**《数据结构》**

**课程设计报告**

|  |  |
| --- | --- |
| 课程名称： | 《数据结构》课程设计 |
| 课程设计题目： | 魔王语言 |
| 姓 名： | 易子顺 |
| 院 系： | 计算机学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 23052314 |
| 学 号： | 23050411 |
| 指导教师： | 葛瑞泉 |

2019年12月25日

1. **需求分析**

#### 功能需求：

用规则B→tAdA和A→sae实现一个魔王语言的解释系统，把他的话解释成人能听得懂的语言。

#### 界面需求：

魔王的语言是依据以下两种形式的规则由人的语言逐步抽象上去：

（1）α→β1β2…β3

（2）

设大写字母表示魔王语言的词汇；小写字母表示人的语言词汇；希腊字母表示可以用大写字母或小写字母代换的变量。魔王语言可含人的词汇。

#### 接口设计

char \*get\_demon\_lang();

//backend

char \*translate(char \*input, int size);

数据结构设计

//stack

typedef struct{

    char \*stack;

    int size;

    int top;

    char \*name;

} Stack;

Stack\* init\_stack(int size, char \*name);

char pop(Stack \*s);

char nopop(Stack \*s);

void push\_s(Stack \*s, char\* c);

void push\_c(Stack \*s, char c);

bool is\_empty(Stack \*s);

char \*stack2str(Stack \*s);

**详细设计**

**#include <demon.h>**

**/\***

**用规则B→tAdA和A→sae实现一个魔王语言的解释系统,把他的话解释成人能听得懂的语言。**

**魔王的语言是依据以下两种形式的规则由人的语言逐步抽象上去：**

**(1)a→β1β2…β3**

**(2)(θδ1δ2…δn)->θδnθδn-1… θδ1θ**

**设大写字母表示魔王语言的词汇；小写字母表示人的语言词汇；希腊字母表示可以用大写字母或小写字母代换的变量。魔王语言可含人的词汇。**

**\*/**

**int main(){**

**char \*demon\_lang = get\_demon\_lang();**

**//char \*demon\_lang = "B(ehnxgz)B";**

**assert(demon\_lang);**

**unsigned int size = strlen(demon\_lang);**

**char \*trans = translate(demon\_lang, size);**

**assert(trans);**

**//print result**

**printf("\nRESULT:\n");**

**printf("\033[1;32m" "%s -> %s\n" "\033[0m", demon\_lang, trans);**

**//assert(strcmp(trans, "tsaedsaeezegexenehetsaedsae"));**

**return 0;**

**}**

**#include <demon.h>**

**int len = 100;**

**char \*get\_demon\_lang(){**

**char \*lang = malloc(sizeof(char) \* 100);**

**assert(lang != NULL);**

**char c;**

**int i = 0;**

**while((c = getchar()) != '\n' && c != EOF){**

**lang[i++] = c;**

**if(i > len-1){**

**lang = realloc(lang, sizeof(char) \* (len += 100));**

**assert(lang != NULL);**

**}**

**}**

**lang[i] = '\0';**

**#ifdef DEBUG**

**printf("lang: %s size = %lu\n", lang, strlen(lang));**

**#endif**

**//check if it is a valid demon language**

**Stack \*s = init\_stack(100, "lang");**

**for(int i = 0; i < strlen(lang); i++){**

**if(lang[i] == '(')**

**push\_c(s, '(');**

**else if(lang[i] == ')')**

**if(pop(s) != '(') {panic("invalid demon language");};**

**}**

**if(!is\_empty(s)){**

**panic("invalid demon language");**

**}**

**return lang;**

**}**

**#include <demon.h>**

**char \*reverseString(const char \*str);**

**char \*append\_c(char c, Stack \*s);**

**char \*translate\_internal(Stack \*input\_stack, Stack \*result\_stack, Stack \*attach);**

**char \*translate(char \*input, int size)**

**{**

**Stack \*input\_stack = init\_stack(size \* 100, "input stack");**

**Stack \*result\_stack = init\_stack(size \* 100, "result stack");**

**Stack \*attach = init\_stack(size \* 100, "attach stack");**

**push\_s(input\_stack, reverseString(input));**

**return translate\_internal(input\_stack, result\_stack, attach);**

**}**

**char \*translate\_internal(Stack \*input\_stack, Stack \*result\_stack, Stack \*attach)**

**{**

**char c;**

**while((c = pop(input\_stack)) != -1){**

**switch (c)**

**{**

**case 'A':**

**push\_s(input\_stack, "sae");**

**break;**

**case 'B':**

**push\_s(input\_stack, reverseString("tAdA"));**

**break;**

**case '(':**

**if(nopop(input\_stack) != ')'){**

**Stack \*tmp = init\_stack(input\_stack->size, "tmp stack");**

**push\_c(attach, pop(input\_stack));**

**char c\_tmp;**

**//TODO:嵌套有bug**

**while((c\_tmp = pop(input\_stack)) != ')'){**

**if(c\_tmp == '('){**

**push\_c(input\_stack, '(');**

**push\_s(tmp, translate\_internal(input\_stack, result\_stack, attach));**

**}**

**push\_c(tmp, c\_tmp);**

**}**

**push\_c(input\_stack, ')');**

**char \*tmp\_str = stack2str(tmp);**

**Stack \*tmp2 = init\_stack(tmp->size, "tmp2 stack");**

**push\_s(tmp2, reverseString(tmp\_str));**

**while((c\_tmp = pop(tmp2)) != -1){push\_c(input\_stack, c\_tmp);}**

**}**

**break;**

**case ')':**

**push\_c(result\_stack, pop(attach));**

**break;**

**default:**

**push\_s(result\_stack, append\_c(c, attach));**

**break;**

**}**

**}**

**return stack2str(result\_stack);**

**}**

**char \*append\_c(char c, Stack \*s)**

**{**

**#ifdef DEBUG**

**printf("\nAPPEND: %c\n", c);**

**#endif**

**push\_c(s, c);**

**char \*str = stack2str(s);**

**pop(s);**

**#ifdef DEBUG**

**printf("\n");**

**#endif**

**return str;**

**}**

**char \*reverseString(const char \*str) {**

**int len = strlen(str);**

**char \*reversed = malloc((len + 1) \* sizeof(char)); // 为新字符串分配内存**

**if (reversed == NULL) {**

**return NULL; // 内存分配失败时返回 NULL**

**}**

**// 复制并反转字符串**

**for (int i = 0; i < len; i++) {**

**reversed[i] = str[len - i - 1];**

**}**

**reversed[len] = '\0'; // 添加字符串终止符**

**return reversed;**

**}**

**用户手册**

