CHƯƠNG 5

HÀM

Nội Dung

- Chương trình con
- Hàm
- Hàm – Ví dụ mở đầu
- Khai báo và định nghĩa hàm.
- Truyền tham số.
- Phạm vi.
- Đệ qui

Chương trình con

- Lập trình cấu trúc là nghệ thuật *che dấu thông tin*.
- Ta lập trình cấu trúc bằng cách dấu các thông tin không cần quan tâm vào trong các *chương trình con.*
- Một chương trình con là một đoạn chương trình được *tách riêng* và *hoạt động độc lập*, ta có thể thực thi chương trình con bằng cách gọi nó. Một chương trình con có được gọi nhiều lần.

Chương trình con

- Chương trình con giúp ta tạo ra các chương trình trong sáng, dễ đọc và dễ sửa chữa và nâng cấp.
- *Chương trình con* phân rã các công việc lớn thành những việc nhỏ hơn, do đó dễ thao tác. Nói cách khác, chương trình con hỗ trợ cách tiếp cận *chia để trị.*
- Chương trình con cho phép xây dựng một hệ thống mới dựa trên những *kết quả đã có* thay vì khởi hành từ con số không.

Chương trình con

- Chương trình con giúp ta trách được những đoạn chương trình tương tự nhau lặp đi lặp lại (nhờ đó tránh được các chương trình *xấu xí*).
- *Hàm* là chương trình con được cài đặt cụ thể trong C.
- C được thiết kế để hàm có thể hoạt động hiệu quả và dễ sử dụng. Một chương trình thường bao gồm *nhiều hàm nhỏ* thay vì một vài hàm lớn.

Hàm

- *Hàm* là chương trình con được cài đặt cụ thể trong C.
- C được thiết kế để hàm có thể hoạt động hiệu quả và dễ sử dụng. Một chương trình thường bao gồm *nhiều hàm nhỏ* thay vì một vài hàm lớn.
- Một chương trình C có thể được viết trên các chương trình nguồn khác nhau, mỗi chương trình nguồn bao gồm một số hàm.
- Các chương trình nguồn được biên dịch riêng rẽ, được liên kết lại với nhau và liên kết với các hàm trong thư viện để tạo ra chương trình.

Hàm – Ví dụ mở đầu

- Bài toán 1: Viết chương trình cho phép nhập vào một số nguyên, kiểm tra số đó có nguyên tố không?
- Bài toán 2: Viết chương trình in ra các số nguyên tố nhỏ hơn 100.
- Bài toán 3: Viết chương trình in ra 100 số nguyên tố đầu tiên.

Bài toán 1: Giải pháp không dùng hàm

```
void main()
{
  int n, a;
  bool b;
  printf("Chuong trinh kiem tra mot so co nguyen to khong.\n");
  printf("Nhap n: ");
  scanf("%d", &n);
  if (n <= 1) b = false;
  else if (n == 2 || n == 3) b = true;
  else if (n % 2 == 0) b = false;
  else {
    a = 3;
    while (a <= n / a && n%a)
        a += 2;
    b = a > n / a;
    }
  if (b)
    printf("%d la so nguyen to.\n", n);
  else
    printf("%d khong la so nguyen to.\n", n);
}
```

Hàm – Ví dụ mở đầu

- Giải quyết bài toán 1: Dùng hàm.
- Ta có thể chia bài toán thành ba phần:
  1. *Nhập* một số.
  2. *Kiểm tra có nguyên tố không*.
  3. Nếu nguyên tố, *in* ra.
- Mỗi việc, ta giao cho một hàm làm, chi tiết cài đặt sẽ được dấu vào trong các hàm.
- *Nhập* và *in* là hàm đã có trong thư viện

Bài toán 1: Giải pháp dùng hàm

```
bool LaSoNguyenTo(int n)
{
   int a;
   if (n == 2 || n == 3)
      return true;
   if (n <= 1)
      return false;
   if (n %2 == 0)
      return false;
   a = 3;
   while (a <= n / a && n%a)
      a += 2;
   return a > n / a;
}
```

Giải pháp dùng hàm

```
void main()
{
   int n;
   printf("Chuong trinh kiem tra mot so co
   nguyen to khong.\n");
   printf("Nhap n: ");
   scanf("%d", &n);
   if (LaSoNguyenTo(n))
      printf("%d la so nguyen to.\n", n);
   else
      printf("%d khong la so nguyen to.\n",
   n);
}
```

Hàm – Ví dụ mở đầu

- Bài toán 2: Viết chương trình in ra các số nguyên tố không quá 100. Tổng quát in ra các số nguyên tố không vượt quá N.
- Xác định các công việc trong bài toán 2.
  - Duyệt từ 2 đến 100
  - Kiểm tra số đang xét có nguyên tố hay không
  - Nếu có, in ra

Bài toán 2: Giải pháp không dùng hàm

```
void main()
{
  int n, N, a;
  bool b;
  printf("Chuong trinh in so nguyen to <= N\n");
  printf("Nhap N: "); scanf("%d", &N);
  for (n = 2; n <= N; n++) {
    if (n <= 1) b = false;
    else if (n == 2 || n == 3) b = true;
    else if (n % 2 == 0) b = false;
    else {
      a = 3;
      while (a <= n / a && n%a)
      a += 2;
      b = a > n / a;
      }
    if (b) printf("%8d", n);
  }
}
```

Bài toán 2: Giải pháp dùng hàm

```
bool LaSoNguyenTo(int n)
{
    if (n == 2 || n == 3) return true;
    if (n <= 1) return false;
    if (n % 2 == 0) return false;
    int a = 3;
    while (a <= n / a && n%a)
        a += 2;
    return a > n / a;
}
void main()
{
    int n, N;
    printf("Chuong trinh in so nguyen to <= N\n");
    printf("Nhap N: "); scanf("%d", &N);
    for (int n = 2; n <= N; n++)
        if (LaSoNguyenTo(n))
            printf("%4d", n);
    printf("\n");
}
```

Hàm – Ví dụ mở đầu

- Bài toán 3: Viết chương trình in ra 100 số nguyên tố đầu tiên. Tổng quát, in ra N số nguyên tố đầu tiên.
- Xác định các công việc trong bài toán 2.
  - Khởi động số nguyên tố đầu tiên là n = 2.
  - Lặp N lần các việc sau:
    - *In* ra số nguyên tố n.
    - *Tính số nguyên tố kế tiếp của n*, gán n là số kế tiếp đó.

Bài toán 3: Giải pháp dùng hàm

```
bool LaSoNguyenTo(int n)
{
    int a;
    if (n == 2 || n == 3) return true;
    if (n <= 1) return false;
    if (n % 2 == 0) return false;
    a = 3;
    while (a <= n / a && n%a)
        a += 2;
    return a > n / a;
}
// Tim so nguyen nho nhat dung sau a
int SoNgToKeTiep(int a)
{
    do
        a++;
    while (!LaSoNguyenTo(a));
    return a;
}
```

Bài toán 3: Giải pháp dùng hàm

```
void main()
{
   int n, N;
   printf("Chuong trinh in N so nguyen to
        dau tien\n");
   printf("Nhap N: ");
   scanf("%d", &N);
   n = 2;
   for (int i = 0; i < N; i++)
   {
      printf("%4d", n);
      n = SoNgToKeTiep(n);
   }
   printf("\n");
}
```

Bài toán 3:
English version

```
bool IsPrime(int n)
{
    int a;
    if (n == 2 || n == 3) return true;
    if (n <= 1) return false;
    if (n % 2 == 0) return false;
    a = 3;
    while (a <= n / a && n%a)
        a += 2;
    return a > n / a;
}
// Find next prime number
int NextPrime(int a)
{
    do
        a++;
    while (!IsPrime(a));
    return a;
}
```

Bài toán 3:
English version

```
void main()
{
   int n, N;
   printf("Program to print the first N
   prime numbers\n");
   printf("Enter N: "); scanf("%d", &N);
   n = 2;
   for (int i = 0; i < N; i++)
   {
      printf("%4d", n);
      n = NextPrime(n);
   }
   printf("\n");
}
```

Khai báo và định nghĩa hàm

- Mọi danh hiệu trong C đều phải được khai báo trước khi dùng, hàm cũng vậy, phải được khai báo trước khi ta gọi (lần đầu tiên).
- *Khai báo hàm* giới thiệu ra tên hàm, kiểu trả về và số lượng, kiểu của các tham số. Ta gọi đó là *mẫu hàm* (prototypes).
- Mẫu hàm cho ta biết các thông tin về hàm.
- Tên của tham số có thể bỏ qua.

Khai báo và định nghĩa hàm

- Khai báo hàm thường được đặt trong tập tin header (.h)
- Cú pháp khai báo hàm:

```
return_type function_name( parameter list );
```

Khai báo và định nghĩa hàm

- Khai báo hàm cho biết tên, danh sách tham số và kiểu trả về, *định nghĩa hàm* là tạo mã nguồn cho phần thân của hàm.
- Cú pháp định nghĩa hàm:

```
return_type function_name( parameter list ) {
   body of the function
}
```

```
bool LaSoNguyenTo(int n);
```

Khai báo hàm

```
void main()
{
   int n;
   printf("Chuong trinh kiem tra mot so co
   nguyen to khong.\n");
   printf("Nhap n: ");
   scanf("%d", &n);
   if (LaSoNguyenTo(n))
      printf("%d la so nguyen to.\n", n);
   else
      printf("%d khong la so nguyen to.\n",
   n);
}
```

Gọi hàm

Định nghĩa hàm

```
bool LaSoNguyenTo(int n)
{
   int a;
   if (n == 2 || n == 3)
      return true;
   if (n <= 1)
      return false;
   if (n %2 == 0)
      return false;
   a = 3;
   while (a <= n / a && n%a)
      a += 2;
   return a > n / a;
}
```

Ví dụ thêm về hàm

```c
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main()
{
/* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf("Max value is : %d\n", ret);

   return 0;
}
```

```
/* function returning the max between two
numbers */
int max(int num1, int num2)
{
   /* local variable declaration */
   int result;

   if (num1 > num2)
   result = num1;
   else
   result = num2;

   return result;
}
```

Ví dụ thêm về hàm

Truyền tham số

- Nếu một hàm có dùng tham số, tham số đó phải được khai báo. Tham số như vậy được gọi là *tham số hình thức.*
- Tham số được truyền khi gọi hàm được gọi là *tham số thực sự.*
- Tham số thực sự phải có kiểu tương thích với tham số hình thức tương ứng.
- Có hai cách truyền tham số là *truyền bằng giá trị* và *truyền bằng tham chiếu.*

Truyền tham số

- *Truyền bằng giá trị:* Một bản sao của tham số thực sự được tạo ra và bản sao này được truyền đến hàm. Sự thay đổi giá trị của tham số bên trong hàm không ảnh hưởng đến tham số thực sự.
- *Truyền bằng tham chiếu:* Một bản sao địa chỉ của tham số thực sự được tạo ra và địa chỉ này được truyền đến hàm. Mọi sự thay đổi giá trị của tham số hình thức cũng làm thay đổi tham số thực sự.
- *C chuẩn* chỉ có truyền tham số bằng *giá trị.*

Ví dụ minh hoạ truyền bằng giá trị

```
void swap(int p1, int p2);

int main()
{
    int a = 10, b = 20;

    printf("Before: Value of a = %d and value of b = %d\n", a,
    b);
    swap(a, b);
    printf("After: Value of a = %d and value of b = %d\n", a, b);
}

void swap(int p1, int p2)
{
    int t;

    t = p2;
    p2 = p1;
    p1 = t;
    printf("Value of a (p1) = %d and value of b(p2) = %d\n", p1,
    p2);
}
```

Output:
Before: Value of a = 10 and value of b = 20
Value of a (p1) = 20 and value of b(p2) = 10
After: Value of a = 10 and value of b = 20

Phạm vi của tên

- *Phạm vi của một tên* là *phần chương trình* "thấy được", có thể truy xuất được tên đó. Ra khỏi phạm vi thì nó trở thành vô hình.
- Có 3 vị trí biến trong C có thể được khai báo:
  1. Bên trong một hàm hay một biến: *Biến địa phương.*
  2. Bến ngoài mọi hàm: *Biến toàn cục.*
  3. Trong phần định nghĩa tham số của hàm: *Tham số hình thức.*

Phạm vi – Biến địa phương

- *Biến địa phương:* Là các biến được khai báo và định nghĩa bên trong một hàm hay một khối, nó có phạm vi truy xuất ở bên trong hàm hay khối đó. Nghĩa là chỉ có thể được dùng bởi các phát biểu ở bên trong hàm hay khối đó.
- Biến địa phương có thể trùng tên với biến địa phương định nghĩa ở một hàm khác hay một khối khác.
- Trong ví dụ sau, các biến a,b,c là địa phương trong hàm main().

Biến địa phương

```c
#include <stdio.h>

int main() {

  /* local variable declaration */
  int a, b;
  int c;

  /* actual initialization */
  a = 10;
  b = 20;
  c = a + b;

  printf("value of a = %d, b = %d and c = %d\n", a, b, c);

  return 0;
}
```

Phạm vi – Biến toàn cục

- *Biến toàn cục:* Là các biến được khai báo và định nghĩa bên ngoài mọi hàm, thường được khai báo ở đầu tập tin nguồn. Biến toàn cục giữ giá trị của nó suốt thời gian sống của chương trình.
- Biến toàn cục có phạm vi truy xuất toàn cục. Nó có thể được dùng bởi các phát biểu ở trong bất kỳ hàm nào trong chương trình, có thể hàm được định nghĩa ở tập tin chương trình nguồn khác.
- Biến địa phương có thể trùng tên với biến toàn cục.

Biến toàn cục

```c
/* global variable declaration */
int g;

int main()
{

  /* local variable declaration */
  int a, b;

  /* actual initialization */
  a = 10;
  b = 20;
  g = a + b;

  printf("value of a = %d, b = %d and g = %d\n", a, b, g);

  return 0;
}
```

Biến địa phương và biến toàn cục trùng tên.

```c
#include <stdio.h>

/* global variable declaration */
int g = 20;

int main() {

   /* local variable declaration */
   int g = 10;

   printf("value of g = %d\n", g);

   return 0;
}
```

Rule: Phép vua thua lệ làng.

Phạm vi – Tham số hình thức

- *Tham số hình thức* hoạt động tương tự như biến địa phương, nó có phạm vi truy xuất là *bên trong hàm* và có che biến toàn cục có cùng tên.

Tham số hình thức che biến toàn cục trùng tên.

```c
/* global variable declaration */
int a = 20;

int main() {

     /* local variable declaration in main function */
     int a = 10;
     int b = 20;
     int c = 0;

     printf("value of a in main() = %d\n", a);
     c = sum(a, b);
     printf("value of c in main() = %d\n", c);

     return 0;
}

/* function to add two integers */
int sum(int a, int b) {

     printf("value of a in sum() = %d\n", a);
     printf("value of b in sum() = %d\n", b);

     return a + b;
}
```

Rule: Phép vua thua lệ làng.

Biến và hàm tĩnh

- Biến hoặc hàm tĩnh toàn cục có phạm vi truy xuất nội bộ bên trong tập tin định nghĩa.
- Biến tĩnh cũng có thể là biến địa phương. Nó tồn tại suốt thời gian sống của chương trình.

Đệ qui

- Một tính chất T được gọi là có tính *đệ qui* nếu trong định nghĩa của T có sử dụng đến chính T. (Hãy cho ví dụ).
- Một *hàm* được gọi là có *tính đệ qui* nếu nó *gọi chính nó*, trực tiếp hoặc gián tiếp.
- Mọi hàm đệ qui đều có thể được viết lại bằng vòng lặp không cần đệ qui. (khử đệ qui).
- Khi một hàm gọi đệ qui, mỗi lần gọi có một bộ các tham số và biến địa phương riêng.

Ví dụ về đệ qui

```c
#include <stdio.h>

/* printd:  print n in decimal */
void printd(int n) {
   if (n < 0) {
      putchar('-');
      n = -n;
   }
   if (n / 10)
      printd(n / 10);
   putchar(n % 10 + '0');
}
```

Đệ qui

- Bài tập:
  1. Tính tổng N số nguyên tự nhiên đầu tiên.
  2. Tính N!
  3. Tính tổng các chữ số của một số nguyên dương.
  4. Tính ước số chung lớn nhất của 2 số nguyên.
  5. Xuất một số dưới dạng hệ 16.
  6. Xuất đảo ngược một số.
  7. Tính số đảo của một số.

Một số nguyên tắc xây dựng hàm

- Dùng tên có tính gợi nhớ để đặt cho hàm, biến. Tên cho thấy mục đích của hàm, biến.
- Tham số chỉ truyền vừa đủ, không dư, không thiếu (Ví dụ?).
- Truyền bằng tham số hình thức thay vì dùng biến toàn cục.
- Hàm không nên quá lớn, không quá 200 dòng lệnh (có thể có ngoại lệ). Nếu hàm lớn, xem xét phân rã thành các hàm bổ trợ.

Một số nguyên tắc xây dựng hàm

- Dùng tên có tính gợi nhớ để đặt cho hàm, biến. Tên cho thấy mục đích của hàm, biến.
- Tham số chỉ truyền vừa đủ, không dư, không thiếu (Ví dụ?).
- Truyền bằng tham số hình thức thay vì dùng biến toàn cục.
- Hàm không nên quá lớn. Nếu hàm lớn, xem xét phân rã thành các hàm bổ trợ.

Một số nguyên tắc xây dựng hàm

- Tránh các đoạn chương trình tương tự nhau lặp đi lặp lại. Nếu có, loại bỏ bằng cách dùng hàm bổ trợ (Thảo luận: Cho ví dụ).

→Dùng hàm bổ trợ để tránh *mã trùng lặp* và tăng mức độ *trừu tượng* của hàm.

- Một hàm *chỉ làm một việc và làm việc đó cho tốt* (Phân tích).
- Mã chương trình cần được thực thi ở cùng một mức (Xem ví dụ).