COMPÈRE Nicolas
JOURDAN Léane
481C

Licence 2 Maths-Info
X22I020

Algorithmique et Structure de Données
Rapport de projet

UNIVERSITÉ DE NANTES

SDA d’un bloc

- `Bloc <C, E>` : type des blocs avec les clés `C` de type chaine de caractères et `E` les valeurs associées de type Entier.
- `est_dans(var)` : retourne vrai si `var` se trouve dans le bloc.
- `inserer(var, val)` : déclare une nouvelle variable `var` avec la valeur `val`.
- `rechercher(var)` : retourne la valeur de `var`.
- `est_vide()` : retourne vrai si le bloc est vide.
- `taille_bloc()` : retourne la taille du bloc, c’est-à-dire le nombre de variables contenu dans le bloc.
- `incremente(var)` : retourne la valeur de `var` incrémentée de 1.
- `decremente(var)` : retourne la valeur de `var` décrémentée de 1.
- `file_vide()` : retourne vrai si la file de variables est vide.
- `file_enumeration()` : retourne la première variable de la file.
- `defile()` : retire la première variable de la file.

SDC d’un bloc

Un bloc est composé de deux éléments :

- **Une table associative** : Chaque clé représente le nom d’une variable, à laquelle est associée une valeur entière unique. Une variable ne peux pas être modifiée après sa déclaration dans le bloc.
- **Un entier** : Représente le nombre de variables que contient le bloc.

L’insertion a un ordre de complexité en $\theta(1)$ amorti en moyenne. Le pire cas est en $O(n)$ avec n étant la taille de la table associative.

L’accès à la valeur d’une variable est en $\theta(1)$ en moyenne car on accède à celle-ci directement grâce à sa clé.

SDA de la structure de blocs

- `Struct_Bloc <B, E>` : type des structures de blocs constituées de `B` de type Bloc et de `E` de type Entier.
- `est_vide()` : retourne vrai si la structure de bloc est vide.
- `ouverture()` : crée un nouveau bloc au sein de la structure.
- `fermeture()`: ferme le dernier bloc de la structure.
- `tailleStructBloc()` : retourne la taille de la structure de blocs, c'est-à-dire le nombre de blocs qui constituent la structure.
- `est_dans_SB(var)` : vérifie si `var` se trouve dans le dernier bloc.
- `rechercherSB(var)` : retourne la valeur de `var`.
- `afficher(var)` : si `var` est présente dans la structure, affiche `var` et sa valeur.
- `inserer(var, val)` : insère `var` dans le dernier bloc avec la valeur `val`.
- `fermeture_V2()` : ferme le dernier bloc de la structure et affiche les variables de ce dernier dans leur ordre de déclaration.
- `incrementation(var)` : incrémente la valeur de `var`.
- `decrementation(var)` : décrémente la valeur de `var`.

SDC de la structure de bloc

La structure de bloc est composée de deux éléments :

- **Une liste de blocs**
- **Un entier** : Représente le nombre de blocs contenus dans la structure

Le bloc principal est représenté par le premier bloc de la liste. À chaque ouverture, on ajoute un bloc dans la liste. Les variables déclarées ensuite le sont uniquement dans le bloc le plus récent. Lorsque l'on veut accéder à la valeur d'une variable, on vérifie si celle ci est présente dans le bloc le plus récent de la liste. Sinon on regarde dans le bloc qui se trouve à l'indice précédent dans la liste et ainsi de suite. À la fermeture d'un bloc, on retire le dernier bloc de la liste. Cela supprime également les variables ayant été déclarées dans celui-ci.

Fonctions	Cas moyen
Ouverture	$\theta(1)$
Fermeture	$\theta(1)$
Insertion	$\theta(1)$ amorti
Affichage	$\theta(n)$
Incrementation	$\theta(n)$
Décrémentation	$\theta(n)$

Réponse à la question 5

Dans le cas où l'on souhaite afficher toutes les variables d'un bloc dans leur ordre d'apparition, on peut utiliser une file.

L'intérêt de cette structure est que la première variable entrée est la première variable sortie.

Les ordres de complexités de cette structure pour l'accès, l'insertion et la suppression d'une variable sont en $\theta(1)$.

L'implémentation de cette structure à été effectuée dans le programme (cf. `fermeture_V2()`).

Erreurs d'un programme

Beaucoup d'erreurs différentes provocants l'affichage d'un message d'erreur adéquat peuvent intervenir dans ce programme :

- Il ne faut pas oublier de commencer le programme avec une accolade ouvrante afin d'ouvrir le premier bloc qui sert de bloc principal. Autrement, le programme affiche un message d'erreur et le programme ASD2 s'arrête et revient au terminal.

- Un espace manquant entre le nom de la variable et sa valeur lors de sa déclaration ou entre un symbole valide du programme et le nom de la variable.

- L'affichage d'une variable qui n'a pas été déclarée au préalable

- La déclaration d'une variable avec une valeur négative n'est pas possible. Cependant, il est possible d'obtenir des valeurs négatives pour les variables grâce à la fonction de décrémentation (cf. `decrementation(var)`).

Augmentations du langage ASD2

Deux nouvelles instructions ont été implémentées dans le langage ASD2 :

- Incrémentation : Il est possible d'incrémenter de 1 la valeur d'une variable en tapant la commande `+ var`.

- Décrémentation : Permet de décrémenter de 1 la valeur d'une variable en tapant la commande `- var`.

Pour implémenter ces nouvelles commandes, il a fallu créer :

- Deux nouveaux tokens dans `parser.cpp` et `parser.hpp` → `TOKEN_INCREMENTE` et `TOKEN_DECREMENTE` et les boucles de comparaison associées

- Deux nouvelles fonctions dans `bloc.cpp` et `bloc.hpp` → `incremente(var)` et `decremente(var)` avec `var` la variable à incrémenter ou décrémenter

- Deux nouvelles fonctions dans `struct_bloc.cpp` et `struct_bloc.hpp` → `incrementation(var)` et `decrementation(var)` avec `var` la variable à incrémenter ou décrémenter. Ces fonctions font appel à ceux de `bloc.cpp` et `bloc.hpp`

Les cas de ces tokens à également été implémentés dans `main.cpp`.

Afficher la liste des variables à la fermeture d'un bloc

L'affichage des variables à la fermeture d'un bloc est effectué par la procédure `fermeture_V2()`. C'est cette fonction qui est implémentée dans `main.cpp` pour `TOKEN_CLOSE`.

Affichage des numéros de lignes

L'affichage des numéros de lignes se fait grâce à une variable entière `numLigne` initialisée à 1. Grâce à la classe `iomanip` de C++, on délimite un espace réservé à l'affichage des numéros grâce à la fonction `setw(a)` avec `a` une valeur entière correspondant au nombre d'espaces réservés. Cela permet de ne pas avoir un décalage de l'indentation lors du passage de la ligne 9 à 10, 99 à 100,… jusqu'à une certaine limite qui se situe vers 10 000.

Annexe

Pour la bonne indentation du programme lors de l'ouverture et la fermeture d'un bloc, deux fichiers `tabulation.cpp` et `tabulation.hpp` sont utilisés.

La seule procédure présente dans ce fichier et `tabulation(t)` avec `t` une valeur entière correspondant au nombre de blocs contenus dans la structure de bloc et donc au nombre de tabulations nécessaires.