (*i.e.* l'exécution de la suite d'opérandes du programme);
 - si l'atome ne correspond ni à l'identificateur d'une variable, ni à l'identificateur d'un programme, le traitement provoque la création d'une nouvelle littérale expression avec la valeur de l'atome entre guillemets.
- Le traitement d'un opérateur n -aire provoque le dépilement des n dernières littérales de la pile avec lesquelles sera effectuée l'opération. L'opération est effectuée sur ces littérales. Le résultat de l'opération (s'il y en a un) est empilé. Si la pile ne contient pas assez de littérales pour exécuter un opérateur, la pile est laissée intacte et un message prévient de la non-validité de l'opération.

Si, lors de son traitement, une opérande n'est pas correcte syntaxiquement, elle est laissée sur la ligne de commande en attente d'une correction de l'utilisateur et un message d'erreur adéquat prévient l'utilisateur. On remarque qu'une littérale expression ou une littérale programme n'est pas évaluée automatiquement; elle est simplement empilée sur la pile.

1.4 Opérateurs

Lorsqu'un opérateur d'arité n est appliqué alors n éléments de la pile sont dépilés. Le premier argument de l'opérateur est alors le dernier dépilé, le 2^e est l'avant dernier dépilé, le 3^e est l'avant-avant dernier dépilé, etc. L'application d'un opérateur peut être restreinte à certains types de littérales. Lors de la tentative d'application d'un opérateur sur un type non prévu, les arguments ne sont pas dépilés et un message d'erreur est affiché pour l'utilisateur.

1.4.1 Opérateurs numériques

Sauf indication, ces opérateurs peuvent être appliqués sur toutes les littérales numériques ou expression.

- **+**, opérateur binaire : addition.
- **-**, opérateur binaire : soustraction.
- *****, opérateur binaire : multiplication.
- **/**, opérateur binaire : division. Une tentative de division par zéro ne modifie pas la pile et affiche un message d'erreur à l'utilisateur.
- **DIV**, opérateur binaire : division entière; **MOD**, opérateur binaire : modulo (reste de la division entière). Ces opérations ne peuvent s'appliquer que sur des littérales entières.
- **[optionnel] POW**, opérateur binaire : puissance.
- **NEG**, opérateur unaire : change le signe de la littérale (transforme un nombre négatif en nombre positif et vice versa). Par exemple **4 NEG** renvoie **-4**.
- **[optionnel] SIN, COS, TAN, ARCSIN, ARCCOS, ARCTAN** : opérateurs unaires trigonométriques. L'unité utilisée est le radian.
- **[optionnel] SQRT**, opérateur unaire : racine carrée.
- **[optionnel] EXP**, opérateur unaire : exponentielle.
- **[optionnel] LN**, opérateur unaire : logarithme népérien.

- **NUM**, opérateur unaire : renvoie le numérateur d'une littérale rationnelle. Appliquée à une littérale entière, cet opérateur renvoie la littérale inchangée. Provoque une erreur sur une littérale réelle ou complexe.
- **DEN**, opérateur unaire : renvoie le dénominateur d'une littérale rationnelle. Renvoie la littérale entière 1 lorsqu'il est appliqué sur une littérale entière. Provoque une erreur sur une littérale réelle ou complexe.
- **\$**, opérateur binaire : renvoie une littérale complexe constituée à partir des deux derniers éléments de la pile qui constitueront la partie réelle et la partie imaginaire du complexe.
- **RE**, opérateur unaire : renvoie la partie réelle d'une littérale complexe qui peut être une littérale entière, rationnelle ou réelle. Appliqué à une littérale entière, rationnelle ou réelle, cet opérateur renvoie la littérale inchangée.
- **IM**, opérateur unaire : renvoie la partie imaginaire d'une littérale complexe qui peut être une littérale entière, rationnelle ou réelle. Appliqué à une littérale entière, rationnelle ou réelle, cet opérateur renvoie la littérale entière 0.
- **[optionnel] ARG**, opérateur unaire renvoyant l'argument d'une littérale complexe (peut également être appliqué à une littérale entière, rationnelle ou réelle en considérant la partie imaginaire comme étant nulle).
- **[optionnel] NORM**, opérateur unaire renvoyant le module d'une littérale complexe (peut également être appliqué à une littérale entière, rationnelle ou réelle en considérant la partie imaginaire comme étant nulle).

Le type du résultat lors d'application des opérateurs numériques dépend des types des littérales utilisées dans l'opération. Par exemple :

- L'addition, la soustraction ou la multiplication de deux littérales entières donne une littérale entière.
- La division de deux littérales entières renvoie une littérale entière si le reste de la division est nul ou un rationnel s'il n'est pas nul.
- L'addition, la soustraction, la multiplication, la division entre deux littérales rationnelles renvoie une littérale rationnelle sauf si après simplification le dénominateur du résultat est égal à 1; auquel cas le type du résultat est une littérale entière.
- L'addition, la soustraction, la multiplication, la division entre une littérale entière ou rationnelle et au moins une littérale réelle renvoie une littérale réelle.
- L'addition, la soustraction, la multiplication, la division entre une littérale entière, rationnelle ou réelle et au moins une littérale complexe renvoie une littérale complexe sauf si la partie imaginaire du complexe est nulle auquel cas le type du résultat est le même que le type de la partie réelle.
- La racine carrée d'une littérale peut renvoyer une littérale entière, rationnelle, réelle ou complexe suivant le cas.

On appliquera des règles de bon sens pour les autres opérateurs.

1.4.2 Opérateurs logiques

La littérale entière 1 est utilisée pour représenter la valeur vraie et la littérale entière 0 est utilisée pour représenter la valeur faux. Toute littérale différente de la littérale entière 0 est aussi considérée comme vraie.

- **=, !=, <=, >=, <, >** : opérateurs binaires pour les tests respectivement égal, différent, inférieur ou égal, supérieur ou égal, strictement inférieur, strictement supérieur.
- **AND**, opérateur binaire : ET logique.
- **OR**, opérateur binaire : OU logique.
- **NOT**, opérateur unaire : NON logique.

1.4.3 Opérateurs pour les littérales expressions

- L'opérateur **EVAL** permet d'évaluer numériquement une littérale expression. Si la littérale expression comprend au moins un atome qui ne correspond pas au nom d'une variable, l'évaluation n'a aucun effet et un message en informe l'utilisateur. Si la littérale expression comprend au moins un atome qui correspond à l'identificateur d'un programme, l'évaluation n'a aucun effet et un message d'erreur informe l'utilisateur. Si tous les atomes de la littérales expression sont des identificateurs de variables, l'expression est évaluée numériquement.
- On peut également appliquer l'ensemble des opérateurs numériques et logiques s'écrivant sous forme de lettres sur une littérale expression. Le résultat de l'opérateur **OP** unaire appliqué à la littérale expression '**EXP**' est la littérale expression '**OP(EXP)**'. Par exemple, la ligne de commande '**3+X**'**SIN** a pour résultat **SIN(3+X)**. Le résultat de l'opérateur **OP** binaire appliqué aux littérales expressions '**EXP**' et '**EXP2**' est la littérale expression '**OP(EXP1,EXP2)**'.
- Le résultat de l'opérateur binaire utilisant un symbole (+, -, /, *, \$) appliqué entre la littérale expression '**EXP1**' et la littérale expression '**EXP2**' renvoie une littérale expression composée des deux littérales initiales, chacune entre parenthèses, les deux littérales étant séparées par le symbole. Par exemple, on obtient '**(EXP1)+(EXP2)**' si l'opérateur est +. Les parenthèses autour des expressions **EXP1** et/ou **EXP2** sont omises dans le résultat si tous les opérateurs non parenthésés dans l'expression ont la même priorité que l'opérateur. Les opérateurs *, / ont une priorité supérieure à + et -. Les opérateurs *, / ont la même priorité entre eux. Les opérateurs + et - ont la même priorité entre eux. Les opérateurs numériques ont priorité sur les opérateurs logiques. Deux opérateurs qui ont la même priorité sont toujours évalués de la gauche vers la droite. Par exemple, la ligne de commande '**3+X**' '**9*Y**'* a pour résultat '**(3+X)*9*Y**'.
- On peut aussi appliquer l'ensemble des opérateurs numériques et logiques binaires entre une littérale expression et une littérale numérique. Si l'un des deux arguments n'est pas une littérale numérique, il est d'abord transformé en littérale expression puis les règles précédentes sont appliquées.

1.4.4 Opérateurs de manipulation de la pile

- **DUP**, opérateur unaire : empile une nouvelle littérale identique à celle du sommet de la pile.
- **DROP**, opérateur unaire : dépile la littérale au sommet de la pile.
- **SWAP**, opérateur binaire : intervertit les deux derniers éléments empilés dans la pile.
- **LASTOP** : applique le dernier opérateur utilisé.
- **LASTARGS** : empile les littérales utilisées pour la dernière opération.
- **UNDO** : rétablit l'état du calculateur avant la dernière opération.
- **REDO** : rétablit l'état du calculateur avant l'application de la dernière opération **UNDO**.
- **CLEAR** : vide tous les éléments de la pile.

1.4.5 Opérateurs conditionnels et de boucle

- L'opérateur binaire **IFT** dépile 2 arguments. Le 1er (*i.e.* le dernier dépilé) est un test logique. Si la valeur de ce test est vrai, le 2^e argument est évalué sinon il est abandonné.
- **[optionnel]** L'opérateur binaire **IFTE** dépile 3 arguments. Le 1er (*i.e.* le dernier dépilé) est un test logique. Si la valeur de ce test est vrai, le 2^e argument est évalué et le 3^e argument est abandonné sinon le 3^e argument est évalué et le 2^e argument est abandonné.
- **[optionnel]** L'opérateur binaire **WHILE** dépile 2 arguments. Le 1er (*i.e.* le dernier dépilé) est un test logique. Tant que le test est vrai, le deuxième argument est évalué.

Dans ces trois opérations, on tiendra compte du fait qu'un argument peut être un programme en attente d'évaluation.

1.5 Littérales programme

- Une **littérale programme** est une suite d'opérandes séparées par des espaces. On peut le voir comme une suite d'opérandes que l'on peut éventuellement appliquer plusieurs fois. Par exemple `[1 +]` est un petit programme qui permet d'augmenter de 1 la valeur d'une littérale. Une littérale programme peut être saisie directement en ligne de commande en la commençant par le caractère `[` et la terminant par le caractère `]`.
- L'opérateur unaire **EDIT** appliqué sur un programme de la pile lance une fenêtre d'édition spacieuse pour éditer un programme.
- Une vue secondaire de l'application, permet également de créer, éditer et gérer des programmes auxquels on associe un identificateur.
- Un programme peut contenir des sous-programmes (entourés également de crochets) qui peuvent être évalués avec l'opérateur **Eval** (si il est utilisé dans le programme) ou laissés sur la pile pour une évaluation ultérieure suivant les besoins.
- Les séparateurs utilisés dans un programmes peuvent être l'espace, la tabulation ou le retour chariot. Lors de l'affichage d'un programme dans la pile, on utilisera cependant le caractère espace à la place de la tabulation et du retour chariot. Le retour chariot et la tabulation pourront être utilisés dans une fenêtre spacieuse dédiée spécifiquement à l'édition des programmes.

Exemple de programme permettant de calculer la valeur absolue du sommet de la pile:

```
[
DUP
0 <
[ NEG ]
IFT
]
```

Dans cet exemple, supposons que le programme a précédemment été enregistré en utilisant l'identificateur '**ABS**'. Supposons que l'état de la pile avant l'exécution du programme est (le **?:** symbolisant la ligne de commande) :

```
-----
4:
3:
2:
1: -56
-----
?:
```

L'utilisateur exécute le programme en saisissant **ABS** sur la ligne de commande :

```
-----
4:
3:
2:
1: -56
-----
?: ABS
```

qui est alors immédiatement évaluée et donc remplacée par le programme :

```
-----
4:
3:
2:
1: -56
-----
?: DUP 0 < [ NEG ] IFT
```

L'exécution de la suite des opérandes du programme commence alors. Le sommet de la pile est dupliquée avec l'exécution de DUP :

```
-----  
4:  
3:  
2: -56  
1: -56  
-----  
?: 0 < [ NEG ] IFT
```

La valeur est testée pour voir si elle est négative : la valeur 0 est d'abord empilée :

```
-----  
4:  
3: -56  
2: -56  
1: 0  
-----  
?: < [ NEG ] IFT
```

L'opérateur < est évalué provoquant le dépilement des deux derniers éléments (-56 et 0) et l'empilement du résultat (1) :

```
-----  
4:  
3:  
2: -56  
1: 1  
-----  
?: [ NEG ] IFT
```

La sous-programme [NEG] est alors empilé :

```
-----  
4:  
3: -56  
2: 1  
1: [ NEG ]  
-----  
?: IFT
```

La commande binaire IFT est alors évaluée, provoquant le dépilement des 2 derniers éléments de la pile (1 et [NEG]) :

```
-----  
4:  
3:  
2:  
1: -56  
-----  
?:
```

Comme 1 est la valeur vraie, [NEG] est évalué, provoquant le dépilement de -56 et l'empilement de 56 :

```
-----  
4:  
3:
```

```
2:
1: 56
```

```
-----
?:
```

1.6 Identificateurs de variables et de programmes

À toute littérale peut être associée un identificateur en utilisant l'opérateur binaire **STO**. Le premier argument est la littérale à stocker (qui peut être une littérale numérique, une littérale expression ou une littérale programme). Le deuxième est une littérale expression ne comportant qu'un atome (*i.e.* un atome entre guillemets). L'atome devient alors l'identificateur d'une variable s'il est associé à une littérale numérique ou l'identificateur d'un programme s'il est associé à une littérale programme.

On ne peut pas utiliser un identificateur égal à un opérateur prédéfini. Une tentative dans ce sens provoque l'affichage d'un message d'erreur. Si l'identificateur utilisé correspondait déjà à une autre variable ou un autre programme, la variable ou le programme est écrasé par cette nouvelle valeur.

Quand un atome est utilisé sans guillemet :

- s'il est l'identificateur d'une variable, il est remplacé par la valeur de la littérale associée;
- s'il est l'identificateur d'un programme, il provoque l'évaluation (l'exécution) du programme.

L'identificateur peut également être placé dans une littérale expression pour ne pas provoquer tout de suite son évaluation. Son évaluation est alors provoquée en utilisant l'opérateur unaire **EVAL**.

L'opérateur unaire **FORGET** permet d'effacer la variable ou le programme associé à l'atome proposé en argument. Des fenêtres secondaires de l'interface de l'application permettent de visualiser la liste des variables qui ont été créées avec l'opérateur **STO** (et qui n'ont pas encore été oubliées) pour connaître leur valeur, les éditer, ou les gérer (création, suppression). D'autres fenêtres secondaires de l'interface de l'application permettent de visualiser la liste des programmes en cours pour les lire, les éditer, ou les gérer (création, suppression).

1.7 Sauvegarde du contexte

Au démarrage de l'application, l'état de la pile, les paramétrages, les variables et les programmes du calculateur présents lors de la dernière exécution sont récupérés.

1.8 Fonctions annuler et rétablir

Le calculateur dispose des fonctions "annuler" et "rétablir". Ces fonctions peuvent être appelées par un menu, par les raccourcis Ctrl-Z et Ctrl-Y ou par les opérateurs **UNDO** ou **REDO**.

2 Livrable attendu

Le livrable est composé des éléments suivants :

- **Code source** : l'ensemble du code source du projet. Attention, ne pas fournir d'exécutable ou de fichier objet.
- **Documentation** : une documentation complète en **html** générée avec **Doxygen**.
- **Video de présentation avec commentaires audio** : une courte video de présentation dans laquelle vous filmerez et commenterez votre logiciel afin d'en démontrer le bon fonctionnement (environ 5 min). Pour réaliser cette video, vous pourrez vous servir des logiciels **Jing** (Windows, Mac OS), **RecordMyDesktop** (Linux). Ces logiciels sont mentionnés uniquement à titre d'exemple.
- **Rapport** : Un rapport en format **.pdf** composé de 2 parties :
 - la description de votre architecture;
 - une argumentation détaillée où vous montrez que votre architecture permet facilement des évolutions.

Vous pouvez ajouter en annexe de ce rapport des instructions à destination de votre correcteur si nécessaire (présentation des livrables, instructions de compilation, ...). Ce rapport ne devra pas dépasser 14 pages (annexes comprises).

L'ensemble des livrables est **à rendre pour le dimanche 12 juin 23h59 (fuseau horaire de Paris) au plus tard**. Les éléments du livrable doivent être rassemblés dans une archive **.zip**. L'archive doit être envoyée par mail aux chargés de TD suivant votre séance :

- mardi matin : Nicola Zema (nicola.zema@utc.fr).
- mardi après midi : Benoit Cantais (benoit.cantais@utc.fr).
- mercredi matin : Milan Erdelj (milan.erdelj@utc.fr).
- jeudi matin et après midi : Antoine Jouglet (antoine.jouglet@utc.fr).
- vendredi matin : Rami Kassab (rami.kassab@utc.fr).
- vendredi après midi : Lucile Callebert (lucile.callebert@utc.fr).

3 Évaluation

Le barème de l'évaluation du projet est comme suit :

- **Couverture des fonctionnalités demandées** : 5 points
- **Choix de conception et architecture** : 5 points. En particulier sera évaluée la capacité de l'architecture à s'adapter aux changements.
- **Évaluation des livrables** : 6 points (code source, documentation, vidéo, exemples de fichiers, rapport)
- **Respect des consignes sur les livrables** : 2 points (échéance, présence de l'ensemble des livrables et respect des consignes sur les livrables).
- **Présence aux séances de TD** : 2 points

Remarque : il est rappelé qu'une note inférieure ou égale à 6/20 au projet est éliminatoire pour l'obtention de l'UV.

4 Consignes

- Le projet est à effectuer en binôme (un seul monôme ou trinôme toléré par groupe de TD).
- Vous êtes libres de réutiliser et modifier les classes déjà élaborées en TD pour les adapter à votre architecture.
- En plus des instructions standards du **C++/C++11**, vous pouvez utiliser l'ensemble des bibliothèques standards du **C++/C++11** et le framework **Qt** à votre convenance.
- Il n'y a pas de contraintes concernant les éléments d'interface et d'ergonomie. Soyez créatifs. Il devrait y avoir autant d'interfaces différentes que de binômes.

5 Conseils

- Plusieurs TDs utilisent le thème du projet afin de commencer à vous familiariser avec les différentes entités de l'application qui est à développer. On ne perdra pas de vue que les questions développées dans ces TDs ne constituent pas une architecture pour le projet. Celle-ci devra être complètement retravaillée en tenant compte de l'ensemble des entités du sujet.
- La partie difficile du projet est la conception de votre architecture : c'est dessus qu'il faut concentrer vos efforts et passer le plus de temps au départ.
- Il est conseillé d'étudier au moins les design patterns suivants qui pourraient être utiles pour élaborer l'architecture de votre projet : decorator, factory, abstract factory, builder, bridge, composite, iterator, template method, adapter, visitor, strategy, facade, memento. En plus de vous donner des idées de conception, cette étude vous permettra de vous approprier les principaux modèles de conception.
- Pour la persistance des informations, vous êtes libres d'élaborer vos formats de fichier. Il est tout de même conseillé d'utiliser XML (comme pour l'export) et d'utiliser les outils XML de **Qt**.
- Au lieu d'utiliser des fichiers, vous pouvez utiliser un SGBD comme SQLite.
- L'apparence de l'application ne sera pas prise en compte dans la notation. Soyez avant tout fonctionnels. Ça peut être moche.