Πανεπιστήμιο Πειραιώς

Ημερομηνία: 7 Ιανουαρίου 2024

Σύνταξη από: Αντώνιος Τσαλμπούρης, Π22272 Δημήτριος Λαζάνας, Π22082

Μεταγλωττιστές - Εργασία 2023-24

Θέμα 1ο

Αρχείο Λύσης: `thema1.c`

Ζήτημα: Υλοποίηση ενός Ντετερμινιστικού Αυτόματου Στοίβας που αναγνωρίζει εκφράσεις υπό τους εξής όρους:

- Ίδιος αριθμός χαρακτήρων “x” και “y”.
- Κατά την ανάγνωση από αριστερά προς δεξιά, οι χαρακτήρες “y” δεν πρέπει ποτέ να υπερβαίνουν τους “x”.
- Εκτύπωση της αλληλουχίας βημάτων για αναγνώριση ή απόρριψη της έκφρασης.

Επιλογή Γλώσσας: C++

Υλοποίηση

Class AutomatoStoivas

- **Περιγραφή:** Η κλάση AutomatoStoivas διαθέτει δομή στοίβας “char” και περιλαμβάνει μεθόδους για επεξεργασία και αναγνώριση (ή απόρριψη) των εκφράσεων, καθώς και για την εκτύπωση των ακολουθηθέντων βημάτων.
- **Συνάρτηση processExpression():** Δέχεται την έκφραση από την `main()`. Για κάθε χαρακτήρα `"x"` προσθέτει ένα `"x"` στη στοίβα, ενώ για κάθε `"y"` ελέγχει τη στοίβα και αφαιρεί ένα `"x"` εφόσον ικανοποιούνται οι όροι. Σε περίπτωση ασυμφωνίας, τυπώνεται μήνυμα απόρριψης.
- **Συνάρτηση printStack():** Εκτυπώνει τα περιεχόμενα της στοίβας, χρησιμοποιώντας βοηθητική στοίβα για την εμφάνιση των στοιχείων της.

main()

- **Περιγραφή:** Δημιουργεί ένα αντικείμενο της κλάσης AutomatoStoivas, ζητά την έκφραση από τον χρήστη, καλεί την `processExpression()` για τους απαραίτητους ελέγχους και τερματίζει το πρόγραμμα.

Παραδείγματα Εκτέλεσης

1. Παράδειγμα Αναγνώρισης

```
Type expression: xxyyxyxy
Push X --> Stack: x
Push X --> Stack: xx
Pop X --> Stack:  x
Pop X --> Stack:
Push X --> Stack: x
Pop X --> Stack:
Push X --> Stack: x
Pop X --> Stack:
Expression Accepted!
```

2. Δύο Παραδείγματα Απόρριψης

```
Type expression: yxxy
Expression Rejected: 'y' does not match to a 'x'
```

```
Type expression: xxyyy
Push X --> Stack: x
Push X --> Stack: xx
Pop X --> Stack:  x
Pop X --> Stack:
Expression Rejected: 'y' does not match to a 'x'
```

Η δομή και η λειτουργία του προγράμματος περιγράφονται αναλυτικά, με σαφήνεια και οργάνωση, προσφέροντας μια συνοπτική και κατανοητή εικόνα του έργου και των λειτουργιών του.

Θέμα 2ο

Αρχείο Λύσης: `randomStringGenerator.c`

Εισαγωγή Βιβλιοθηκών

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

Ανάλυση κάθε Βιβλιοθήκης

1. `<stdio.h>`

Η βιβλιοθήκη `<stdio.h>` (Standard Input/Output Header) χρησιμοποιείται για την εκτέλεση εισόδου και εξόδου στο πρόγραμμα. Συγκεκριμένα, αυτή η βιβλιοθήκη παρέχει δυνατότητες για την εκτύπωση στην οθόνη (μέσω της συνάρτησης `printf`) και για την ανάγνωση εισόδου από τον χρήστη αν χρειαστεί. Στο συγκεκριμένο πρόγραμμα, χρησιμοποιείται για την εμφάνιση μηνυμάτων και των παραγόμενων αλφαριθμητικών.

2. `<stdlib.h>`

Η `<stdlib.h>` (Standard Library Header) περιέχει συναρτήσεις για διαχείριση μνήμης, αλλά και για την παραγωγή τυχαίων αριθμών, που είναι βασικό για τη λειτουργία αυτού του προγράμματος. Η συνάρτηση `rand()` χρησιμοποιείται για την παραγωγή τυχαίων αριθμών που καθορίζουν την επιλογή των ακολουθιών κατά τη δημιουργία του αλφαριθμητικού.

3. `<time.h>`

Η βιβλιοθήκη `<time.h>` περιλαμβάνει συναρτήσεις για τη διαχείριση και την απόκτηση πληροφοριών για χρόνο και ημερομηνία. Σε αυτό το πρόγραμμα, χρησιμοποιείται για την αρχικοποίηση του σπόρου της τυχαίας αριθμοποίησης μέσω της συνάρτησης `srand(time(NULL))`. Αυτό εξασφαλίζει ότι οι τυχαίοι αριθμοί που παράγονται είναι διαφορετικοί κάθε φορά που εκτελείται το πρόγραμμα.

Κάθε βιβλιοθήκη που έχει επιλεγεί για αυτό το πρόγραμμα παίζει συγκεκριμένο ρόλο στην υλοποίηση και την αποδοτικότητα του κώδικα, συμβάλλοντας στην ολοκληρωμένη λειτουργία και την αλληλεπίδραση με τον χρήστη.

Δήλωση Συναρτήσεων

```c
void generate_Z(char *result, int *pos, int depth);
void generate_K(char *result, int *pos, int depth);
void generate_G(char *result, int *pos, int depth);
void generate_M(char *result, int *pos, int depth);
```

Δηλώνονται οι συναρτήσεις που χρησιμοποιούνται για την παραγωγή τυχαίων αλφαριθμητικών.

Κύρια Συνάρτηση

```c
int main()
{
    srand(time(NULL));
    char result[1000] = {0};
    int pos = 0;
    generate_Z(result, &pos, 0);
    printf("Generated string: %s\n", result);
    return 0;
}
```

Η `main()` αρχικοποιεί το περιβάλλον, τοποθετεί τον σπόρο για την τυχαία παραγωγή αριθμών και καλεί την `generate_Z()` για να ξεκινήσει τη δημιουργία του αλφαριθμητικού.

Συνάρτηση generate_Z

```
void generate_Z(char *result, int *pos, int depth)
{
    ...
}
```

Η `generate_Z` αρχικοποιεί το αλφαριθμητικό με έναν αριστερό παρενθετικό και καλεί την `generate_K`.

Συνάρτηση generate_K

```
void generate_K(char *result, int *pos, int depth)
{
    ...
}
```

Η `generate_K` καλεί τις `generate_G` και `generate_M` για να συνεχίσει τη δημιουργία του αλφαριθμητικού.

Συνάρτηση generate_G

```
void generate_G(char *result, int *pos, int depth)
{
    ...
}
```

Η `generate_G` προσθέτει είτε έναν χαρακτήρα 'v' εί

τε καλεί αναδρομικά την `generate_Z`.

Συνάρτηση generate_M

```
void generate_M(char *result, int *pos, int depth)
{
    ...
}
```

Η `generate_M` μπορεί να προσθέσει τους χαρακτήρες '-' ή '+' και να καλέσει αναδρομικά την `generate_K` ή να μην προσθέσει τίποτα (επιστρέφοντας το κενό αλφαριθμητικό).

Αυτή η τεκμηρίωση παρέχει μια επισκόπηση των βασικών λειτουργιών και της δομής του προγράμματος. Ο κάθε κώδικας εξηγείται με λεπτομέρεια για να κατανοήσουν οι αναγνώστες τη λογική και τη ροή του προγράμματος.

Θέμα 3

Αρχείο Λύσης: `stringAnalyzer.cpp`

Πρόβλημα

Υλοποίηση συντακτικού αναλυτή top-down για την αναγνώριση ή απόρριψη συμβολοσειρών με βάση τη δοσμένη γραμματική. Η γραμματική έχει οριστεί ως εξής:

- G→ (M)
- M→ YZ
- Y→ a | b | G
- Z→ *M | -M | +M| ε (ε = κενή συμβολοσειρά)

Ο αναλυτής επιστρέφει το σχετικό δέντρο και εκτυπώνει την ανάλυση, με παράδειγμα την έκφραση ((a-b)*(a+b)).

Εισαγωγή Βιβλιοθηκών

```
#include <iostream>
#include <map>
#include <string>
#include <set>
#include <vector>
#include <iomanip> // Για τη διαμόρφωση της εξόδου στην κονσόλα.
```

Ανάλυση Βιβλιοθηκών

- `<iostream>`: Χρήση για είσοδο/έξοδο στο πρόγραμμα (π.χ., `std::cout`).
- `<map>`: Δομή δεδομένων map για την αντιστοίχιση κλειδιών με τιμές.
- `<string>`: Διαχείριση συμβολοσειρών στο C++.
- `<set>`: Δομή συνόλου για αποθήκευση μοναδικών τιμών.
- `<vector>`: Δυναμικός πίνακας για αποθήκευση στοιχείων.
- `<iomanip>`: Παρέχει λειτουργίες για τη διαμόρφωση της εξόδου (π.χ., `std::setw` για τον καθορισμό του πλάτους εξόδου).

Κύριες Συναρτήσεις και Δομές

`Node` Class

```
#ifndef NODE_H
#define NODE_H

#include <iostream>
#include <vector>

class Node {
public:
    Node(char name);
    ~Node();
```

```
        void push(const std::string &production);

        Node *getNextNode();
        char getName();
        std::vector<Node *> getChildren();
        void print

    Children();

    private:
        char name;
        Node *parent;
        Node *nextNode;
        std::vector<Node *> *children;

        void setParent(Node *parent);
        void addChild(Node *child);
        Node *climbUp();
        Node *searchNextNode(Node *currentNode);
        void setNextNode();
    };

    #endif // NODE_H
```

Η κλάση `Node` αναπαριστά έναν κόμβο στο δέντρο ανάλυσης. Κάθε κόμβος περιέχει ένα χαρακτήρα που αντιπροσωπεύει ένα μη τερματικό ή τερματικό στοιχείο της γραμματικής, καθώς και δείκτες προς τον γονέα και τα παιδιά του. Η κλάση παρέχει λειτουργίες για την προσθήκη παιδιών, την αναζήτηση επόμενων κόμβων, και την εκτύπωση των παιδιών για την οπτικοποίηση της δομής του δέντρου.

`initializeRulesTable()`

Αρχικοποιεί τον πίνακα κανόνων `rulesTable`, που περιέχει τους κανόνες της γραμματικής.

`initializeTerminalCharacters()`

Αρχικοποιεί το σύνολο `terminalCharacters` με τους τερματικούς χαρακτήρες της γραμματικής.

`isTerminal(char character)`

Ελέγχει αν ένας χαρακτήρας είναι τερματικός σύμφωνα με το σύνολο `terminalCharacters`.

`M(const std::string &nonTerminal, const std::string &character)`

Επιστρέφει την παραγωγή από τον πίνακα κανόνων βάσει ενός μη τερματικού και ενός χαρακτήρα.

`generateIsEmpty(const std::string& production)`

Ελέγχει αν μια παραγωγή είναι κενή μετά τον χαρακτήρα '>'.

`stackPush(std::vector<char>& stack, const std::string& production)`

Εισάγει χαρακτήρες μιας παραγωγής στη στοίβα.

`stackPop(std::vector<char>& stack, bool print)`

Αφαιρεί τον επάνω χαρακτήρα από τη στοίβα και προαιρετικά εκτυπώνει τη στοίβα.

`terminate()`

Τερματίζει την εκτέλεση με μήνυμα απόρριψης της συμβολοσειράς.

`parse(const std::string& input)`

Αναλύει μια δοθείσα συμβολοσειρά και εκτυπώνει αν αυτή αναγνωρίζεται ή όχι.

`main(int argc, char* argv[])`

Η κύρια συνάρτηση του προγράμματος εκκίνησης. Ακολουθούνται τα εξής βήματα:

1. **Αρχικοποίηση Κανόνων και Χαρακτήρων:** Καλεί τις `initializeRulesTable()` και `initializeTerminalCharacters()` για την αρχικοποίηση των κανόνων της γραμματικής και του συνόλου τερματικών χαρακτήρων αντίστοιχα.
2. **Χειρισμός Ορισμάτων Γραμμής Εντολών:** Έλεγχος αν έχει δοθεί ένα όρισμα μέσω της γραμμής εντολών (π.χ., `./stringAnalyzer "(a-b)*(a+b)"`). Αν ναι, χρησιμοποιεί αυτό το όρισμα ως την είσοδο για την ανάλυση.
3. **Λήψη Εισόδου Από Χρήστη:** Εάν δεν έχει δοθεί όρισμα γραμμής εντολών, ζητείται από τον χρήστη να εισάγει μια συμβολοσειρά.
4. **Χρήση Προεπιλεγμένης Συμβολοσειράς:** Αν ο χρήστης δεν παρέχει είσοδο, τότε το πρόγραμμα χρησιμοποιεί μια προεπιλεγμένη συμβολοσειρά για την ανάλυση, η οποία είναι `"((a-b)*(a+b))"`.
5. **Κλήση της `parse()`**: Η συμβολοσειρά (είτε από το όρισμα εντολής, είτε από τον χρήστη, είτε η προεπιλεγμένη) δίνεται στην `parse()` για ανάλυση.

Η `main()` είναι σχεδιασμένη για να είναι ευέλικτη στην επεξεργασία διαφορετικών συμβολοσειρών, επιτρέποντας τόσο τη διαδραστική λειτουργία με τον χρήστη όσο και την αυτοματοποιημένη δοκιμή μέσω ορισμάτων γραμμής εντολών.

Η τεκμηρίωση παρέχει λεπτομερή επισκόπηση της λογικής και της δομής του προγράμματος, βοηθώντας τον αναγνώστη να κατανοήσει τον τρόπο λειτουργίας του συντακτικού αναλυτή σε C++.