Sujets de mini-projet 2016

Progammation C++

VAYSSADE Jehan-Antoine

Description de votre travail et de vos choix de programmation.

1 Analyse de l’expression

Découpage:

J'ai écrit mon propre « Tokeniser » je ne voulais pas utiliser la fonction « split » définie dans le sujet, puisqu'elle engendre des copies de strings et n'est par conséquent pas optimiser, je suis passer par std::experimental::string_view, elle est toute fois disponible dans « src/Token.cpp » une des solutions pouvait être donc de découper l'expression par priorité inverse dans ce cas le passage en notation RPN n'est pas nécessaire.

Priorités :

```
template<typename ops, int p, int i>
class ExprBinary : public ExprToken
```

J'ai préférer définir deux classes template ExprUnary et ExprBinary qui respectivement me servent a définir toutes les opérations disponible dans l'application en passent des « function object » en paramètre du template, le deuxième argument est la priorité et le troisième l'index dans le buffer des opérateurs (unary_operators_name / binary_operators_name) encore une fois pour éviter les copies.

```
template <class T> struct mod : std::binary_function <T,T,T> {
   T operator() (const T& x, const T& y) const noexcept { return std::fmod(x,y); }
};
using ExprMod = ExprBinary<mod<float>, 1,  9>;
```

Ce qui me permet de réduire le code a écrire et a maintenir, mais également de pouvoir plus tard ajouter un typage des expressions et non plus uniquement des float ou encore de faire des conjonction de fonction. Utilisation également dans Context.cpp d'un template de template et définition récursive pour diminuer le « copier /coller » moins d'erreur idiote possible (impOperator)

- math.h le buffer et l’opération
- Operator.h la définition de l'Expr
- Context.cpp l'ajout dans l’interpréteur via un warper (BinaryBuilderWarper/UnaryBuilderWarper) de construction pour distinguer les expressions a construire et celle déjà construite, dans le cas d'un appelle a une fonctions définie par l'utilisateur

2 Évaluation

Ce n'est que la transposition de l'algo donner dans le tp (Expr.cpp), on push les tokens quand on trouve UNARY_BUILDER / EXT_BINARY_BUILDER on pop toutes les opérations de priorité inférieur a celle en court puis on l'ajout a la liste des opérateur a traiter. A la fin ont push les opérateurs qu'il reste, dans un std::stack.
Il ne reste plus cas remplacer les éléments par leurs évaluations, j'ai utiliser std::deque pour la possibilité d’itérer facilement et d'ajouter/remplacer/supprimer avec un coût en O(1) ce que ne permet pas de faire std::queue

3 Pour aller plus loin

Mini-projet 2016 - Phase 1

A partir de l'application développée dans le TP1, incluant les extensions de la section 3 du sujet hors représentation par un arbre, le développement demandé dans cette phase de mini-projet consiste au rajout des fonctionnalités suivantes à votre application.

Ça fait partie du Sujet 1 ou pas ?

- Ajouter la gestion des parenthèses
  - fait au début, plus simple : le block de la parenthèse est extrait en string_view et une Expr avec un nouveau Token est utiliser ce qui permet de définir d'autre opération facilement

- Gérer le cas où les tokens ne sont pas séparés par des espaces
  - fait au début, plus simple du coup les espace sont supprimer d'office

- Stocker l'expression sous forme d'un arbre
  - fait au début, j'aimerais cependant revoir les fonctions utilisateurs afin de pré-compiler la fonction dans un arbre pour optimiser l'appel
  - actuellement l'appelle d'une fonction utilisateur fait une nouvelle pass de compilation pour évaluation puisque les variables sont remplacer et non pas des pointeurs

- Définir une hiérarchie de classe pour les `Token`s en utilisant le polymorphisme pour la fonction d'évaluation.
  - fait au début : ExprNumber, ExprCall, ExprUnary, ExprBinary, Expr sont toutes dériver d'une classe virtuel pure ExprToken, idem pour les warpers.

- Gestion des nombres réels
  - fait au début : juste vérifier que le caractère suivant est '.' ...

- Capacité à évaluer une séquence d'expressions.
  - Une simple boucle for avec getline sur istringstream et faire unget() puis get() pour récupérer le caractère de terminaison, ici ' ;' du coup la fonction split() n'apporte rien, mais plutôt une perte

- Gestion d'une mémoire.
  - Cf les classes Memory et Context simplement un std::map ...

4 Pour aller plus loin, plus loin

Mini-projet 2016 - Phase 3 ?

- Ajout des exceptions pour les messages d'erreurs
- Ajout de la variable « ans » mise a jours automatique qvec la valeurs du dernier résultat calculer
- Ajout des fonctions utilisateurs
  - f(x,y,z)=x*y+z
- Ajout de la notion de Context pour l'appelle des fonctions utilisateurs
  - permet de ne pas remplacer les variables précédemment définie par celle des paramétré de la fonction appelé

- En cours de réflexion
  - Ajout du changement de Context par l'utilisateur '{expr;expr ;...}'
    - Permettre de définir des variable temporaires dans l'appel de la fonction, tous est en place il suffi de re-factoriser le main
  - Ajout des conditions et boucle ?
    - possibilité de passer par logical_or, logical_and, min, max, ...
  - Implementer l'analyse de « 7/-5 » avec '-' comme opérateur unaire
    - on peut donc supposer que le token suivant une opération binaire est forcement un nombre ou un opérateur unaire ?

- Phase 1 : Bases de la programmation C++.
  - fait ...

- Phase 2 : La bibliothèque standard du C++.
  - fait ? math / container / string / exception / pair / map / vector / deque / ...

- Phase 3 : Programmation générique et méta programmation en C++.
  - fait : template / concept / numeric_limits / binary_function / ...