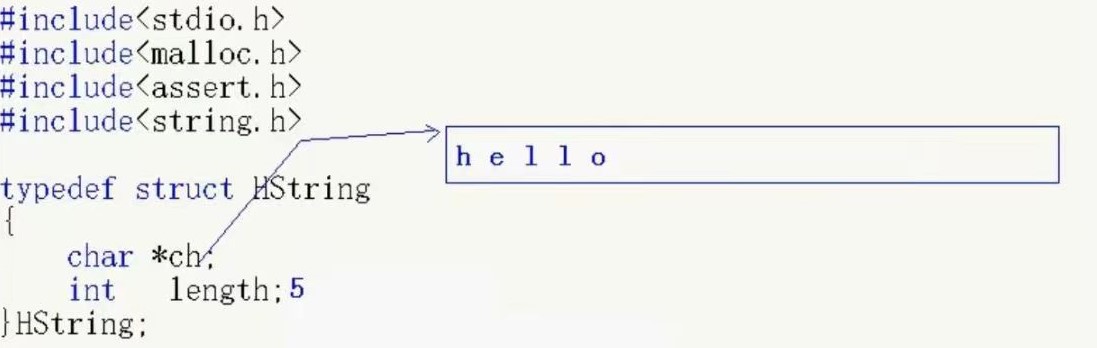
**串结构之堆分配存储实现**



#include <stdio.h>

#include <string.h>

#include<malloc.h>

#include<assert.h>

typedef struct HString {

char\* ch;

int length;

}HString;

void InitString(HString \*S) {

S->ch = NULL;

S->length = 0;

}

void PrintString(HString\* S) {

for (int i = 0; i < S->length; i++) {

printf("%c", S->ch[i]);

}

printf("\n");

}

void StrAssign(HString\*S,const char\* str) { //用一个字符串对定长字符串进行赋值

int len = strlen(str);

if (S->ch != NULL) {

free(S->ch);

}

S->ch = (char\*)malloc(sizeof(char) \* len);

assert(S->ch != NULL);

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

S->ch[i] = str[i];

}

S->length = len;

}

int StrLength(HString \*S) { //求长度

return S->length;

}

void StrCopy(HString\*T, HString\*S) { //拷贝函数

int len = StrLength(S);

if (T->ch != NULL) {

free(T->ch);

}

T->ch= (char\*)malloc(sizeof(char) \* len);

assert(T->ch != NULL);

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

T->ch[i] = S->ch[i];

}

T->length = len;

}

bool StrEmpty(HString \*S) { //判空函数

return S->length == 0;

}

int StrCompare(HString\* S, HString \*T) { //比较函数

if (S->length == 0 && T->length == 0) {

return 0;

}

int result = 0;

int i = 0;

int j = 0;

while (i < S->length && j < T->length) {

if (S->ch[i] > T->ch[j]) {

return 1;

}

else if (S->ch[i] < T->ch[j]) { return -1; }

else {

i++;

j++;

}

}

if (i < S->length) {

result = 1;

}

if (j < T->length) {

result = -1;

}

return result;

}

void StrConcat(HString\*T, HString \*S1, HString \*S2) { //链接函数

if (T->ch != NULL);

free(T->ch);

int len1 = StrLength(S1);

int len2 = StrLength(S2);

T->ch = (char\*)malloc(sizeof(char) \* (len1+len2));

assert(T->ch != NULL);

for (int i = 0; i < len1; i++) {

T->ch[i] = S1->ch[i];

}

for (int j = 0; j < len2; j++) {

T->ch[len2 + j] = S2->ch[j];

}

T->length = len1 + len2;

}

void SubString(HString\*S, HString\* sub ,int pos,int len) { //求子串函数

if (pos < 0 || pos >= S->length || len<0 || len>S->length - pos) {

return;

}

if (sub->ch != NULL);

free(sub->ch);

sub->ch= (char\*)malloc(sizeof(char) \* len);

assert(sub->ch != NULL);

int j = pos;

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

sub->ch[i] = S->ch[pos + i];

}

sub->length = len;

}

void StrInsert(HString\*S, int pos, HString\* T) { //插入函数

if (T->length == 0) {

return;

}

if (pos<0 || pos>S->length) {

return;

}

S->ch = (char\*)realloc(S->ch, sizeof(S->length + T->length));

assert(S->ch != NULL);

for (int i = S->length - 1; i >= pos; i--) {

S->ch[i + T->length] = S->ch[i];

}

int j = pos;

for (int i = 0; i < T->length; i++) {

S->ch[j + i] = T->ch[i];

}

S->length = S->length + T->length;

}

void StrDelete(HString\* S,int pos,int len) { //删除函数

if (pos < 0 || pos >= S->length) {

return;

}

if (len <= 0 || len > S->length) {

return;

}

int j = pos;

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

S->ch[j + i] = S->ch[j + i + len];

}

S->length -= len;

}

void StrClear(HString \*S) { //清除函数

S->length = 0;

if (S->ch != NULL) {

free(S->ch);

}

S->ch = NULL;

}

void main() {

HString S;

InitString(&S);

StrAssign(&S, "abc");

HString T;

InitString(&T);

StrAssign(&T, "xyz");

int res = StrCompare(&S, &T);

HString X;

InitString(&X);

StrConcat(&X, &S, &T);

PrintString(&X);

HString sub;

InitString(&sub);

SubString(&X, &sub, 2, 3);

PrintString(&sub);

StrInsert(&S, 1, &T);

PrintString(&S);

}