5. Hàm – function

Nội dung

- Khái niệm về hàm
- Tham số và biến cục bộ
- Giá trị trả về
- Hàm gọi hàm
- Hàm với tham số là mảng, xâu ký tự
- Biến, hằng toàn cục
- Biến tĩnh – static
- Hàm đệ quy

5.1. Khái niệm về hàm

5.1. Khái niệm về hàm

```
printf ("Programming is fun.\n");
scanf("%i",&n);
int main(void)
```

- Mọi chương trình C đều ẩn chứa một khái niệm cơ bản là hàm
- **Hàm (function)** cung cấp cơ chế cho phép các chương trình dễ dàng ***viết***, ***đọc***, ***hiểu***, ***debug*** (bắt lỗi), ***sửa đổi*** và ***bảo trì***.

5.1. Khái niệm về hàm

- **Định nghĩa hàm**

```
#include <stdio.h>
void printMessage (void)
{
    printf ("Programming is fun.\n");
}

int main (void)
{
    printMessage ();
    return 0;
}
```

5.1. Khái niệm về hàm

- Dòng đầu tiên trong định nghĩa hàm cung cấp cho chương trình dịch 4 thông tin về hàm (theo thứ tự từ trái sang)
  - Người có thể gọi hàm này
  - Kiểu giá trị nó trả về
  - Tên hàm
  - Các tham số mà nó cần

Ví dụ

```
void printMessage (void)
int intTong(int x, int y)
```

- Dòng đầu tiên trong định nghĩa hàm được gọi là ***mẫu hàm***

5.1. Khái niệm về hàm

- Kết quả chạy chương trình ?

```
#include <stdio.h>
void printMessage (void)
{
    printf ("Programming is fun.\n");
}
int main (void)
{
    printMessage ();
    printMessage ();
    return 0;
}
```

5.1. Khái niệm về hàm

- Kết quả chạy chương trình ?

```
#include <stdio.h>
void printMessage (void)
{
    printf ("Programming is fun.\n");
}
int main (void)
{
   int i;
   for ( i = 1; i <= 5; ++i )
     printMessage ();
   return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h> //cho ham system()
#include <math.h>
void CircleArea(float r)
{
   float S;
   S= M_PI*r*r;
   printf("%.2f co dien tich la: %.2f\n",r,S);
}
int main()
{
    CircleArea(5);
    CircleArea(45);
    return 0;
}
```

5.2. Tham số và biến cục bộ

5.2. Tham số và biến cục bộ

- **Tham số của hàm**:
  - giống tham số trong một hàm toán học,
  - Tăng hiệu quả và tính mềm dẻo của hàm

```
void giaiThua(int n)
{
   int i;
   long gt;
   for(i=1;i<=n;i++) gt=gt*i;

   printf("%d!= %ld\n",n,gt);
}
```

5.2. Tham số và biến cục bộ

- Với hàm `giaiThua(int)` đã được định nghĩa

```
int main (void)
{
  giaiThua(5);
  giaiThua(7);
  giaiThua(10);

  return 0;
}
```

5.2. Tham số và biến cục bộ

- **Hàm không có tham số**: phần khai báo danh sách tham số của hàm để trống hoặc dùng từ khóa **void** (nên dùng cách này)
- Dùng hàm không có tham số: **tên_hàm();**

```
void trinhBay(void)
{
  printf("CHUONG TRINH GIAI PHUONG TINH BAC HAI\n");
  printf("====================================\n");
}

int main (void)
{
    trinhBay();
    return 0;
}
```

5.2. Tham số và biến cục bộ

- **Biến cục bộ**: các biến được định nghĩa trong hàm là biến cục bộ.
  - Chúng được tự động tạo ra mỗi khi hàm được gọi
  - Giá trị của chúng là cục bộ (chỉ được truy cập trong phạm vi của hàm đó)
  - Nếu biến có giá trị khởi tạo thì giá trị đó được gán mỗi khi gọi hàm

5.2. Tham số và biến cục bộ

```
void USCLN(int u, int v)
{
    int tmp;
    printf ("USCLN cua %i va %i la ", u, v);
    while ( v != 0 ) {
        tmp = u % v;
        u = v;
        v = tmp;
    }
    printf ("%i\n", u);
}
int main (void)
{
    USCLN (150, 35);
    USCLN (1026, 405);
    return 0;
}
```

5.3. Giá trị trả về

5.3. Giá trị trả về

- Hàm có thể thực hiện và in luôn kết quả nên không cần giá trị trả về (khai báo hàm với từ khóa **void**)

```
void USCLN(int u, int v)
void giaiThua(int n)
```

- Khi không muốn hiển thị ngay kết quả, hoặc kết quả thực hiện của hàm chưa phải kết quả cuối cùng. Ta cần trả về giá trị của hàm cho nơi gọi hàm.
- Trả về giá trị trong hàm bằng từ khóa **return**

```
return biểu_thức;
```

5.3. Giá trị trả về

- Trong khai báo hàm kiểu giá trị trong khai báo phải ***trùng*** với kiểu trả về trong lệnh **return**.

```
int gcd (int u, int v)
float goc_do (float goc_rad)
```

"I'd like to return this shredder."

- Ví dụ

```
int USCLN (int u, int v)
{
    int temp;
    while ( v != 0 ) {
        temp = u % v;  u = v;  v = temp;
    }
    return u;
}
```

5.3. Giá trị trả về

- Với hàm USCLN đã khai báo ở trên

```
int main (void)
{
    int result;
    result = USCLN(150, 35);
    printf ("USCLN cua 150 va 35 la %i\n", result);

    result = USCLN(1026, 405);
    printf ("USCLN cua 1026 va 405 la %i\n", result);

    printf ("USCLN cua 83 va 240 la %i\n", USCLN(83,240));
    return 0;
}
```

5.3. Giá trị trả về

- Hàm có giá trị trả về được dùng giống như các hàm trong thư viện **`math.c`** như **`abs(), floor(), sqrt(), pow(),…`**
- Kiểu trả về phải trùng với kiểu của giá trị trả về của biểu thức trong lệnh **`return`**, nếu không sẽ lỗi
- Hàm khai báo với từ khóa **`void`** thì không có giá trị trả về. Mọi cách thử dùng hàm này như một biểu thức đều gây ra lỗi biên dịch.

  Ví dụ. Với hàm **`void giaiThua(int n)`** ta không thể dùng

```
int n=giaiThua(10);
```

5.4. Hàm gọi hàm

5.4. Hàm gọi hàm

- Chương trình đọc số nguyên có nhiều nhất 3 chữ số, ví dụ đầu vào là số -125 thì chương trình sẽ in ra màn hình là âm một trăm hai mươi năm.
- **Hướng giải quyết**: chia nhỏ vấn đề thành các bài toán con
  - Xây dựng hàm đọc chữ số
  - Xây dựng hàm kiểm tra số đầu vào có hợp lệ hay không (có tối đa ba chữ số)
  - Xây dựng hàm đọc số từ hàm trên

5.4. Hàm gọi hàm

```
//hàm đọc chữ số
void docChuSo(int a)
{
     if(a==0) printf("khong");
     else if(a==1)printf("mot");
     else if(a==2)printf("hai");
     else if(a==3)printf("ba");
     else if(a==4)printf("bon");
     else if(a==5)printf("nam");
     else if(a==6)printf("sau");
     else if(a==7)printf("bay");
     else if(a==8)printf("tam");
     else if(a==9)printf("chin");
     else printf("ERROR!!!");
}
```

5.4. Hàm gọi hàm

- Hàm kiểm tra số đầu vào số hợp lệ hay không (có tối đa ba chữ số). Hàm này trả về giá trị -1 nếu số đầu vào không hợp lệ, trả về 0 nếu ngược lại

```
int kiemTra(int n)
{
     if(n>999 || n<-999) return -1;
     else return 0;
}
```

5.4. Hàm gọi hàm

- Hàm đọc số

```
void docSo(int n)
{
     int a,b,c;
     if(kiemTra(n)==-1)
        printf("Loi! So khong hop le.\n");
     else
     {
         if(n<0){
            printf("am ");
            n=-n;
         }
         c=n%10; n=n/10;
         b=n%10; n=n/10;
         a=n%10;
         if(a>0){ docChuSo(a); printf(" tram ");}
         if(a>0 && b==0)printf("le ");
         if(b==1){ printf("muoi ");}
         if(b>1){ docChuSo(b);printf(" muoi ");}
         docChuSo(c);  printf("\n");     }
}

int main(void)
{
    docSo(-115);
    docSo(-125);
    docSo(-105);
    docSo(-5);
    return 0;
}
```

5.4. Hàm gọi hàm

Hàm và thiết kế top-down

- **Thiết kế top-down**: chia bài toán ban đầu thành các bài toán con cho đến khi các bài toán con có thể xử lý trực tiếp được.
- Viết các hàm để xử lý các bài toán con
- Chương trình chính sẽ chỉ cần gọi các hàm để thực hiện các công việc
- **Ưu điểm**: chương trình gọn, dễ bắt lỗi

5.5. Hàm với tham số là mảng và xâu ký tự

5.5 Tham số là mảng, xâu ký tự

- Hàm in giá trị của các phần tử trong một mảng số thực gồm 10 phần tử

```
void display(float A[10])
{
     int i;
     for(i=0;i<10;i++)
        printf("%.2f ",A[i]);
     printf("\n");
}
```

5.5 Tham số là mảng, xâu ký tự

- Hàm tìm giá trị lớn nhất trong một mảng số thực.

```
float maximum(float A[], int size)
{
     float maxVal=A[0];
     int i;
     for(i=0;i<size;i++)
       if(A[i]>max) maxVal=A[i];

     return maxVal;
}
```

- Số lượng phần tử không cố định, biểu diễn bởi tham số size → hàm mềm dẻo hơn.

5.5 Tham số là mảng, xâu ký tự

```
int main(void)
{
    float array1[10] = { 15.7, -2.8, -3.7, 26, 1.08, 64,
                                        5.6, 12, 34, 9.01};
    float array2[7] = { 1.2, 4.5, 1, -10, 6.5, 3, 2.2 };

    printf("Noi dung mang array1\n");
    display(array1);

    printf("Gia tri lon nhat trong array1: %.2f\n",
                                    maximum(array1,10));

    printf("Gia tri lon nhat trong array2: %.2f\n",
                                     maximum(array2,7));

    return 0;
}
```

5.5 Tham số là mảng, xâu ký tự

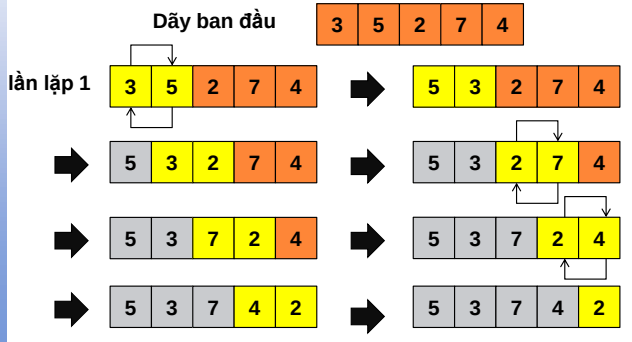
Thay đổi giá trị của các phần tử trong mảng

- Hàm tăng giá trị các phần tử trong mảng lên 2 đơn vị

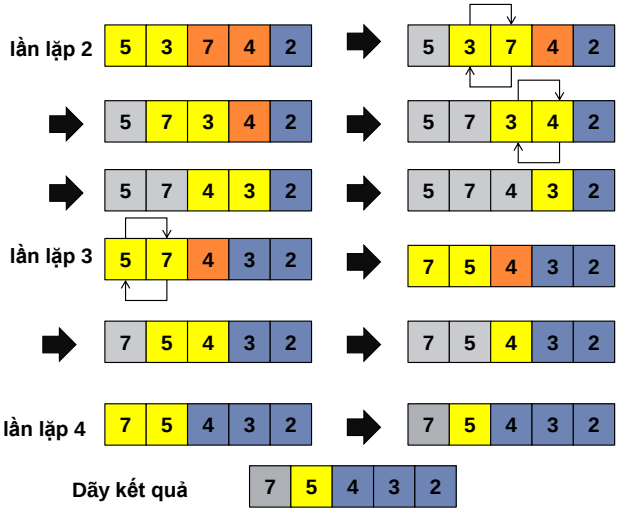
```
void incrementBy2(float A[], int size)
{
    int i;
    for(i=0;i<size;i++) A[i]+=2;
}
```

5.5 Tham số là mảng, xâu ký tự

- Hàm sắp xếp giá trị các phần tử trong mảng theo thứ tự giảm dần
- Thuật toán sắp xếp mảng đơn giản:



5.5 Tham số là mảng, xâu ký tự



5.5 Tham số là mảng, xâu ký tự

- Thuật toán nổi bọt (đổi chỗ):
  - Gán
  - Bước lặp: Lặp từ phần tử đầu tiên đến phần tử thứ (với biến lặp là)
    - So sánh giá trị hai phần tử kề nhau thứ và
    - Nếu giá trị phần tử lớn hơn thì thực hiện đổi chỗ hai phần tử này
  - Gán
  - Thực hiện lại bước lặp cho tới khi

```
void sort (int a[], int n)
{
    int i, j, temp;
    for ( i = 0; i < n - 1; ++i )
        for ( j = i + 1; j < n; ++j )
            if ( a[i] < a[j] ) {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
}
void display(int A[], int size)
{
     int i;
     for(i=0;i<size;i++)
        printf("%d ",A[i]);
     printf("\n");
}
```

5.5 Tham số là mảng, xâu ký tự

```
int main(void)
{
    int i;
    int array[16] = { 34, -5, 6, 0, 12, 100, 56, 22,
                      44, -3, -9, 12, 17, 22, 6, 11 };

    printf ("Mang ban dau:\n");
    display(array,16);

    sort (array, 16);

    printf("Mang sau khi sap xep:\n");
    display(array,16);

    system("pause");
    return 0;
}
```

5.5 Tham số là mảng, xâu ký tự

- **Tham số là mảng nhiều chiều**. Với mảng 2 chiều ***tham số được khuyết chỉ có thể là số dòng***.
- Hàm in nội dung mảng 2 chiều dưới dạng bảng với số lượng dòng không xác định

```
void display(int A[][5], int size)
{
     int i,j;
     for(i=0;i<size;i++)
     {
        for(j=0;j<5;j++)
           printf("%d\t",A[i][j]);
        printf("\n");
     }
}
```

5.5 Tham số là mảng, xâu ký tự

- Hàm nhân đôi giá trị các phần tử trong mảng

```
void nhanDoi(int A[][5],int nRow)
{
     int i,j;
     for(i=0;i<nRow;i++)
        for(j=0;j<5;j++)
            A[i][j]*=2;
}
```

```
int main(void)
{
    int i;
    int Matrix[3][5] = { 34, -5, 6, 0, 12,
                         100, 56, 22, 44, -3,
                         -9, 12, 17, 22, 6};

    printf("Mang ban dau\n");
    display(Matrix,3);

    nhanDoi(Matrix,3);

    printf("Mang sau nhan doi\n");
    display(Matrix,3);

    system("pause");
    return 0;
}
```

5.5 Tham số là mảng, xâu ký tự

- Hàm tìm chiều dài của một xâu ký tự

```
int length(char str[])
{
    int i=0;
    while(str[i]!='\0') i++;
    return i;
}
```

- Sử dụng

```
printf("Chieu dai : %d\n",length("HelloWorld"));
```

Tham số là mảng, xâu ký tự

- **Tham số là mảng có kích thước biến đổi – variable length** (**chỉ có trong C99**): phải đặt tham số về số hàng và số cột trước trước tham số tên mảng

```
void displayMatrix (int nRows, int nCols,
                            int matrix[nRows][nCols])
{
    int row, column;
    for ( row = 0; row < nRows; ++row) {
      for ( column = 0; column < nCols; ++column )
      printf ("%5i", matrix[row][column]);
      printf ("\n");
    }
}
```

5.6. Biến, hằng toàn cục

5.6. Biến toàn cục


```
int globalVar;

void functionA(void)
{
     globalVar=5;
}
void functionB(void)
{
     globalVar=7;
}
int main(void)
{
    printf("Gia tri cua GlobalVar: %d\n",globalVar);
    functionA();
    printf("Gia tri cua GlobalVar: %d\n",globalVar);
    functionB();
    printf("Gia tri cua GlobalVar: %d\n",globalVar);
    return 0;
}
```

5.6. Biến toàn cục

- **Biến toàn cục**:
  - Được khai báo ngoài phạm vi các hàm (kể cả hàm main) và có thể được truy cập bởi bất cứ hàm nào trong chương trình.
  - Thường được dùng để trao đổi thông tin giữa các hàm, giảm bớt số lượng tham số của mỗi hàm.
  - Khó kiểm soát được giá trị do có thể thay đổi tại mọi vị trí, vì thế tránh dùng biến toàn cục đến mức thấp nhất.
- **Hằng toàn cục** (khai báo giống biến toàn cục) hay được dùng.

5.6. Biến tĩnh – static

5.6. Biến tĩnh – static

- **Biến cục bộ và biến tĩnh**

```
float squareRoot (float x)
{
    const float epsilon = .00001;
    float guess = 1.0;
    . . .
}
```

- Biến cục bộ được tự động tạo ra mỗi khi hàm được thực hiện, và được giải phóng (biến mất) khi kết thúc thực hiện hàm
- Nếu hàm được thực hiện lại, giá trị của biến cục bộ trong lần thực hiện trước không được giữ lại.

5.6. Biến tĩnh – static

- Sử dụng khai báo biến tĩnh – static nếu ta muốn lưu trữ giá trị của biến trong hàm trong lần thực hiện ngay trước đó.

```
void staticVariable(void)
{
    int j=0; //bien cuc bo tu dong
    static int i=0; //bien tinh
    i++; j++;
    printf("i = %d, j = %d\n",i,j);
}
```

5.6. Biến tĩnh – static

```
int main(void)
{
   staticVariable();
   staticVariable();
   staticVariable();
   staticVariable();

   return 0;
}
```

- Biến tĩnh có thể dùng để đếm số lần hàm được gọi

5.7. Hàm đệ quy

5.7. Hàm đệ quy

- Hàm đệ quy trong toán học

$$\begin{cases} f(0) = 0 \\ f(x) = 2f(x-1) + x^2 \quad neu\ x > 0 \end{cases}$$

- Từ định nghĩa ta tính được

```
f(3) =  2*f(2)+3*3
     =  2*(2*f(1) + 2*2)+3*3
     =  2*(2*(2*f(0) +1*1) + 2*2) + 3*3
     =  2*(2*(2*0+1*1) + 2*2) + 3*3
     =  2*(2*1+ 2*2) + 3*3
     =  2*6+ 3*3
     =  21
```

5.7. Hàm đệ quy

- Hàm đệ quy là hàm mà trong lời định nghĩa hàm nhắc lại chính nó.

  Ví dụ. Hàm tính giai thừa **n!=n*(n-1)!**

- Cài đặt hàm đệ quy tính f(x)

```
int f( int x )
{
   if( x == 0 )
     return 0;
   else
     return 2 * f( x - 1) + x*x;
}
```

5.7. Hàm đệ quy

- VD. Viết hàm đệ quy tính số **Fibonacci** thứ n