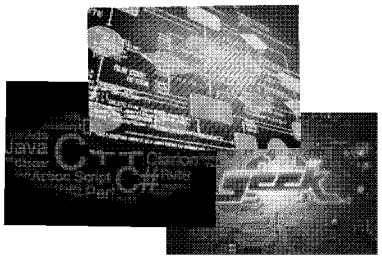
ĐẠI HỌC QUỐC GIA TPHCM
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

NHẬP MÔN LẬP TRÌNH

CHƯƠNG V

HÀM VÀ THAM SỐ



Nguyễn Trọng Chỉnh
chinhnt@uit.edu.vn

HÀM VÀ THAM SỐ

- GIỚI THIỆU
- TRUYỀN THAM SỐ CHO HÀM
- BIẾN TOÀN CỤC VÀ BIẾN CỤC BỘ

GIỚI THIỆU

❖KHÁI NIỆM

Hàm là một đoạn chương trình độc lập, được tổ chức nhằm thực hiện một công việc xác định dựa trên các tham số và có thể được thực hiện nhiều lần. **Ví dụ:**

```
int giaithua(int n) {
  int r = 1;
  while (n > 0) r = r * n--;
  return r;
}
```

Do hàm là một đơn vị độc lập, nên không cho phép xây dựng một hàm bên trong một hàm khác.

GIỚI THIỆU

❖MỤC ĐÍCH XÂY DỰNG HÀM

- Lập trình theo module.
- Tránh lặp lại một thao tác.

Ví dụ: Viết chương trình nhập họ tên, năm sinh và điểm Toán, Lý và Anh văn của 3 học sinh và in ra thông tin của 3 học sinh đó.

```
#include <stdio.h>
void main() {
    char ht1[30], ht2[30], ht3[30];
    float t1,t2,t3,l1,l2,l3,av1,av2,av3;
```

GIỚI THIỆU

```
printf("Ho ten: "); gets(ht1);
printf("Toan: "); scanf("%d%*c", &t1);
printf("Ly: "); scanf("%d%*c", &l1);
printf("Anh Van: "); scanf("%d%*c", &av1);
printf("Ho ten: "); gets(ht2);
printf("Toan: "); scanf("%d%*c", &t2);
printf("Ly: "); scanf("%d%*c", &l2);
printf("Anh Van: "); scanf("%d%*c", &av2);
printf("Ho ten: "); gets(ht3);
printf("Toan: "); scanf("%d%*c", &t3);
printf("Ly: "); scanf("%d%*c", &l3);
printf("Anh Van: "); scanf("%d%*c", &av3);
```

GIỚI THIỆU

```
printf("Ho ten: %s\n",ht1);
 printf("Toan: %f, Ly: %f, Anh van: %f\n",t1,l1,av1);
 printf("Ho ten: %s\n",ht2);
 printf("Toan: %f, Ly: %f, Anh van: %f\n",t2,l2,av2);
 printf("Ho ten: %s\n",ht3);
 printf("Toan: %f, Ly: %f, Anh van: %f\n",t3,l3,av3);
}
```

GIỚI THIỆU

chương trình được viết lại theo module

```
#include <stdio.h>
void nhap(char *hoten, float *toan, float *ly, float *anhvan);
void xuat(char *hoten, float toan, float ly, float anhvan);
   void main() {
      char ht1[30], ht2[30], ht3[30];
      float t1,t2,t3,l1,l2,l3,av1,av2,av3;
      nhap(ht1, &t1, &l1, &av1); nhap(ht2, &t2, &l2, &av2);
      nhap(ht3, &t3, &l3, &av3);
      xuat(ht1, t1, l1, av1); xuat(ht2, t2, l2, av2);
      xuat(ht3, t3, l3, av3);
   }
```

GIỚI THIỆU

```
    void nhap(char *hoten, float *toan, float *ly, float *anhvan) {
        printf("Ho ten: "); gets(hoten);
     printf("Toan: "); scanf("%d%*c", toan);
     printf("Ly: "); scanf("%d%*c", ly);
     printf("Anh Van: "); scanf("%d%*c", anhvan);
   }
void xuat(char *hoten, float toan, float ly, float anhvan) {
     printf("Ho ten: %s\n", hoten);
     printf("Toan: %f, Ly: %f, Anh van: %f\n",toan,ly,anhvan);
}
```

GIỚI THIỆU

❖CÁC THÀNH PHẦN

* **Prototype (Nguyên mẫu):** Là phần khai báo các tên hàm, danh sách các đối số đầu vào và giá trị đối số đầu ra. Các đối số đầu vào cách nhau bằng dấu phẩy ,. Nguyên mẫu thường được khai báo để trình biên dịch biết các hàm có thể được sử dụng. Kết thúc khai báo nguyên mẫu của hàm phải có dấu chấm phẩy ;

Ví dụ:

```
int Giaithua(int n);
float Max(float a, float b);
int nhap(int &a, int &b, int &c);
```

GIỚI THIỆU

❖CÁC THÀNH PHẦN

* **Prototype (Nguyên mẫu):**

C/C++ biên dịch chương trình theo thứ tự từ trên xuống. Vì thế, nếu một hàm chưa được khai báo mà được gọi sẽ bị lỗi.

Ví dụ:

```
int tinh1(int x) {
   if (x > 0) return x + tinh2(x – 1); else return 1;
}
int tinh2(int x) {
    if (x > 0) return x * tinh1(x – 2); else return 2;
}
```

GIỚI THIỆU

❖CÁC THÀNH PHẦN

*** Prototype (Nguyên mẫu):**

Để khắc phục, cần khai báo nguyên mẫu của hàm trước khi sử dụng.

Ví dụ:

```
int tinh1(int x);
int tinh2(int x);
int tinh1(int x) {
   if (x > 0) return x + tinh2(x – 1); else return 1;
}
int tinh2(int x) {
    if (x > 0) return x * tinh1(x – 2); else return 2;
}
```

GIỚI THIỆU

❖CÁC THÀNH PHẦN

*** Thân hàm:**

Là một khối lệnh bắt đầu ngay sau nguyên mẫu của hàm, chứa các cấu trúc và các câu lệnh để thực hiện công việc của hàm. Tham số đầu ra của hàm thuộc kiểu gì thì trong thân hàm phải có câu lệnh với từ khóa **return** để trả về một giá trị hoặc biểu thức thuộc kiểu đó. Giữa thân hàm và nguyên mẫu của hàm không có dấu **;. Ví dụ:**

```
int Max(int x, int y) {
   int r = x; if (r < y) r = y;
   return r;
}
```

GIỚI THIỆU

❖CÁC THÀNH PHẦN

*** Thân hàm:**

Nếu tham số đầu ra của hàm là kiểu **void** thì thân hàm có thể không cần câu lệnh trả về giá trị của hàm, hoặc nếu có thì câu lệnh này chỉ chứa từ khóa **return**.

Ví dụ:

```
void xuat(char *hoten, float toan, float ly, float anhvan) {
  printf("Ho ten: %s\n", hoten);
  printf("Toan: %f, Ly: %f, Anh van: %f\n",toan,ly,anhvan);
}
```

GIỚI THIỆU

❖THỰC THI MỘT HÀM

Để thực thi một hàm, C/C++ đưa ra biểu thức gọi hàm như sau:

Tên_hàm([các tham số])

Vì là một biểu thức, nên kết quả gọi hàm có thể được tính toán trong một biểu thức, hoặc có thể gọi thực hiện như một biểu thức con.

Ví dụ:

```
1. printf("Hello");
2. x = pow(2, (float)3/2) * 4;
```

GIỚI THIỆU

❖THỰC THI MỘT HÀM

Lưu ý: Khi gọi thực hiện một hàm, phải truyền đủ tham số cho hàm, và kiểu của các tham số phải tương ứng với kiểu của đối số trong nguyên mẫu của hàm.

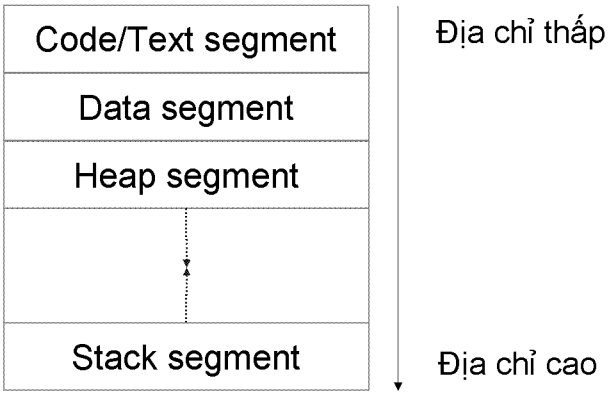
Ví dụ:

```
void nhap(char *hoten, float *toan, float *ly, float *anhvan);
....
char hoten;
float t, l, av;
nhap(hoten, &t, &l, av);
```

GIỚI THIỆU

❖BỘ NHỚ KHI CHƯƠNG TRÌNH CHẠY

Khi chạy, chương trình được nạp vào bộ nhớ RAM theo các phân đoạn (segment) như sau:



GIỚI THIỆU

❖BỘ NHỚ KHI CHƯƠNG TRÌNH CHẠY

- Code (Text) segment: chứa mã lệnh chương trình. Kích thước được xác định khi biên dịch chương trình.
- Data segment: chứa hằng và các biến được lưu trữ trong suốt thời gian thực hiện chương trình. Kích thước được xác định khi biên dịch chương trình.
- Heap segment: là vùng nhớ được sử dụng khi cấp phát động do người lập trình tự cấp phát và hủy.
- Stack segment: là vùng nhớ sử dụng cho các biến có thời gian sử dụng trong một khối lệnh, được quản lý tự động theo cấu trúc ngăn xếp (stack).

GIỚI THIỆU

❖BỘ NHỚ KHI CHƯƠNG TRÌNH CHẠY

Khi một hàm được gọi, máy tính sẽ thực hiện theo trình tự sau:

- Đẩy địa chỉ của lệnh kế tiếp vào Stack.
- Con trỏ lệnh (IP) chỉ đến mã lệnh của hàm được gọi.
- Cấp phát vùng nhớ cho các đối số và biến của hàm trên vùng Stack.
- Gán giá trị của tham số cho các đối số tương ứng

GIỚI THIỆU

❖BỘ NHỚ KHI CHƯƠNG TRÌNH CHẠY

- Thực hiện các câu lệnh trong thân hàm
- Khi gặp từ khóa return hoặc kết thúc khối lệnh của thân hàm thì sẽ lấy tham số đầu ra của hàm để tiếp tục biểu thức, giải phóng vùng nhớ trên Stack của các đối số và các biến.
- Lấy địa chỉ lệnh khỏi Stack để tiếp tục thực hiện chương trình.

TRUYỀN THAM SỐ CHO HÀM

❖TRUYỀN THAM TRỊ (VALUE)

Trong cách truyền tham trị, các đối số của hàm chỉ nhận giá trị của tham số. Vì thế, quá trình thực hiện trên các đối số không ảnh hưởng đến tham số đầu vào.

Để khai báo đối số được truyền theo tham trị, trong danh sách đối số khai báo như sau:

kiểu tên_đối/tên_hàm

TRUYỀN THAM SỐ CHO HÀM

❖TRUYỀN THAM TRỊ (VALUE)

Ví dụ:

```
int luythua(int x, int y) {
      int r = 1;
    while (y-- > 0) r = r * x;
      return r;
}
  void main() {
    int x = 2, y = 4, z;
      z = luythua(x, y);
  } // x = 2, y = 4, z = 16
```

TRUYỀN THAM SỐ CHO HÀM

❖TRUYỀN THAM CHIẾU (REFERENCE)

Trong cách truyền tham chiếu, các đối số của hàm là các biến giả của tham số. Tức là không có các biến mới được tạo ra trên Stack mà sử dụng ngay vùng nhớ của tham số. Vì thế, quá trình thực hiện trên các đối số sẽ hưởng đến tham số đầu vào.

Truyền tham chiếu là cách truyền tham số của C++, để khai báo một đối được truyền theo kiểu tham chiếu, trong danh sách đối số khai báo như sau:

kiểu &tên_đối/tên_hàm

TRUYỀN THAM SỐ CHO HÀM

❖TRUYỀN THAM CHIẾU (VALUE)

Ví dụ:

```
void hoandoi(int &x, int &y) {
      int tmp = x;
      x = y; y = tmp;
}
 void main() {
    int x = 2, y = 3;
    hoandoi(x, y);
 } // x = 3, y = 2
```

TRUYỀN THAM SỐ CHO HÀM

❖TRUYỀN THAM CHIẾU (REFERENCE)

Ưu điểm của truyền tham chiếu là không tạo biến tạm nên tiết kiệm được vùng nhớ và thời gian thực hiện chương trình. Tuy nhiên, do truyền tham chiếu nên tham số có thể bị thay đổi giá trị dẫn đến sai khi thực hiện chương trình.

Để truyền tham số theo tham chiếu mà đảm bảo không thay đổi giá trị của tham số, trong khai báo đối số thêm từ khóa const như sau

const kiểu &tên_đối/tên_hàm

TRUYỀN THAM SỐ CHO HÀM

❖TRUYỀN THAM CHIẾU (REFERENCE)

Khi đó, việc tham số giữ nguyên giá trị được đảm bảo bằng cách trình biên dịch không cho phép bất cứ câu lệnh nào trực tiếp thay đổi giá trị của tham số đó.

Ví dụ:

```
int luythua(const int &x, const int &y) {
    int r = 1;
    while(y-- > 0) // báo lỗi biên dịch
        r *= x;
    return r;
}
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

❖BIẾN TOÀN BỘ (GLOBAL)

Biến toàn bộ là biến mà câu lệnh khai báo của nó không nằm trong bất kỳ hàm hoặc khối lệnh nào. Biến toàn bộ có thể được sử dụng trong tất cả các câu lệnh được viết bên dưới nó.

Khi khai báo biến toàn bộ, nếu không có giá trị khởi tạo thì biến sẽ lấy giá trị là 0.

Ví dụ:

```
#include <stdio.h>
int n;
void main() {
    scanf("%d", &n); printf("%d", n);
}
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

Ví dụ: Tìm chỗ sai trong đoạn chương trình

```
#include <stdio.h>
int giaithua() {
    int r = 1, i;
    for (i = 1; i <=n; i++) r = r * i;
    return r;
}
int n;
void main() {
    scanf("%d", &n); printf("%d! = %d", giaithua());
}
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

```
#include <stdio.h>
int giaithua() {
    int r = 1, i;
    for (i = 1; i <=n; i++) r = r * i; // sai vì n chưa khai báo.
    return r;
}
int n;
void main() {
    scanf("%d", &n); printf("%d! = %d", giaithua());
}
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

Ví dụ: tìm chỗ sai trong đoạn chương trình

```
#include <stdio.h>
int n;
void main() {
    scanf("%d", &n); printf("%d! = %d", giaithua());
}
int giaithua() {
    int r = 1, i;
    for (i = 1; i <=n; i++) r = r * i;
    return r;
}
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

```
#include <stdio.h>
int n;
void main() {
    scanf("%d", &n); printf("%d! = %d", giaithua());
}
int giaithua() {
    int r = 1, i;
    for (i = 1; i <=n; i++) r = r * i;
    return r;
}
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

❖BIẾN CỤC BỘ (LOCAL)

Biến cục bộ là biến mà câu lệnh khai báo của nó được đặt trong thân hàm hoặc trong một khối lệnh. Biến cục bộ chỉ được sử dụng bởi những câu lệnh trong cùng một khối lệnh chứa nó và đặt sau nó.

Khi khai báo biến cục bộ, nếu không có giá trị khởi tạo thì giá trị của nó là không xác định

Ví dụ:

```
#include <stdio.h>
void main() {
    int n;
    scanf("%d", &n); printf("%d", n);
}
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

Ví dụ: Tìm chỗ sai trong đoạn chương trình

```
#include <stdio.h>
void main() {
    int n;
    {
        int m = 0;
        scanf("%d", &n);
        while (n-- > 0) m = m + n * n;
    }
    printf("tong i^2, i tu 1 den %d là %d", n, m);
}
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

```
#include <stdio.h>
void main() {
    int n;
    {
        int m = 0;
        scanf("%d", &n);
        while (n-- > 0) m = m + n * n;
    }
    printf("tong i^2, i tu 1 den %d là %d", n, m);
}
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

Ví dụ: Tìm chỗ sai trong đoạn chương trình

```
#include <stdio.h>
void kiemtra(int x) {
    int ketqua = x % 2 == 0;
}
int n;
void main() {
    scanf("%d", n);
    kiemtra();
    if (ketqua)
        printf("%d la so chan", n);
    else
        printf("%d la so le", n);
}
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

```
#include <stdio.h>
void kiemtra(int x) {
    int ketqua = x % 2 == 0;
}
int n;
void main() {
    scanf("%d", n);
    kiemtra();
    if (ketqua) // Biến này chưa được khai báo.
        printf("%d la so chan", n);
    else
        printf("%d la so le", n);
}
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

❖LƯU Ý

- Tên biến toàn cục và biến cục bộ trong cùng một khối lệnh không được trùng nhau.
- Nếu biến cục bộ trong khối lệnh A cùng tên với biến cục bộ trong khối lệnh B, và khối lệnh A chứa khối lệnh B, thì các lệnh trong khối lệnh B truy xuất đến biến thuộc khối lệnh B. Tương tự nếu A là toàn bộ chương trình và biến của nó là biến toàn cục.

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

Ví dụ

```
#include <stdio.h>
#include <math.h>
void main() {
    int i, n;
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        int n = 0, j;
        for (j = 2; j <= sqrt(i); j++)
            if(i % j == 0) n++;
        if (n == 0) printf("%d ", i);
    }
}
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

Ví dụ

```
#include <stdio.h>
#include <math.h>
int n;
void main() {
    int i;
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        int n = 0, j;
        for (j = 2; j <= sqrt(i); j++)
            if(i % j == 0) n++;
        if (n == 0) printf("%d ", i);
    }
}
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

❖LƯU Ý

- Đối với biến cục bộ, nếu khai báo bắt đầu bằng từ khóa **static**, thì biến cục bộ đó sẽ được khởi tạo 1 lần trên vùng Data segment và không bị mất giá trị khi kết thúc khối lệnh. Giá trị của biến này sẽ tồn tại đến khi chương trình kết thúc. **Ví dụ**

```
void main() {
   int i;
   for (i = 0; i < 20; i++) {
      static int n = 0; n++;
      printf("%d ", n);
   }
}//in: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

BIẾN TOÀN BỘ VÀ BIẾN CỤC BỘ

❖LƯU Ý

- Đối với biến toàn bộ, có thể mở rộng tác dụng của nó sang các file được biên dịch khác sau khi liên kết chương trình nếu các file này khai báo lại biến đó bắt đầu với từ khóa **extern**. **Ví dụ**:

```
extern int n;
```

- Nếu biến toàn bộ được khai báo bắt đầu bằng từ khóa **static**, thì biến toàn bộ đó sẽ được khởi tạo trên vùng Data segment như biến toàn bộ thông thường, nhưng biến này chỉ có tác dụng trong file được biên dịch sau khi liên kết chương trình.

BÀI TẬP

Xây dựng các hàm sau:

1) Tính diện tích tam giác với tọa độ 3 đỉnh.
2) Nhập giá trị cho biến kiểu int, float, và char* có kèm thông báo.
3) Kiểm tra x là số chính phương
4) Kiểm tra x là số nguyên tố
5) Giải phương trình bậc 2
6) In n ký tự tùy ý c lên màn hình.