Nội dung

* Bảng ký tự tổng quát
* Cài đặt và hoàn thiện phonebook dùng tìm kiếm tuần tự và phiên bản tìm kiếm nhị phân

Chú ý.

Truyền tham số trong hàm

**Cách 1.**

PhoneEntry \* getPhoneNumber(char \* name, **PhoneBook\* book**)

{

for(int i=0; i<book->total; i++)

if(strcmp(name, book->entries[i])==0) return book->entries[i];

return NULL;

}

Và

**Cách 2.**

PhoneEntry \* getPhoneNumber(char \* name, **PhoneBook book**)

{

for(int i=0; i<book.total; i++)

if(strcmp(name, book.entries[i])==0) return book.entries[i];

return NULL;

}

Cách 1: Truyền vào địa chỉ của PhoneBook thông qua con trỏ

Tham số cho hàm chỉ là con trỏ trỏ tới địa chỉ ô nhớ thực sự của biến → kích thước tham số chỉ 4->8 Byte (nhỏ)

Mọi tác động trong hàm liên quan đến ghi/ cập nhật giá trị sẽ ảnh hưởng trực tiếp tới giá trị của biến thực sự. Nếu chỉ cho phép đọc giá trị thay vì có thể thay đổi giá trị tham số, dùng thêm từ khóa const

PhoneEntry \* getPhoneNumber(char \* name, **const PhoneBook\* book**)

Cách 2. Truyền vào 1 bản copy của biến (có kích thước bằng kích thước của biến thực sự)

* Nếu bản copy chỉ gồm biến thông thường (KHÔNG phải kiểu con trỏ) : Bản copy và bản gốc là độc lập nhau về bộ nhớ → thay đổi với bản copy sẽ không tác động lên biến gốc
* Nếu bản copy là con trỏ (các bản copy đều cùng trỏ tới địa chỉ ô nhớ là giống nhau): Thay đổi nội dung ô nhớ thông qua bản copy nào thì đều sẽ tác động đến giá trị thực sự.

Cách 2 truyền tham số sẽ tốn bộ nhớ hơn so với cách 1

Kết luận:

* Cách 2 dùng khi truyền tham số với kiểu tham số là các biến đơn giản, kích thước nhỏ (VD. biến đơn int, float, double,...)
* Cách 1 dùng khi truyền tham số cho hàm là các dữ liệu kích thước lớn (VD. kiểu struct ….)

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**typedef struct {**

**char name[80];**

**long number;**

**} PhoneEntry;**

**typedef struct {**

**PhoneEntry \* entries;**

**int total;**

**int size;**

**} PhoneBook;**

**#define INITIAL\_SIZE 5**

**#define INCREMENTAL\_SIZE 5**

**PhoneBook createPhoneBook()**

**{**

**PhoneBook pb;**

**pb.total = 0;**

**pb.size = INITIAL\_SIZE;**

**pb.entries = (PhoneEntry\*)malloc(sizeof(PhoneEntry)\*pb.size);**

**return pb;**

**}**

**void dropPhoneBook(PhoneBook\* book)**

**{**

**free(book->entries);**

**book->entries = NULL;**

**book->total=0;**

**book->size=0;**

**}**

**// them 1 ban ghi moi vao book**

**void addPhoneNumber(char \* name, long number, PhoneBook\* book)**

**{**

**if(book->size>book->total)// chua day, co the them**

**{**

**strcpy((book->entries[book->total]).name, name);**

**(book->entries[book->total]).number = number;**

**book->total++;**

**}**

**else // cap phat dong mang moi, copy sang mang moi dung memcpy**

**{**

**// tao mang moi kich thuoc bang mang cu cong them INCREMENTAL\_SIZE**

**PhoneEntry \* newBook = (PhoneEntry\*)malloc(sizeof(PhoneEntry)\*(book->size+INCREMENTAL\_SIZE));**

**// copy phan tu mang cu sang mang moi**

**for(int i=0; i<book->total; i++)**

**memcpy(&newBook[i],&(book->entries[i]),sizeof(PhoneEntry));**

**// xoa mang cu**

**free(book->entries);**

**// cap nhat lai thong tin cua phonebook moi**

**book->entries = newBook;**

**book->size = book->size + INCREMENTAL\_SIZE;**

**// them phan tu moi vao cuoi**

**strcpy((book->entries[book->total]).name, name);**

**(book->entries[book->total]).number = number;**

**book->total++;**

**}**

**}**

**void printAllPhoneBook(const PhoneBook\* book)**

**{**

**printf("Danh ba hien tai co %d phan tu, kich thuoc toi da %d\n",book->total, book->size);**

**for(int i=0; i<book->total; i++)**

**printf("%20s\t%15ld\n", (book->entries[i]).name, (book->entries[i]).number);**

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

**}**

**// phonebook duoc them vao ko theo thu tu ho getPhoneNumber**

**// tim kiem chi don gian la tuan tu**

**// tra ve gia tri NULL neu khong tim thay**

**// tra ve dia chi phan tu neu tim thay**

**PhoneEntry \* getPhoneNumber(char \* name, const PhoneBook\* book)**

**{**

**for(int i=0; i<book->total; i++)**

**if(strcmp(name, (book->entries[i]).name)==0) return &(book->entries[i]);**

**return NULL;**

**}**

**int main()**

**{**

**PhoneBook pb;**

**pb = createPhoneBook();**

**addPhoneNumber("Nguyen Van Mau",123456789,&pb);**

**addPhoneNumber("Nguyen Van 1",123456789,&pb);**

**addPhoneNumber("Nguyen Van 2",123456789,&pb);**

**addPhoneNumber("Nguyen Van 3",123456789,&pb);**

**addPhoneNumber("Nguyen Van 4",123456789,&pb);**

**addPhoneNumber("Nguyen Van 5",123456789,&pb);**

**printAllPhoneBook(&pb);**

**PhoneEntry \* ret = getPhoneNumber("Nguyen Van 2",&pb );**

**if(ret==NULL) printf("Khong ton tai trong danh ba!\n");**

**else**

**printf("tim thay thong tin : %20s\t%15ld\n", ret->name, ret->number);**

**return 0;**

**}**

**Bài tập 1.**

Sửa lại việc thêm phần tử: Thêm theo đúng thứ tự chữ cái để có thể áp dụng được hàm tìm kiếm nhị phân

* PA1. Thực hiện thêm như tuần tự, tuy nhiên trước khi tìm kiếm nhị phân sẽ phải gọi lại hàm sắp xếp. Để tránh phải gọi hàm sắp xếp nhiều lần khi tìm kiếm thì mỗi lần thêm phần tử mới xong chúng ta sẽ gọi hàm sắp xếp.
* PA2. Mỗi lần thêm, ta thêm đúng thứ tự họ tên (sử dụng thuật toán tương tự thuật toán chèn) → Không cần gọi hàm sắp xếp, tuy nhiên hàm addPhoneNumber sẽ phải xử lý nhiều hơn.

**Bài tập 2**.

Sửa lại hàm tìm kiếm phần tử: Tìm kiếm theo 1 phần của tên

VD. danh bạ hiện có

Nguyen van A

Le thi B

Tran xuan C

Nguyen Le D

Khi người dùng tìm với từ khóa “Nguyen” thì sẽ in ra cả Nguyen van A và Nguyen Le D

Dùng hàm **strstr** thay vì **strcmp** khi tìm kiếm

Chú ý: Không phân biệt hoa/thường

Tìm và in ra chứ không trả về, vì **strstr** có thể trả về nhiều hơn 1 kết quả

========================

Bảng dữ liệu tổng quát

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define INITIAL\_SIZE 5

#define INCREMENTAL\_SIZE 5

typedef struct {

void \* key;

void \* value;

} Entry;

typedef struct {

Entry \* entries;

int size, total;

Entry (\*makeNode)(void\*, void\*);

int (\*compare)(void\*, void\*);

} SymbolTable;

Entry makePhone(void\* name, void\* phone) {

Entry res;

res.key = strdup( (char\*)name );

res.value = malloc(sizeof(long));

memcpy( res.value, phone, sizeof(long) );

return res;

}

int comparePhone(void \* key1, void\* key2) {

return strcmp((char\*)key1, (char\*)key2);

}

SymbolTable createSymbolTable(Entry (\*makeNode)(void\*, void\*),

int (\*compare)(void\*, void\*)

)

{

SymbolTable spb;

spb.size = INITIAL\_SIZE;

spb.total = 0;

spb.entries = (Entry\*)malloc(sizeof(Entry)\*spb.size);

return spb;

}

void dropSymbolTable(SymbolTable\* book)

{

for(int i=0; i<book->total; i++)

{

free(book->entries[i].key);

free(book->entries[i].value);

}

free(book->entries);

book->entries = NULL;

book->total=0;

book->size=0;

}

void addEntry(void\* key, void\* value, SymbolTable\* book)

{

if(book->size>book->total)// chua day, co the them

{

//strcpy((book->entries[book->total]).name, name);

//(book->entries[book->total]).number = number;

book->entries[book->total] = makePhone(key, value);

book->total++;

}

}

//Entry\* getEntry(void\* key, SymbolTable book);

void printAllPhoneBook(const SymbolTable\* book)

{

printf("Danh ba hien tai co %d phan tu, kich thuoc toi da %d\n",book->total, book->size);

for(int i=0; i<book->total; i++)

printf("%20s\t%15ld\n", (char\*)(book->entries[i]).key, \*((long\*)(book->entries[i]).value));

printf("==================================\n");

}

int main()

{

SymbolTable phoneBook = createSymbolTable(makePhone, comparePhone);

long number = 983984775;

char name[] = "Ta Tuan Anh";

addEntry(name, &number, &phoneBook);

printAllPhoneBook(&phoneBook);

return 0;

}

**Bài tập 3.**

Hoàn thiện hàm tìm kiếm tuần tự với kiểu dữ liệu tổng quát

Entry\* getEntry(void\* key, SymbolTable book)