**Kỹ Thuật Lập Trình**

**Học kỳ 2 – Năm học 2017-2018**

**Lab 8 – Hàm**

# MỤC TIÊU CỦA BÀI THỰC HÀNH

* Xác định được phần mô tả hàm cho một công việc
* Viết được phần hiện thực hàm để làm một công việc
* Hiểu và sử dụng các kiểu truyền tham số.
* Hiểu và sử dụng mảng, con trỏ với hàm
* Sử dụng được kỹ thuật đệ quy để giải bài toán trong thực tế

# BÀI TẬP BẮT BUỘC

**Câu 1:**

Cho chương trình

#include <iostream>

using namespace std;

int main(){

int a = 9;

cout << “before:” << “a =” << a << endl;

thay\_doi1(a);

thay\_doi2(&a);

cout << “after:” << “a =” << a << endl;

return 0;

}

void thay\_doi1(int& x){

x = 100;

}

void thay\_doi2(int\* x){

\*x = 100;

}

Khi biên dịch chương trình báo lỗi, vì bộ biên dịch không tìm thấy khai báo cho các hàm ở các lời gọi hàm sau: thay\_doi1(a); và thay\_doi2(a);

1. Không cần phải copy cả hai hàm trên lên phía trước hàm main(). Có cách nào khác?
2. Nếu bỏ dấu & trong phần định nghĩa hàm “thay\_doi1” nói trên, thì kết quả của biến a trước và sau khi gọi hàm thay\_doi1(a); có khác không vì sao?
3. Thay dòng: int a = 9; bằng phát biểu: int\* a = new int(9);

Cũng như thêm dòng: delete a; vào trước dòng có lệnh return.

Hãy thay 2 lời gọi hàm: thay\_doi1(a); và thay\_doi2(&a); bằng cách lời gọi khác cho phù hợp để thay đổi giá trị **mà a chỉ đến** (hiện tại là 9) thay cho chính địa chỉ được giữ trong a.

1. Nếu đổi tên để hai hàm đang có đều là “thay\_doi” thay cho “thay\_doi1” và “thay\_doi2”. Đổi luôn lời gọi hàm sang “thay\_doi” tương ứng. Biên dịch có lỗi không, vì sao? Khái niệm tiếng Anh nào chỉ ra khả năng các hàm có thể cùng tên, nhưng khác chữ ký hàm?
2. Bổ sung luôn hàm “thay\_doi”, nhưng truyền thông số bằng trị. Biên dịch chương báo lỗi không? Lỗi ở khâu nào? Vì sao?

**Câu 2:**

Viết chương trình cho phép nhập vào số nguyên bất kỳ. Xuất ra màn hình cách đọc bằng tiếng Anh tương ứng với số đó.

Chương trình phải giải quyết được cả trường hợp số nguyên nhập vào là âm hoặc dương.

Ví dụ:

0 -> zero

10 -> ten

121 -> one hundred twenty one

-1032 -> negative one thousand thirty two

11043 -> eleven thousand forty three

-1200000 -> negative one million two hundred thousand

**Hướng dẫn:**

* Định nghĩa prototype của các hàm cần thiết

void num\_to\_text(int num);

void tens\_to\_text(int tens);

void power\_to\_text(int power);

int get\_num\_digits(int n); //Sinh viên tự hiện thực

void translateHundred(int hundred\_chunk);//Sinh viên tự hiện thực

void translateThousand(int thousand\_chunk);//Sinh viên tự hiện thực

* Hàm **num\_to\_text** để xử lý các trường hợp chữ số hàng đơn vị nói chung (bao gồm hàng đơn vị, hàng nghìn, hàng triệu, hàng tỷ)

void num\_to\_text(int num) {

switch(num) {

case 1: printf("one"); break;

case 2: printf("two"); break;

case 3: printf("three");break;

case 4: printf("four"); break;

case 5: printf("five"); break;

case 6: printf("six"); break;

case 7: printf("seven");break;

case 8: printf("eight");break;

case 9: printf("nine"); break;

case 10:printf("ten");break;

case 11:printf("eleven");break;

case 12:printf("twelve");break;

case 13:printf("thirteen");break;

case 14:printf("fourteen");break;

case 15:printf("fifteen");break;

case 16:printf("sixteen");break;

case 17:printf("seventeen");break;

case 18:printf("eighteen");break;

case 19:printf("nineteen"); break;

default: printf("");

}

}

* Hàm **tens\_to\_text** để xử lý các chữ số hàng chục nói chung (bao gồm hàng chục, hàng chục nghìn, hàng chục triệu)

void tens\_to\_text(int tens) {

switch (tens) {

case 2: printf("twenty"); break;

case 3: printf("thirty"); break;

case 4: printf("forty"); break;

case 5: printf("fifty"); break;

case 6: printf("sixty"); break;

case 7: printf("seventy"); break;

case 8: printf("eighty"); break;

case 9: printf("ninety"); break;

default: printf("");

}

}

* Hàm **power\_to\_text** để xử lý chèn các tên gọi “thousand”, “million”, “billion” giữa các hàng nghìn và hàng trăm, hàng triệu và hàng trăm nghìn, hàng tỷ và hàng trăm triệu.

void power\_to\_text(int power)

{

switch(power) {

case 1: printf("thousand "); break;

case 2: printf("million ");break;

case 3: printf("billion "); break;

default: printf("");

}

}

* Hàm **get\_num\_digits** trả về số lượng chữ số của 1 số nguyên (sinh viên tự hiện thực hàm này)
* Ngoài ra, sinh viên cũng hiện thực hai hàm **translateHundred** và **translateThounsand**.
* Hàm main như sau:

int main()

{

int n;

scanf("%d", &n);

if ( n == 0 ) {

printf("zero");

}

if ( n < 0 )

{

printf("negative ");

n \*= -1;

}

int num\_digits = get\_num\_digits(n);

int chunk\_count = (num\_digits % 3 == 0) ? (num\_digits /3) : (

num\_digits /3) + 1;

int t;

while (chunk\_count > 0) {

t = pow(10, (chunk\_count - 1)\*3);

translateThousand(n / t);

if (chunk\_count > 1) printf(" ");

power\_to\_text(chunk\_count - 1);

n = n % t;

chunk\_count--;

}

return 0;

}

**Câu 3: Viết chương trình trộn bộ bài Tây (hay còn gọi là bài lơ khơ)**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <iostream>

using namespace std;

typedef struct Card\_t

{

int nSuit;

char strNumber[3];

}Card;

Card card[52];

void Init(){ }

void ShowCard(){ }

void Shuffle(){ }

int main()

{

Init();

cout << "Init state: " << endl;

ShowCard();

Shuffle();

cout << "After shuffling: " << endl;

ShowCard();

return 0;

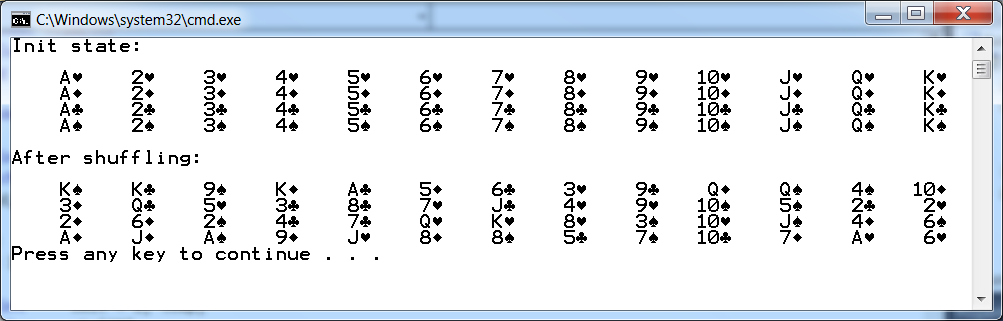
}

Chương trình trên có chức năng xáo trộn bộ bài tây. Sinh viên hoàn thiện các hàm Init(), ShowCard() và Shuffle().

Bộ bài Tây gồm 52 quân bài, mỗi quân bài gồm có chất (cơ, rô, nhép, bích) và số (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K). Thông tin của quân bài được lưu bằng kiểu dữ liệu Card. Trong đó nSuit lưu chất, strNumber lưu số.

* Hàm Init() khởi động trạng thái ban đầu của bộ bài.
* Hàm ShowCard() hiển thị bộ bài.
* Hàm ShuffleCard() trộn bộ bài.

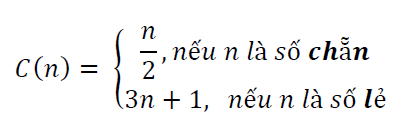
Kết quả chương trình như hình sau:



**Câu 4: Viết chương trình với các hàm sau:**

* Hàm **bool IsPrime(int n)** dùng để kiểm tra xem số nguyên n có phải là số nguyên tố hay không.
* Hàm **bool IsAmstrong(int n)** dùng để kiểm tra xem số nguyên n có phải là số Armstrong hay không. Số Armstrong là số có giá trị bằng tổng lập phương các chữ số của nó. Ví dụ 153 là số Armstrong, vì 13 + 53 + 33 = 153.
* Hàm **bool IsPerfectNumber(int n)** dùng để kiểm tra xem số nguyên n có phải là số hoàn hảo hay không. Số hoàn hảo là số có giá trị bằng tổng các ước nguyên dương của chính nó. Ví dụ 28 là số hoàn hảo, vì 1 + 2 + 4 + 7 + 14 = 28, trong đó 1, 2, 4, 7, 14 là các ước nguyên dương của 28.
* Trong hàm main(), yêu cầu người sử dụng nhập vào hai số nguyên dương n1 và n2, với n1 < n2. Sau đó chương trình sẽ in ra:
  + Tất cả các số nguyên tố trong đoạn [n1, n2]
  + Tất cả các số Armstrong trong đoạn [n1, n2]
  + Tất cả các số hoàn hảo trong đoạn [n1, n2]

**Câu 5:** Dùng **phương pháp đệ quy** để thực hiện các hàm sau:

1. Tính tổng và tích các phần tử trong mảng bằng đệ quy.
2. In các giá trị trong mảng theo thứ tự thuận hay nghịch bằng đệ quy.
3. Viết hàm đệ quy tìm giá trị lớn nhất của một mảng.
4. Đổi một số thập phân sang nhị phân bằng đệ quy.
5. Chuỗi số Hailstone là các số tự nhiên cho bởi: 

Ví dụ với n = 6 ta được chuỗi:

6, 3, 10, 5, 16, 8, 4, 2, 1, ….

Viết chương trình đệ quy tìm dãy số Hailstone bắt đầu từ **n** và kết thúc tại giá trị 1 với n nhập từ bàn phím. Kết quả output bao gồm dãy số Hailstone và độ dài của dãy số.

VD output của chương trình:

Input the number : 13

The hailstone sequence starting at 13 is:

13 40 20 10 5 16 8 4 2 1

The length of the sequence is 10.