Chương 6
Hàm

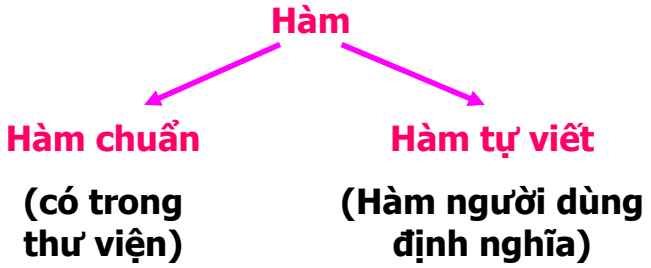
Nội dung chương này

6.1. Giới thiệu

- Có những đoạn chương trình được thực hiện lặp đi lặp lại nhiều lần, tuy dữ liệu có khác nhưng bản chất các công việc lại giống nhau.
- →Viết gộp những đoạn chương trình đó lại thành một chương trình con mà khi cần chỉ việc truyền dữ liệu cho nó?
- Tư tưởng đó cũng dẫn chúng ta tới việc chia một chương trình lớn thành nhiều phần nhỏ rồi giải quyết từng phần; sau đó sẽ ráp nối chúng lại là sẽ hoàn tất một chương trình lớn. Các chương trình nhỏ này trong C chính là các *hàm* (function).
- Như vậy một chương trình sẽ là một tập hợp các định nghĩa hàm riêng biệt.

6.2. Module chương trình trong C

- Các module trong C được gọi là hàm.



- Hàm: viết một lần, dùng lại ở nhiều nơi.

6.2. Module chương trình trong C (tiếp)

- Các hàm được gọi để thực hiện bằng các *lời gọi hàm.*
- Các lời gọi hàm xác định tên của hàm và cung cấp các thông tin (hay còn gọi là các tham số) mà hàm được gọi theo đúng trình tự khi khai báo hàm đó.

6.3. Các hàm toán học trong C

- Khai báo tệp tiêu đề #include<math.h>
- Khi dùng chỉ việc viết lời gọi hàm
- Ví dụ: viết hàm tính và in ra căn bậc 2 của 900.0 là : *printf("%.2f", sqrt(900.0));*
  - sqrt là hàm khai căn bậc 2
  - số 900.0 là tham số của hàm sqrt
  - hàm sqrt nhận tham số kiểu double và trả về giá trị kiểu double
  - Câu lệnh sẽ in ra 30.00

6.3. Các hàm toán học trong C (tiếp)

- Các tham số của hàm có thể là các hằng, biến hay các biểu thức
- Ví dụ nếu c1 = 13.0, d = 3.0 thì câu lệnh

  *printf("%.2f", sqrt(c1 + d * 4.0));*

  sẽ tính và in ra căn bậc 2 của 13.0 + 3.0 * 4.0 = 25.0 là 5.00

6.3. Các hàm toán học trong C (tiếp)

Hàm	Mô tả	Ví dụ
sqrt(x)	Căn bậc 2 của x	sqrt(9.00) bằng 3.0
exp(x)	hàm mũ e^x	exp(1.0) bằng 2.718282
log(x)	logarithm tự nhiên (cơ số e) của x	log(2.718282) bằng 1.0
log10(x)	logarithm thập phân (cơ số 10) của x	log10(1.0) bằng 0.0 log10(10.0) bằng 1.0
fabs(x)	giá trị tuyệt đối của x	nếu x≥0 thì fabs(x) bằng x nếu x<0 thì fabs(x) bằng -x
ceil(x)	làm tròn thành số nguyên nhỏ nhất lớn hơn x	ceil(9.2) bằng 10.0 ceil(-9.8) bằng -9.0

6.3. Các hàm toán học trong C (tiếp)

floor(x)	làm tròn thành số nguyên lớn nhất nhỏ hơn x	floor(9.2) bằng 9.0 floor(-9.8) bằng -10.0
pow(x,y)	x mũ y (x^y)	pow(2,7) bằng 128
fmod(x,y)	phần dư của phép chia x cho y	fmod(13.657,2.333) bằng 1.992
sin(x)	sin của x (x theo radian)	sin(0.0) bằng 0.0
cos(x)	cos của x (x theo radian)	cos(0.0) bằng 1.0
tan(x)	tan của x (x theo radian)	tan(0.0) bằng 0.0

6.4. Hàm người dùng định nghĩa

- Xét bài toán: tính bình phương của 10 số tự nhiên từ 1 đến 10;
- Ta sẽ tổ chức chương trình thành 2 hàm hàm main() và một hàm nữa mà ta đặt tên là square(); nhiệm vụ của hàm square() là tính bình phương của một số nguyên; còn hàm main() là sẽ gọi hàm **square()** để thực hiện nhiệm vụ được đề ra.

6.4. Hàm người dùng định nghĩa (tiếp)

```
#include<stdio.h>
#include<conio.h>
int square(int) /* Hàm nguyên mẫu */
void main()
{
   int x;
   for (x = 1; x <= 10; x++)
   printf( "%d   ", square(x));
   printf("\n");
   getch();
}
int square(int y) /* Định nghĩa hàm square */
{
   return   y * y;
}
```

Kết quả

1 4 9 16 25 36 49 64 81 100

6.4. Hàm người dùng định nghĩa (tiếp)

- Khuôn dạng một hàm:

kiểu tên-hàm (danh-sách-tham-số)
{
Các khai báo
...
Các câu lệnh
[return [bieu_thuc];]
}

6.4. Hàm người dùng định nghĩa (tiếp)

- Giải thích:
- Ở đây *kiểu* là kiểu dữ liệu của kết quả của hàm , *tên-hàm* là một cái tên do người lập trình tự đặt;
- *Danh-sách-tham-số* gồm danh sách các tham số hàm nhận được khi nó được gọi và được ngăn cách nhau bởi dấu phẩy.
- Các khai báo và các câu lệnh trong ngoặc là *phần thân của hàm*. Khai báo và cài đặt một hàm lại không được nằm trong một hàm khác trong bất kỳ tình huống nào.

6.4. Hàm người dùng định nghĩa (tiếp)

- Hàm có thể có giá trị trả về hoặc không có giá trị trả về.
  - Nếu có giá trị trả về, trong thân hàm có ít nhất một lệnh **`return`**.
  - Nếu không có giá trị trả về cần khai báo cho hàm đó có kiểu trả về là **`void`**.

6.4. Hàm người dùng định nghĩa (tiếp)

- Ví dụ:

```
int giai_thua(int a)                  ------> Dong dau ham
{
   int ket_qua;                       ------> Cac khai bao
   int i;

   ket_qua = 1;
   for(i = 1;i<=a;i++)
     ket_qua = ket_qua * i;           ------> Cac cau lenh
   if(a < 0) ket_qua = -1;
   if(a == 0) ket_qua = 1;
   return ket_qua;
}
```


6.4. Hàm người dùng định nghĩa
Ví dụ thứ ba

- Xây dựng hàm maximum trả về số nguyên lớn nhất trong ba số nguyên.

```
#include<stdio.h>
#include<conio.h>
int maximum(int, int, int);   /* Hàm nguyên mẫu */
void main() /* Hàm main */
{
  int a, b, c, largest;
  printf( " Vào 3 số nguyên : ");
  scanf( " %d%d%d ", &a, &b, &c);
  largest = maximum(a,b,c);
  printf( " Số lớn nhất là : %d\n", largest);
  getch();
}
```

6.4. Hàm người dùng định nghĩa
Ví dụ thứ ba (tiếp)

```
/* Định nghĩa hàm maximum */
int maximum(int x, int y, int z)
{
  int max = x;
  if (y > max)
  max = y;
  if (z > max)
  max = z;
  return  max;
}
```

- Kết quả

Vào 3 số nguyên : 22 85 17
Số lớn nhất là : 85
Vào 3 số nguyên : 85 22 17
Số lớn nhất là : 85

6.5. Nguyên mẫu hàm (Prototype)

- Một trong những đặc trưng quan trọng của ANSI C là hàm nguyên mẫu (khai báo hàm). Hàm nguyên mẫu thông báo cho trình biên dịch biết kiểu dữ liệu hàm trả lại, số lượng, kiểu và trình tự của các tham số truyền cho hàm. Trình biên dịch dùng hàm nguyên mẫu để kiểm tra các lời gọi hàm.
- Quan sát lại ví dụ đầu tiên:

6.5. Nguyên mẫu hàm (tiếp)

```
#include<stdio.h>
#include<conio.h>
int square(int); /* Hàm nguyên mẫu */
void main()
{
   int x;
   for (x = 1; x <= 10; x++)
   printf( "%d   ", square(x));
   printf("\n");
   getch();
}
int square(int y) /* Định nghĩa hàm square */
{
   return   y * y;
}
```


6.6. Các tệp header

- Mỗi một thư viện chuẩn tương ứng có một tệp header chứa các khai báo của tất cả các hàm trong thư viện này cùng với các định nghĩa các kiểu dữ liệu khác nhau, các hằng dùng trong các hàm đó
- Lập trình viên có thể tạo các tệp header riêng của mình. Các tệp header này cũng thường có kiểu .h và có thể dùng chỉ thị tiền xử lý #include để thêm vào chương trình.

6.7. Truyền tham số cho hàm

- Có hai cách để truyền các tham số cho hàm:
  - Khi các tham số được truyền theo trị, một bản sao giá trị của tham số được tạo ra và truyền cho hàm. Vì vậy mọi sự thay đổi trong hàm trên bản sao sẽ không ảnh hưởng đến giá trị ban đầu của biến nằm trong hàm gọi.
  - Khi một tham số được truyền theo con trỏ, hàm gọi sẽ truyền trực tiếp tham số đó cho hàm bị gọi thay để đổi giá trị nguyên thuỷ của biến truyền vào tham số.
- Trong C, tất cả các tham số được truyền đều là truyền theo trị.
- Ta có thể mô phỏng cách truyền tham số theo tham chiếu bằng cách truyền địa chỉ của các biến vào tham số.

6.7. Truyền tham số cho hàm - ví dụ

- Viết hàm hoán đổi hai số nguyên
- Hãy quan sát hai cách sau

6.7. Truyền tham số cho hàm - ví dụ

- **Cách 1:**

```
#include<stdio.h>
void swap(int, int);     /* Hàm nguyên mẫu */
void main(){
   int x, y;
   x = 10; y = 20;
   printf("Ban đầu x = %d, y = %d", x, y);
   swap(x, y);
   printf("Sau đó x = %d, y = %d", x, y);
}
void swap( int x, int y){    /* Định nghĩa hàm swap */
   int temp;
   temp = x;
   x = y;
   y = temp;
}
```

Liệu có thực sự hoán đổi?

6.7. Truyền tham số cho hàm - ví dụ

- **Cách 2:**

```
#include <stdio.h>
void swap( int* , int* );     /* Hàm nguyên mẫu */
void main(){
   int x, y;
   x = 10; y = 20;
   printf("Ban đầu x = %d, y = %d", x, y);
   swap(&x, &y);
   printf("Sau đó x = %d, y = %d", x, y);
}
void swap( int* x, int* y){    /* Định nghĩa hàm swap */
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
}
```

Truyền địa chỉ của biến vào vị trí các tham số

6.7. Truyền tham số cho hàm-nhận xét

- Nếu đối của hàm là con trỏ kiểu int, float, double,... thì tham số thực sự tương ứng trong lời gọi hàm phải là địa chỉ của biến hoặc địa chỉ của phần tử mảng có kiểu int, float, double... Khi đó, địa chỉ của biến được truyền cho đối con trỏ tương ứng. Do đã biết được địa chỉ của biến nên có thể gán cho biến các giá trị mới.
- Các đối của hàm có hai loại: các đối chứa dữ liệu đã biết (gọi là đối vào), các đối chứa kết quả mới nhận được (đối ra). Các đối ra phải khai là các đối con trỏ.

6.7. Truyền tham số cho hàm

- Hàm có đối là mảng một chiều: Hai cách khai báo hàm:
  - void Tên_hàm(float *pa,...): pa là một con trỏ.
  - void Tên_hàm(float pa[],...): pa coi như một mảng hình thức.
  - Nếu tham số thực là tên mảng a một chiều kiểu float thì ta gọi **Tên_hàm(a,...)**.
  - Khi hàm bắt đầu làm việc thì giá trị của a (địa chỉ phần tử a[0]) được truyền cho pa. Do đó, muốn truy nhập tới phần tử a[i] trong thân hàm ta dùng *(pa+i) hoặc pa[i].
  - Với các kiểu char, int, double, cấu trúc,...ta cũng làm tương tự như trên.

6.7. Truyền tham số cho hàm

- Hàm có đối là mảng 2 chiều: Giả sử a là mảng 2 chiều float a[20][30]. Ta có thể khai báo theo hai cách sau:
  - Cách 1:
    - void Tên_hàm(float (*pa)[50], int m, int n)
    - void Tên_hàm(float pa[][50], int m, int n)
    - pa là con trỏ kiểu float[50] (chứa địa chỉ đầu của vùng nhớ dành cho 50 số thực, tức là vùng nhớ 200 byte), m là số hàng và n là số cột
    - Trong thân hàm, để truy nhập tới phần tử hàng i cột j, ta dùng kí hiệu pa[i][j].
    - Theo cách này, ta chỉ làm việc với mảng 2 chiều có tối đa 50 cột. Lời gọi hàm **Tên_hàm(a, m, n)**

6.7. Truyền tham số cho hàm

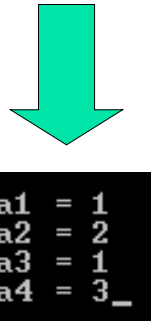
- Hàm có đối là mảng 2 chiều
  - Cách 2:
    - float *pa; biểu thị địa chỉ đầu của mảng a
    - int N; biểu thị số cột của mảng a
    - Khai báo: Tên_hàm(float *pa, int N,...)
    - Lời gọi: Tên_hàm(a, 30,...)
    - Trong thân hàm, để truy nhập tới phần tử a[i][j] ta dùng công thức *(pa + i*N + j)
    - Theo cách này, mảng 2 chiều quy về mảng 1 chiều, hàm có thể dùng cho bất kỳ mảng 2 chiều nào.

6.8. Phạm vi biến

- Biến địa phương (Local Variable):
  - Là các biến được khai báo trong lệnh khối hoặc trong thân chương trình con.
- Biến toàn cục (Global Variable):
  - Là biến được khai báo trong chương trình chính.
  - Vị trí khai báo của biến toàn cục là sau phần khai báo tệp tiêu đề và khai báo hàm nguyên mẫu

6.8. Phạm vi biến - ví dụ 1:

```
#include <stdio.h>
void main(){
   {
   int a = 1;
   printf("\n a1 = %d",a);
   {
   int a = 2;
   printf("\n a2 = %d",a);
   }
   printf("\n a3 = %d",a);
   }
   {
   int a = 3;
   printf("\n a4 = %d",a);
   }
}
```



6.8. Phạm vi biến - ví dụ 2:

```
#include <stdio.h>
#include <conio.h>
int a, b, c;
int tich()
{
   printf("\n Gia tri cac bien tong the a, b, c: ");
   printf(" a = %-5d b = %-5d c = %-5d", a, b, c);
   return a*b*c;
}

void main()
{
   clrscr();
   printf("\n Nhap gia tri cho 3 so nguyen a, b, c: ");
   scanf("%d %d %d",&a,&b,&c);
   printf("\n Tich cua 3 so la %d",tich());
   getch();
}
```

```
Nhap 3 so nguyen a, b, c: 2 3 4

Gia tri cac bien tong the a, b, c:
a = 2     b = 3     c = 4
Tich cua 3 so la 24
```


Biến register

- Thanh ghi có tốc độ truy nhập nhanh hơn so với các loại bộ nhớ khác (RAM, bộ nhớ ngoài).
- Nếu một biến thường xuyên sử dụng được lưu vào trong thanh ghi thì tốc độ thực hiện của chương trình sẽ được tăng lên.
- Để làm điều này ta đặt từ khóa **`register`** trước khai báo của biến đó.
  - Ví dụ: **`register int a;`**
- Số lượng và kích thước các thanh ghi có hạn → Số lượng biến khai báo **`register`** sẽ không nhiều và thường chỉ áp dụng với những biến có kích thước nhỏ như kiểu **char**, **int**.

Biến static

- Một biến cục bộ khi ra khỏi phạm vi của biến đó thì bộ nhớ dành để lưu trữ biến đó sẽ được giải phóng.
- Nếu cần lưu giá trị của các biến cục bộ này, cần khai báo biến với từ khóa **`static`**.
  - Ví dụ: **`static int a;`**
- Biến **`static`** là biến tĩnh, nghĩa là nó sẽ được cấp phát một vùng nhớ thường xuyên từ lúc khai báo và chỉ giải phóng khi chương trình chính kết thúc.

Ví dụ

```
# include <stdio.h>
# include <conio.h>
void fct()
{
   static int count = 1;
   printf("\n Day la lan goi ham fct lan thu %2d",
   count++);
}
void main()
{
   int i;
   for(i = 0;  i < 10; i++)
       fct();
   getch();
}
```

```
Day la lan goi ham fct lan thu  1
Day la lan goi ham fct lan thu  2
Day la lan goi ham fct lan thu  3
Day la lan goi ham fct lan thu  4
Day la lan goi ham fct lan thu  5
Day la lan goi ham fct lan thu  6
Day la lan goi ham fct lan thu  7
Day la lan goi ham fct lan thu  8
Day la lan goi ham fct lan thu  9
Day la lan goi ham fct lan thu 10_
```


6.9. Đệ quy

- Trong C cho phép trong thân một hàm có thể gọi ngay đến chính nó, cơ chế này gọi là đệ qui.
- Có thể định nghĩa hàm đệ qui là hàm sẽ gọi đến chính nó trực tiếp hay gián tiếp thông qua các hàm khác.
- Cách tiến hành giải một bài toán đệ qui nhìn chung có những điểm chung sau.
  - Có trường hợp cơ sở
  - Nó gọi đến chính dạng của nó nhưng có sự chuyển biến.
  - Sự chuyển biến ấy đến một lúc nào đó phải kết thúc.

6.9. Đệ quy - ví dụ 1

- Ta xem xét ví dụ tính n giai thừa (**n!**).
  - Theo định nghĩa n! bằng tích của tất cả các số tự nhiên từ 1 đến n: n! = n * (n-1) ... 2 * 1
  - Ví dụ: 5! = 5 * 4 * 3 * 2 * 1 = 120, 0! được định nghĩa bằng 1.
  - Để tính n! có thể dùng vòng lặp như sau:

```
fact = 1;
for (i = n; i >= 1; i--)
fact *= i;
```

  - Tuy nhiên cũng có thể định nghĩa đệ qui hàm giai thừa như sau :

nếu n>0 thì n! = n * (n-1)!

nếu n=0 thì n! = 0! = 1

6.9. Đệ quy - ví dụ 1

```
/* Tính giai thừa theo phương pháp đệ qui */
#include <stdio.h>
long fact( long );    /* Hàm nguyên mẫu */
main()
{
  int i;
  for ( i=0; i<=10; i=i+2)
  printf("%2d! = %ld\n", i, fact(i));
}
/* Định nghĩa đệ qui hàm factorial */
long fact( long n)
{
  if ( n == 0 )return 1;
  else return (n*fact(n-1));
}
```


6.9. Đệ quy - ví dụ 2

- Một ví dụ thứ hai dùng đệ qui là tính dẫy số Fibonaci
  - Dãy số Fibonaci gồm những số

    0, 1, 1, 2, 3, 5, 8, 13, 21 ...
  - Bắt đầu từ hai số 0 và 1, tiếp sau đó các số Fibonaci sau bằng tổng của 2 số Fibonaci trước nó.
  - Dãy Fibonaci có thể định nghĩa đệ qui như sau:

    $fibonaci(0) = 0; fibonaci(1) = 1;$

    $fibonaci(n) = fibonaci(n-1) + fibonaci(n-2)\ \forall n>1$

6.9. Đệ quy - ví dụ 2

```
/* Tính dẫy số Fibonaci phương pháp đệ qui */
#include <stdio.h>
long fibo( long );    /* Hàm nguyên mẫu */
void main(){
  long result, n;
  printf("Hãy nhập vào một số nguyên : ");
  scanf("%ld", &n);
  result = fibo(n);
  printf("Fibonaci thứ %ld là : %ld\n", n, result);
}
/* Định nghĩa đệ qui hàm fibonaci */
long fibo( long n){
  if ( n == 0 || n == 1 )return n;
  else return fibo(n-1) + fibo(n-2);
}
```


So sánh đệ quy và lặp

- So sánh:
  - Lập trình đệ quy sử dụng cấu trúc lựa chọn, còn lặp sử dụng cấu trúc lặp.
  - Cả hai phương pháp đều phải kiểm tra khi nào thì kết thúc. Phương pháp lặp kết thúc khi điều kiện để tiếp tục vòng lặp sai, còn phương pháp đệ quy kết thúc khi đến trường hợp cơ sở
  - Tuy nhiên đệ quy tồi hơn, nó liên tục đưa ra các lời gọi hàm làm tốn thời gian xử lý của bộ vi xử lý và không gian nhớ.
- "*Công nghệ phần mềm*" là quan trọng nhưng *hiệu năng của chương trình* cũng quan trọng và thật không may hai mục tiêu đó lại thường loại trừ lẫn nhau.
- Vậy khi module hoá chương trình phải khôn ngoan cân đối giữa hai mục tiêu là *Công nghệ phần mềm* và *Hiệu năng*

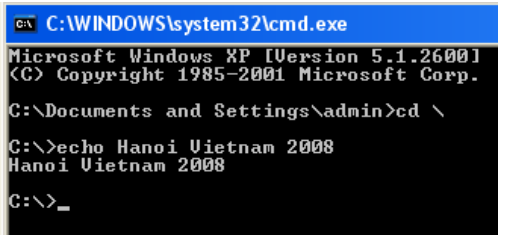
6.10. Hàm main có đối dòng lệnh

- Giả sử ta có chương trình echo.exe và chạy nó theo dòng lệnh:

  C:\>echo Hanoi 2008
- Khi đó, "Hanoi" và "2008" là các tham số dòng lệnh.
- Hàm main có 2 tham số:
  - argc (argument count): số lượng đối số dòng lệnh
  - argv (argument vector): con trỏ trỏ tới mảng xâu kí tự, mảng này chứa các đối dòng lệnh, mỗi đối là một xâu
- Khi gõ vào từ dòng lệnh, các tham số phải cách nhau bởi dấu cách. Chương trình xử lý các tham số dòng lệnh như các dữ liệu vào.

Mã nguồn chương trình echo.c

```
#include <stdio.h>
int main(int argc, char *argv[]){
   int i;
   for(i=1; i<argc; i++)
   printf("%s%s", argv[i],(i<argc-1)?" ":" ");
   printf("\n");
   return 0;
}
```



Bài tập

- Bài 1: Viết hàm tìm ước số chung lớn nhất uscln(int x, int y) và bội số chung nhỏ nhất bscnn(int x, int y) của hai số.
- Bài 2: Lập hàm giải phương trình bậc 2 ptb2(float a, float b, float c, float *x1, float *x2), trong đó a,b,c là các hệ số của phương trình. Hàm nhận giá trị 0 nếu phương trình vô nghiệm, giá trị 1 nếu phương trình có nghiệm kép (chứa trong *x1), giá trị 2 nếu phương trình có 2 nghiệm (chứa trong *x1 và *x2), giá trị 3 nếu phương trình có vô số nghiệm.

Bài tập

- Bài 3: Lập hàm tính giai thừa và dùng hàm này tính tổng:

  1! + 2! +...+N! (N nhập từ bàn phím và <=10)
- Bài 4: Đa thức cấp n được tính theo công thức:

  $f(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x^1 + a_0 x^0$
  - Lập hàm đa thức f(float a[], int n, float x) để tính giá trị của đa thức.

Bài chữa - Bài 1:

```
#include<stdio.h>
#include<conio.h>
int uscln(int, int);
int bscnn(int, int);
int main(int argc, char *argv[])
{
    int x, y;
    printf("Nhap x = ");scanf("%d",&x);
    printf("\nNhap y = ");scanf("%d",&y);
    printf("\nUSCLN = %d\n",uscln(x,y));
    printf("\nBSCNN = %d\n",bscnn(x,y));
    getch();
    return 0;
}
```


Bài chữa - Bài 1 (tiếp)

```
int uscln(int a,int b){
    while(a!=b) if(a>b)a-=b; else b-=a;
    return a;
}
int bscnn(int a, int b){
    int max,i;
    if(a>b)max=a; else max=b;
    for(i=max;i<=a*b;i++)
       if((i%a==0)&&(i%b==0))return i;
}
```


Bài chữa – Bài 3:

```
#include<stdio.h>
#include<conio.h>
long gt(int n){
    int i;
    long gt=1;
    for(i=1;i<=n;i++) gt*=i;
    return gt;
}
```

Bài chữa – Bài 3 (tiếp)

```
int main(int argc, char *argv[])
{
    int i, n;
    long t=0;;
    do{
        printf("Nhap n = ");scanf("%d",&n);
    }while(n<0 || n>10);
    for(i=1;i<=n;i++)t += gt(i);
    printf("\nTong la: %d",t);
    getch();
    return 0;
}
```



```
#include<stdio.h>
#include<conio.h>
int ptb2(float, float, float, float *, float *);
int main(int argc, char *argv[])
{
    float a,b,c,x1,x2;
    int kq;
    printf("Nhap vao a, b, c: ");scanf("%f%f%f",&a,&b,&c);
    kq = ptb2(a,b,c,&x1,&x2);
    if(kq==0)printf("\nPhuong trinh vo nghiem");
    else if(kq==1)printf("\nPhuong trinh co mot nghiem: %6.2f",x1);
    else if(kq==2)printf("\nPhuong trinh co hai nghiem: %6.2f  %6.2f", x1,
x2);
    else printf("\nPhuong trinh co vo so nghiem");
    getch();
    return 0;
}
```



```
int ptb2(float a, float b, float c, float *x1, float *x2){
    float delta;
    if(a==0){
        if((b==0)&&(c!=0)) return 0;
        if((b==0)&&(c==0)) return 3;
        if(b!=0){*x1 = -c/b; return 1;}
    }
    else{ delta = b*b-4*a*c;
        if(delta<0) return 0;
        else if(delta==0){*x1=-b/(2*a);return 1;}
        else{*x1=(-b-sqrt(delta))/(2*a);
              *x2=(-b+sqrt(delta))/(2*a);return 2;}
    }
}
```