CHƯƠNG 9
HÀM

9.1 Khái niệm hàm
9.2 Khai báo hàm
9.3 Đối số của hàm - đối số là tham trị
9.4 Kết quả trả về của hàm - lệnh RETURN
9.5 PROTOTYPE của một hàm
9.6 Hàm đệ quy
Bài tập cuối chương

CHƯƠNG 9
HÀM

9.1 KHÁI NIỆM HÀM

Chương trình con là đoạn chương trình đảm nhận thực hiện một thao tác nhất định.

Đối với C, chương trình con chỉ ở một dạng là hàm (function), không có khái niệm thủ tục (procedure).

CHƯƠNG 9
HÀM

9.1 KHÁI NIỆM HÀM

Hàm main () là hàm đặc biệt của C, nó là một hàm mà trong đó các thao tác lệnh (bao gồm các biểu thức tính toán, gọi hàm, ...) được C thực hiện theo một trình tự hợp logic để giải quyết bài toán được đặt ra.

Việc sử dụng hàm trong C sẽ làm cho chương trình trở nên rất dễ quản lý, dễ sửa sai.

CHƯƠNG 9
HÀM

9.1 KHÁI NIỆM HÀM

Tất cả các hàm trong C đều ngang cấp nhau. Các hàm đều có thể gọi lẫn nhau, dĩ nhiên hàm được gọi phải được khai báo trước hàm gọi.

CHƯƠNG 9
HÀM

9.1 KHÁI NIỆM HÀM

Các hàm trong một chương trình có thể nằm trên các tập tin khác nhau và khác với tập tin chính (chứa hàm main ()), mỗi tập tin được gọi là một module chương trình,
Các module chương trình sẽ được dịch riêng rẽ và sau đó được liên kết (link) lại với nhau để tạo ra được một tập tin thực thi duy nhất.
Cách tạo chương trình theo kiểu nhiều module như vậy trong C là project

CHƯƠNG 9
HÀM

9.1 KHÁI NIỆM HÀM

Ví dụ: *Chương trình 1*

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
main ()
{
        double a, b, c, delta, n1, n2;
        clrscr();
        printf ("Nhap 3 he so phuong trinh bac hai; ");
        scanf ("%lf %lf %lf", &a, &b, &c);
```

CHƯƠNG 9
HÀM

9.1 KHÁI NIỆM HÀM

```
if (a ==0)/* phuong trinh suy bien ve bac nhat */
{
        printf ("Phuong trinh suy bien ve bac nhat va
");
        if (b == 0)
                if (c == 0)
                        printf ("vo so nghiem\n");
                else /* c != 0 */
                        printf ("vo nghiem\n");
        else / * b != 0 */
```

CHƯƠNG 9
HÀM

9.1 KHÁI NIỆM HÀM

```
        {
                                n1 = -c/b;
                                printf ("co 1 nghiem: = %5.2f \n",
n1);
                        }

        }
        else /* a != 0 */
        {
                printf ("Phuong trinh bac hai va ");
                delta = b*b - 4*a*c;
```

CHƯƠNG 9
HÀM

9.1 KHÁI NIỆM HÀM

```
if (delta < 0)
                printf ("vo nghiem thuc\n");
        else if (delta == 0)
        {
                n1 = n2 = -b/2/a;
                printf ("co nghiem kep x1 = x2 = %5.2f \n"
,n1);
        }
```

9.1 KHÁI NIỆM HÀM

```
                else /* delta > 0 */
        {
                n1 = (-b + sqrt(delta))/2/a;
                n2 = (-b - sqrt(delta))/2/a;
                printf ("co hai nghiem phan biet; \n");
                printf ("x1 = %5.2f \n", n1);
                printf ( x2 = %5.2f \n", n2);
        }
    }
    getch();
}
```

CHƯƠNG 9
HÀM

9.1 KHÁI NIỆM HÀM

Ví dụ: *Chương trình 2*

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
void gptb1 (double a, double b);
void gptb2 (double a, double b, double c);
```

CHƯƠNG 9
HÀM

9.1 KHÁI NIỆM HÀM

```
void gptb1 (double a, double b)
{
        printf ("Phuong trinh suy bien ve bac nhat va ");
        if (a == 0)
                if (b == 0)
                        printf ("vo so nghiem\n");
                else /* b != 0 */
                        printf ("vo nghiem\n");
        else
                        printf ("co 1 nghiem: x = %5.2f \n",-b/a);
}
```

CHƯƠNG 9
HÀM

9.1 KHÁI NIỆM HÀM

```
void gptb2 (double a,double b,double c)
{       double delta, x1, x2;
        printf ("Phuong trinh bac hai va ");
        delta = b*b - 4*a*c;
        if (delta < 0)
                printf ("vo nghiem thuc\n");
        else if (delta == 0)
                printf ("co nghiem kep x1 = x2 = %5.2f \n", -
b/2/a);
```

CHƯƠNG 9
HÀM

9.1 KHÁI NIỆM HÀM

```
else /* delta > 0 */
    {           x1 = (-b + sqrt(delta))/2/a;
        x2 = (-b - sqrt(delta))/2/a;
        printf ("co hai nghiem phan biet: \n");
        printf ("x1 = %5.2f \n ", x1);
        printf ("x2 = %5.2f \n" , x2);
    }
}
```

CHƯƠNG 9
HÀM

9.1 KHÁI NIỆM HÀM

```
main()
{
        double a, b, c;
        clrscr();
        printf ("Nhap 3 he so phuong trinh bac hai: ");
        scant ("%lf %lf %lf", &a, &b, &c);
        if (a == 0)    /* phuong trinh suy bien ve bac nhat */
                gptb1 (b, c);
        else /* a != 0 */
                gptb2 (a, b, c);
        getch();
}
```

CHƯƠNG 9
HÀM

9.2 KHAI BÁO HÀM

Khai báo một hàm là chỉ ra rõ rằng trả về vị trí kiểu gì, đối số đưa vào cho hàm có bao nhiêu đối số, mỗi đối số có kiểu như thế nào và các lệnh bên trong thân hàm xác định thao tác của hàm.

Có hai loại hàm: hàm trong thư viện của C và hàm do lập trình viên tự định nghĩa.

CHƯƠNG 9
HÀM

9.2 KHAI BÁO HÀM

- Nếu hàm sử dụng là hàm chuẩn trong thư viện thì việc khai báo hàm chỉ đơn giản là khai báo prototype của hàm, các prototype này đã được phân loại và ở trong các file .h, lập trình viên cần ra lệnh **#include** bao hàm các file này vào chương trình hoặc module chương trình sử dụng nó.

CHƯƠNG 9
HÀM

9.2 KHAI BÁO HÀM

- Nếu các hàm sử dụng là do lập trình viên tự định nghĩa thì việc khai báo hàm bao gồm hai việc: khai báo prototype của hàm đầu chương trình và định nghĩa các lệnh bên trong thân hàm (hay thường được gọi tắt là định nghĩa hàm).

CHƯƠNG 9
HÀM

9.2 KHAI BÁO HÀM

Dạng 1:

```
kiểu tên_hàm
(danh_sách_khai_báo_đối_số)
{
        khai_báo_biến_cục_bộ

        lệnh
}
```

Dạng 2: (Lạc hậu)

```
kiểu tên_hàm (danh_sách_đối_số)
khai_báo_đối_số
{
        khai_báo_biến_cục_bộ
        lệnh
}
```

CHƯƠNG 9
HÀM

9.2 KHAI BÁO HÀM

Dạng 1:

```
int so_sanh (int a, int b)
{
        int ket_qua;
        if (a >b)
                ket_qua = 1:
        else if (a == b)

                ket_qua = 0;
        else if (a < b)
                ket_qua = -1;
        return ket_qua;
}
```

Dạng 2:

```
int so_sanh (a, b)
int a, b;
{
        int ket_qua:
        if (a >b)
                ket_qua = 1;
        else if (a == b)
                ket_qua = 0;
        else if (a < b)
                ket_qua = -1;
        return ket_qua;
}
```

9.2 KHAI BÁO HÀM

Ví dụ:

```
#include <stdio.h>
#include <conio.h>
int so_sanh (int a, int b); ← prototype của hàm so_sanh
main()
{
        int a, b, ket_qua;
        clrscr();
        printf ("Moi nhap hai so ");
        scanf ("%d %d" , &a, &b);
        ket_qua = so_sanh (a, b);                    ←gọi hàm
```

9.2 KHAI BÁO HÀM

```
        switch (ket_qua)
                { case -1:
                        printf ("So %d nho hon so %d \n" , a, b);
                        break;
                  case 0:
                        printf ("So %d bang so %d \n", a, b);
                        break;
                  case 1:
                        printf ("So %d lon hon so %d \n" , a, b);
                        break;          }
        getch();
}
```

CHƯƠNG 9
HÀM

9.2 KHAI BÁO HÀM

```
int so_sanh (int a, int b)
{
        int ket_qua:
        if (a >b)
                ket_qua = 1;
        else if (a == b)
                ket_qua = 0;
        else if (a < b)
                ket_qua = -1;
        return ket_qua;
}
```

CHƯƠNG 9
HÀM

9.3 ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

Khi gọi hàm thì đối số thật cần gởi cho hàm chỉ được gởi dưới dạng tham số trị, có nghĩa là các biến, trị hoặc biểu thức được gởi đến cho một hàm, qua đối số của nó, sẽ được lấy trị để tính toán trong thân hàm.

Có thể nói trị của biến thật bên ngoài khi gọi hàm đã được chép sang đối số giả, ta có thể xem như là biến cục bộ của hàm, và mọi việc tính toán chỉ được thực hiện trên biến cục bộ này mà thôi.

9.3 ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

Ví dụ: Viết chương trình tính lũy thừa n của x(x^n), với n nguyên và thực.

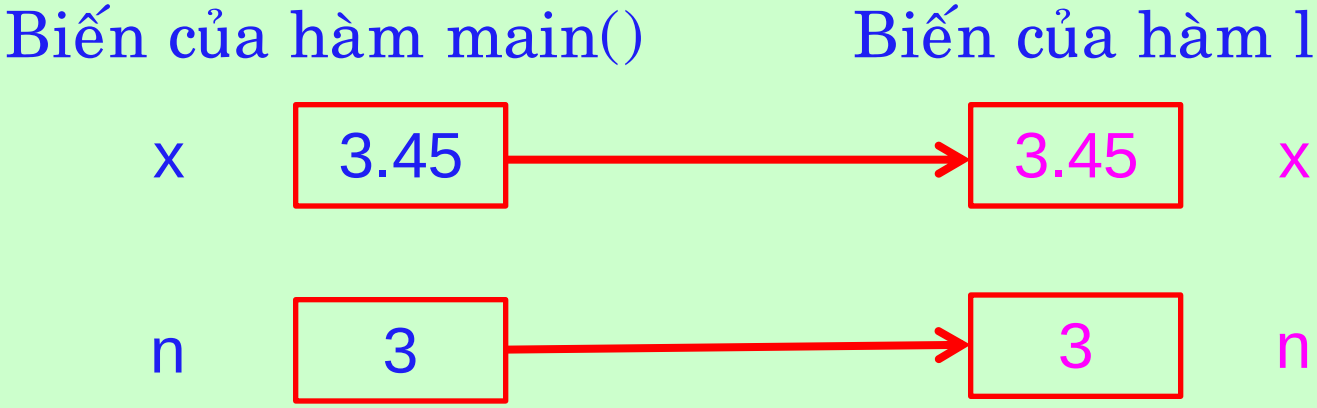
```
#include <stdio.h>
#include <conio.h>
double luy_thua(double x, int n);
main()
{       int n;
        double x, xn;
        clrscr();
        printf ("Moi nhap so tinh luy thua: ");
        scant ("%lf", &x);
```

9.3 ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

```
        printf ("Moi nhap so luy thua: ");
        scanf ("%d", &n);
        xn =luy_thua (x, n);
        printf("Ket qua: %5.2f luy thua %d bang: %7.2f\n",
x,n,xn);
        printf ("Tri cua so mu la %d", n);
        getch();}
double luy_thua(double x, int n)
{       double t = 1;
        for ( ; n > 0; n--)
        t *= x;
        return t;       }
```

9.3 ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

Khi gọi hàm lũy thừa, trị của biến x và n sẽ được chép vào cho hai đối số giả x và n, do đó ta có đồng thời các hộp biến như sau khi vào trong hàm luy_thua():



CHƯƠNG 9
HÀM

9.3 ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

Ví dụ:

Ta có thể gọi hàm *luy_thua()* và truyền cho hàm này một biểu thức:

```
xn = luy_thua( 3*a + x , 5);
```

Tuy nhiên, cách truyền tham số như trên không thể thay đổi trị của biến, mà điều này đôi khi lại cần thiết.

CHƯƠNG 9
HÀM

9.3 ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

Ví dụ: Viết chương trình dùng hàm nhập số liệu

```
#include <stdio.h>
#include <conio.h>
void nhap_tri (int a, int b);
main()
{       int a = 0, b = 0;
        clrscr();
        printf ("Truoc khi goi ham nhap_tri: a = %d, b = %d\n",
a, b);
        nhap_tri (a, b);
        printf("Sau khi goi ham nhap_tri a = %d, b = %d\n", a,
b);
        getch();
}
```

9.3 ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

```
void nhap_tri (int a, int b)
{
        printf ("Moi nhap hai so: ");
        scanf ("%d %d", &a, &b);
}
```

9.3 ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

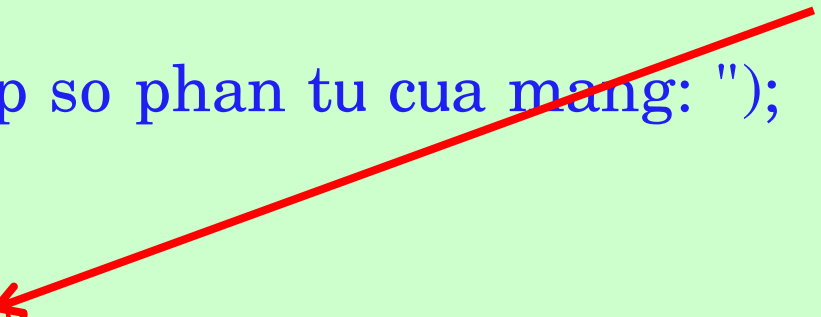
Ví dụ:
Thiết kế chương trình dùng hàm nhập mảng, tính tổng các phần tử và in ra màn hình kết quả.

```
#include <stdio.h>
#include <conio.h>
void nhap_tri (int a[], int n);
int tong (int a[], int n);
```

9.3 ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

```
main()
{       int a[100], n;
        int sum;
        clrscr();
        printf ("Moi nhap so phan tu cua mang: ");
        scanf (%d , &n);
        nhap_tri (a, n);
        sum = tong (a, n);
        printf ("Tong cac phan tu cua mang la: %d \n", sum);
        getch();
}
```

Giá trị của mảng có thể bị thay đổi trong hàm



9.3 ĐỐI SỐ CỦA HÀM - ĐỐI SỐ LÀ THAM TRỊ

```
void nhap_tri (int a[], int n)
{       int i;
        printf ("Moi nhap cac phan tu cua mang: ");
        for (i = 0; i <n; i++)
                scanf ("%d", &a[i]);
}
int tong (int a[], int n)
{       int i, s = 0;
        for (i = 0; i < n; s += a[i++])
                ;
        return s;}
```

Giá trị của mảng có thể bị thay đổi trong hàm

9.4 KẾT QUẢ TRẢ VỀ CỦA HÀM - LỆNH RETURN

Đối với C không có sự phân biệt giữa thủ tục (procedure) và hàm (function), mà thủ tục cũng được xem là một hàm mà không trả về giá trị nào cả. Để khai báo kiểu trả về từ hàm như vậy C đưa ra kiểu **void**, tạm gọi là kiểu không hiểu.

Ví dụ: so sánh 2 trường hợp sử dụng hàm

c = getch(); và getch();

hoặc

c = getche(); và getche();

9.4 KẾT QUẢ TRẢ VỀ CỦA HÀM - LỆNH RETURN

Trong chương trình, ta cũng biết lệnh **return** dùng để thực hiện việc trả trị của hàm về cho nơi gọi nó, dù trị này có được sử dụng hay không tùy nơi gọi.

Ví dụ:

Thiết kế hàm so sánh hai số.

9.4 KẾT QUẢ TRẢ VỀ CỦA HÀM - LỆNH RETURN

```
int so_sanh (int a, int b)
{       if (a >b)
        {       printf ("So %d lon hon so %d", a, b);
                return 1;       }
        else if (a == b)
        {       printf ("So %d bang so %d", a, b);
                return 0;       }
        else /* a <b */
        {       printf ("So %d nho hon so %d", a, b);
                return -1;      }
}
```

CHƯƠNG 9
HÀM

9.4 KẾT QUẢ TRẢ VỀ CỦA HÀM - LỆNH RETURN

```
main()
        {
                int so1, so2;
                clrscr();
                printf("Moi nhap hai so: ");
                scanf ("%d %d", &so1, &so2);
                so_sanh (so1, so2);
    getch();
        }
```

9.4 KẾT QUẢ TRẢ VỀ CỦA HÀM - LỆNH RETURN

```
main()
{
        int so1, so2;
        int kq;
        clrscr();
        printf ("Moi nhap hai so: ");
        scanf ("%d %d", &so1, &so2);
        kq = so_sanh (so1, so2);
```

9.4 KẾT QUẢ TRẢ VỀ CỦA HÀM - LỆNH RETURN

```
        switch (kq)
        {
         case -1:
          printf (", nen tri tuyet doi hieu hai so la: %d\n,so2-
so1");
          break;
         case 0:
         case 1:
          printf (", tri tuyet doi hieu hai so la %d\n , so1-so2");
          break;
        }
        getch();
}
```

9.4 KẾT QUẢ TRẢ VỀ CỦA HÀM - LỆNH RETURN

Ví dụ: Xét hàm sau (không trả về giá trị)

```
void so_sanh (int a, int b)
{       if (a > b)
                printf ("So %d lon hon so %d", a, b);
        else if (a == b)
                printf ("So %d bang so %d", a, b);
        else /* a < b */
                printf ("So %d nho hon s %d", a, b);
}
```

9.4 KẾT QUẢ TRẢ VỀ CỦA HÀM - LỆNH RETURN

Khi khai báo hàm mà ta không nêu cụ thể kiểu trả về của hàm, C mặc nhiên xem như hàm trả về kết quả là int. Ví dụ:

```
so_sanh (int a, int b)
        {       if (a > b)
                        return 1;
                else if (a == b)
                        return 0;
                else /* a < b */
                        return -1;              }
```

CHƯƠNG 9
HÀM

9.4 KẾT QUẢ TRẢ VỀ CỦA HÀM - LỆNH RETURN

Đối với các hàm có kiểu trả về trị khác int, khi khai báo cần phải trình bày đầy đủ các thành phần của hàm.

Khi gọi sử dụng hàm thì trong hàm gọi cần phải có nêu kết quả trả về của các hàm được gọi trong đó.

Kiểu khai báo kết quả này có thể được đặt bên ngoài tất cả các hàm để thông báo cho tất cả các hàm về trị trả về của nó, hoặc có thể được đặt trong hàm mà hàm sử dụng được gọi: **kiểu tên_hàm();**

9.4 KẾT QUẢ TRẢ VỀ CỦA HÀM - LỆNH RETURN

Ví dụ:

```
main()
        {       int so_sanh ();
                int so1, so2;
                clrscr();
                printf ("Moi nhap hai so: ");
                scanf ("%d %d", &so1, &so2);
                so_sanh (so1, so2);
                getch();                }
```

CHƯƠNG 9
HÀM

9.5 PROTOTYPE CỦA MỘT HÀM

Như vậy để một hàm có thể sử dụng trong một hàm khác thì trong hàm sử dụng phải có khai báo hàm cần sử dụng. Tuy nhiên khai báo này rất hạn chế ở chỗ không cho phép kiểm tra số đối số thật đưa vào hàm cũng như kiểu của đối số có phù hợp không

9.5 PROTOTYPE CỦA MỘT HÀM

Để khắc phục những lỗi trên, trong những phát triển sau này của C theo ANSI, người ta đưa ra khái niệm prototype của một hàm, đây thật sự là một dạng khai báo hàm mở rộng hơn, có dạng tổng quát như sau

kiểu tên_hàm (danh_sách_khai_báo_đối_số);

Ví dụ : int so_sanh (int a, int b);

void gptb1 (double a, double b, doubbe c);

char kiem_tra (double n);

CHƯƠNG 9
HÀM

9.5 PROTOTYPE CỦA MỘT HÀM

C cho phép khai báo prototype của hàm trong phần khai báo đối số chỉ cần có kiểu mà không cần có tên của đối số giả.

Ví dụ :

```
int so_sanh (int, int);
```

CHƯƠNG 9
HÀM

9.5 PROTOTYPE CỦA MỘT HÀM

Công dụng của prototype của hàm: prototype của một hàm ngoài việc dùng để khai báo kiểu của kết quả trả về từ một hàm, nó còn được dùng để kiểm tra số đối số.

Ví dụ :Nếu đã khai báo prototype

```
int so_sanh (int a, int b);
```

mà khi gọi hàm ta chỉ gửi một đối số như sau:

```
so_sanh (so2);
```

thì sẽ bị C phát hiện và báo lỗi

CHƯƠNG 9
HÀM

9.5 PROTOTYPE CỦA MỘT HÀM

Ví dụ: Chương trình sau luôn cho kết quả so sánh 2 số nhập vào là bằng

```
main()
        {       int so1, so2;
                int n;
                clrscr();
                printf(Moi nhap hai so: );
                scanf (%d %d, &so1, &so2);
                so_sanh (so2);
                getch();                }
```

CHƯƠNG 9
HÀM

9.5 PROTOTYPE CỦA MỘT HÀM

Chuyển kiểu của đối số: khi một hàm được gọi, mà hàm đó có prototype, các đối số được gởi cho hàm sẽ được chuyển kiểu bắt buộc theo kiểu của các đối số được khai báo trong prototype, sự chuyển kiểu này làm cho các đối số được sử dụng phù hợp với các phép toán trong thân hàm.

9.5 PROTOTYPE CỦA MỘT HÀM

Trường hợp mà sự chuyển kiểu không cho phép thực hiện thì C sẽ đưa ra các thông báo lỗi, hoặc một lời cảnh báo (warning).

9.5 PROTOTYPE CỦA MỘT HÀM

Đối với các hàm chuẩn trong thư viện C, prototype của chúng đã được C viết sẵn và để trong các file có phần mở rộng là .h, muốn lấy các prototype này vào chương trình ta cần ra chỉ thị bao hàm file .h chứa prototype của các hàm cần sử dụng vào đầu chương trình bằng lệnh tiền xử lý #include theo cú pháp sau:

 # include <file. h>

Ví dụ 3: #include <stdio.h>

CHƯƠNG 9
HÀM

9.6 HÀM ĐỆ QUY

C được gọi là một ngôn ngữ đệ quy vì C cho phép một hàm có thể gọi đến chính nó một cách trực tiếp, hoặc gián tiếp (tức là gọi qua trung gian một hàm khác), khi đó ta nói hàm đó có tính đệ quy (recursive).

Một giải thuật đệ quy sẽ dẫn đến một sự lặp đi lặp lại không kết thúc các thao tác, nhưng trong thực tế, chúng cần phải được kết thúc, sử dụng các điều kiện kết thúc đệ quy.

9.6 HÀM ĐỆ QUY

Ví dụ 3: Hàm đệ quy tính giai thừa n!

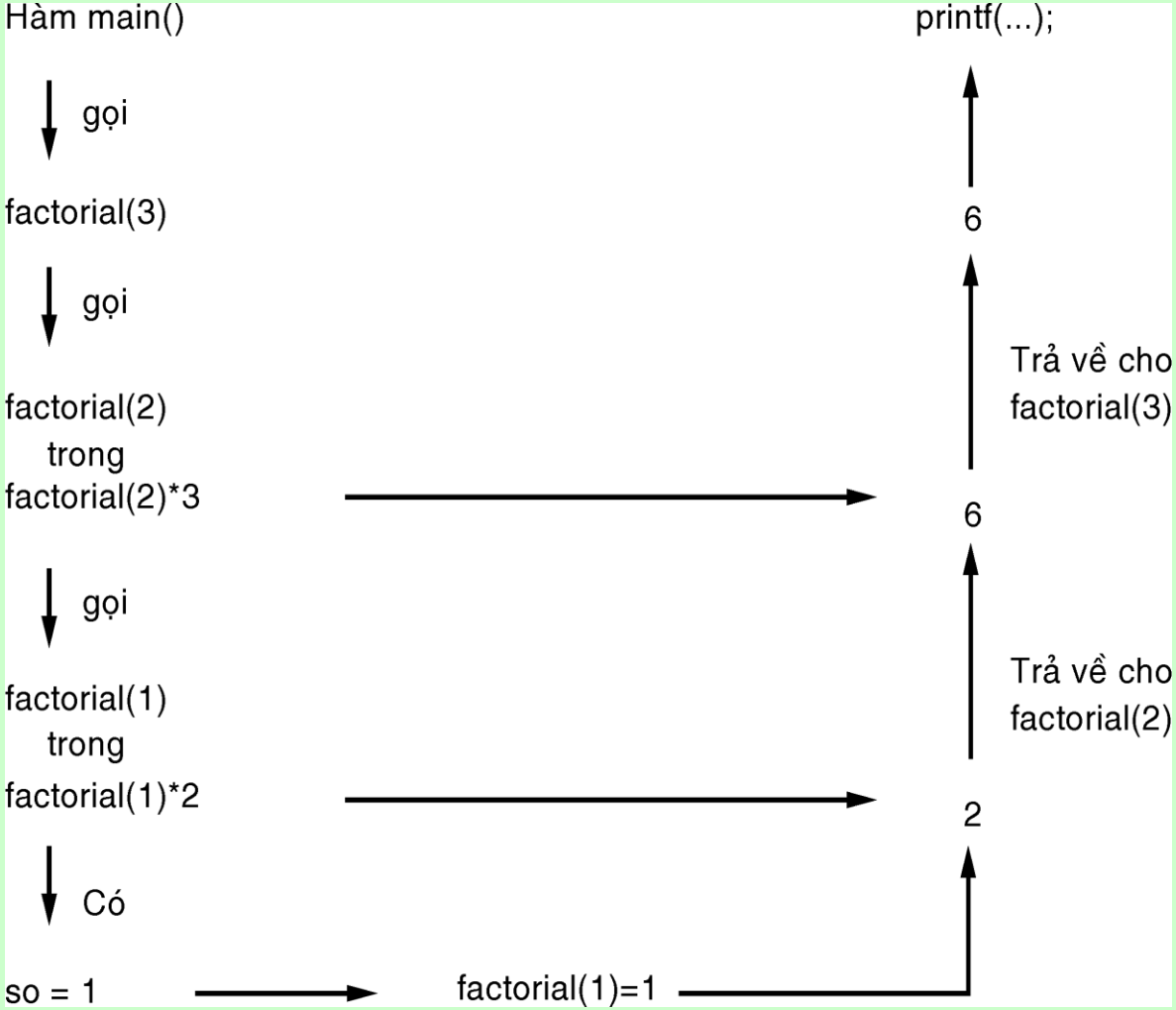
```
#include <stdio.h>
#include <conio.h>
long factorial (long so);
int main()
{       long so, kq = 0; clrscr();
        printf ("Moi nhap mot so khac 0: ");
        scanf ("%ld", &so);
        kq = factorial (so);
        printf ("Ket qua %ld! la %ld \n", so, kq);
        getch();
        return 0;}
```

9.6 HÀM ĐỆ QUY

```
long factorial (long so)
{
        if (so > 1)
                return (factorial(so - 1) * so);
        else
                return 1;
}
```

CHƯƠNG 9
HÀM

9.6 HÀM ĐỆ QUY



9.7 HIỆN THỰC HÀM TRONG C

Trong C, để gọi hàm ta cần ba bước cơ bản:

(1) các tham số từ nơi gọi được chuyển cho hàm được gọi và điều khiển được chuyển cho hàm được gọi,

(2) hàm được gọi thực hiện tác vụ,

(3) một giá trị trả về được gởi ngược lại cho nơi gọi hàm, và điều khiển được trả về cho nơi gọi.

9.7 HIỆN THỰC HÀM TRONG C

Một quy ước quan trọng mà chúng ta quy định cho cơ chế gọi là hàm được gọi nên độc lập với nơi gọi.

9.7 HIỆN THỰC HÀM TRONG C

9.7.1 Ngăn xếp thực thi (Run-time stack)

Trước khi tiến hành, đầu tiên chúng ta cần thảo luận về cách thức kích hoạt một hàm khi nó được gọi. Nghĩa là, khi một hàm bắt đầu thực thi, các biến cục bộ của nó phải được cấp các vị trí trong bộ nhớ.

Mẫu tin kích hoạt được định vị ở đâu trong bộ nhớ? Có hai chọn lựa như sau:

9.7 HIỆN THỰC HÀM TRONG C

9.7.1 Ngăn xếp thực thi (Run-time stack)

Chọn lựa 1: Bộ dịch có thể quy định một cách hệ thống một số vùng trống trong bộ nhớ cho mỗi hàm để chứa mẫu tin kích hoạt.

VD: Hàm A có thể được gán cho vùng bộ nhớ X để đặt mẫu tin kích hoạt của nó, hàm B lại có thể được gán cho vùng nhớ Y.

$\Rightarrow$cái gì xảy ra khi hàm A gọi chính nó?

Bản được gọi của hàm A sẽ ghi đè các biến cục bộ của hàm A, và chương trình sẽ không chạy như ta mong đợi.

9.7 HIỆN THỰC HÀM TRONG C

9.7.1 Ngăn xếp thực thi (Run-time stack)

Chọn lựa 2: Mỗi lần một hàm được gọi, một mẫu tin kích hoạt được định vị cho riêng nó trong bộ nhớ. Khi hàm đó trở về nơi gọi, vùng nhớ của mẫu tin kích hoạt đó sẽ được đòi lại để gán cho các hàm khác sau này.

$\Rightarrow$ Mỗi lần gọi một hàm đều lấy một vùng nhớ riêng cho các biến cục bộ của nó.

9.7 HIỆN THỰC HÀM TRONG C

9.7.1 Ngăn xếp thực thi (Run-time stack)

Chọn lựa 2: Ví dụ, nếu hàm A gọi hàm A, mẫu tin kích hoạt của bản được gọi sẽ được định vị ở một vị trí khác với vị trí của bản gọi ban đầu trong bộ nhớ, và dĩ nhiên là hai mẫu tin kích hoạt này độc lập nhau.

Có một tác nhân làm giảm bớt tính phức tạp của việc thực hiện chọn lựa 2: quá trình gọi của hàm (tức hàm A gọi hàm B, hàm B lại gọi hàm C, …) có thể được theo dõi bằng một cấu trúc dữ liệu ngăn xếp.

9.7 HIỆN THỰC HÀM TRONG C

9.7.1 Ngăn xếp thực thi (Run-time stack)

Ví dụ:

```
1        int main()
2        {
3            int a;
4            int b;
5
6              …
7            b = Watt (a);
8            b = Volta (a, b);
9         }
```

9.7 HIỆN THỰC HÀM TRONG C

9.7.1 Ngăn xếp thực thi (Run-time stack)

Ví dụ:

```
11          int Watt (int a)
12          {
13              int w;
14
15                …
16              w = Volta (w, 20);
17
18              return w;
19           }
```

9.7 HIỆN THỰC HÀM TRONG C

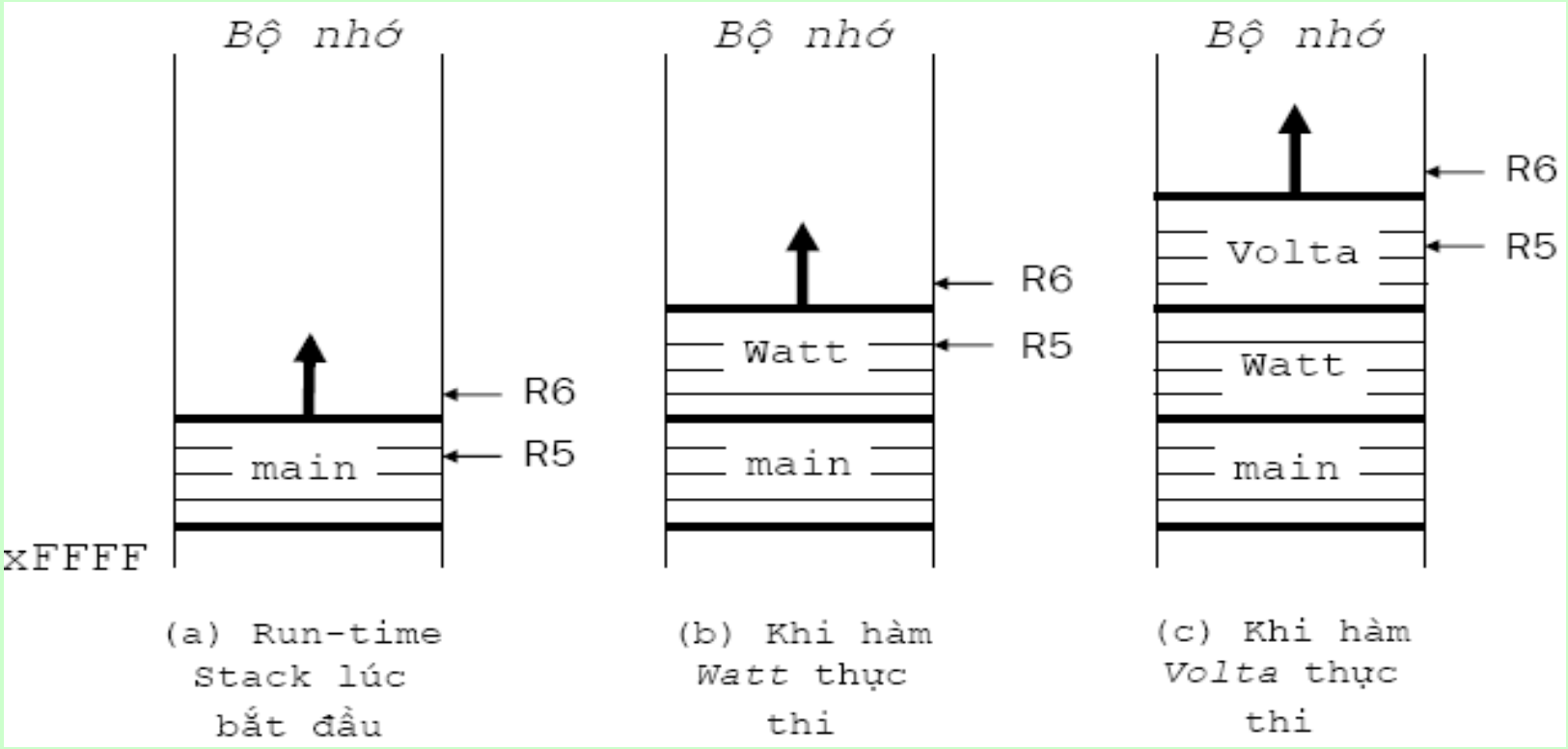
9.7.1 Ngăn xếp thực thi (Run-time stack)

Ví dụ:

```
21          int Volta (int q, int r)
22          {
23              int k;
24              int m;
25
26                …
27              return w;
28           }
```

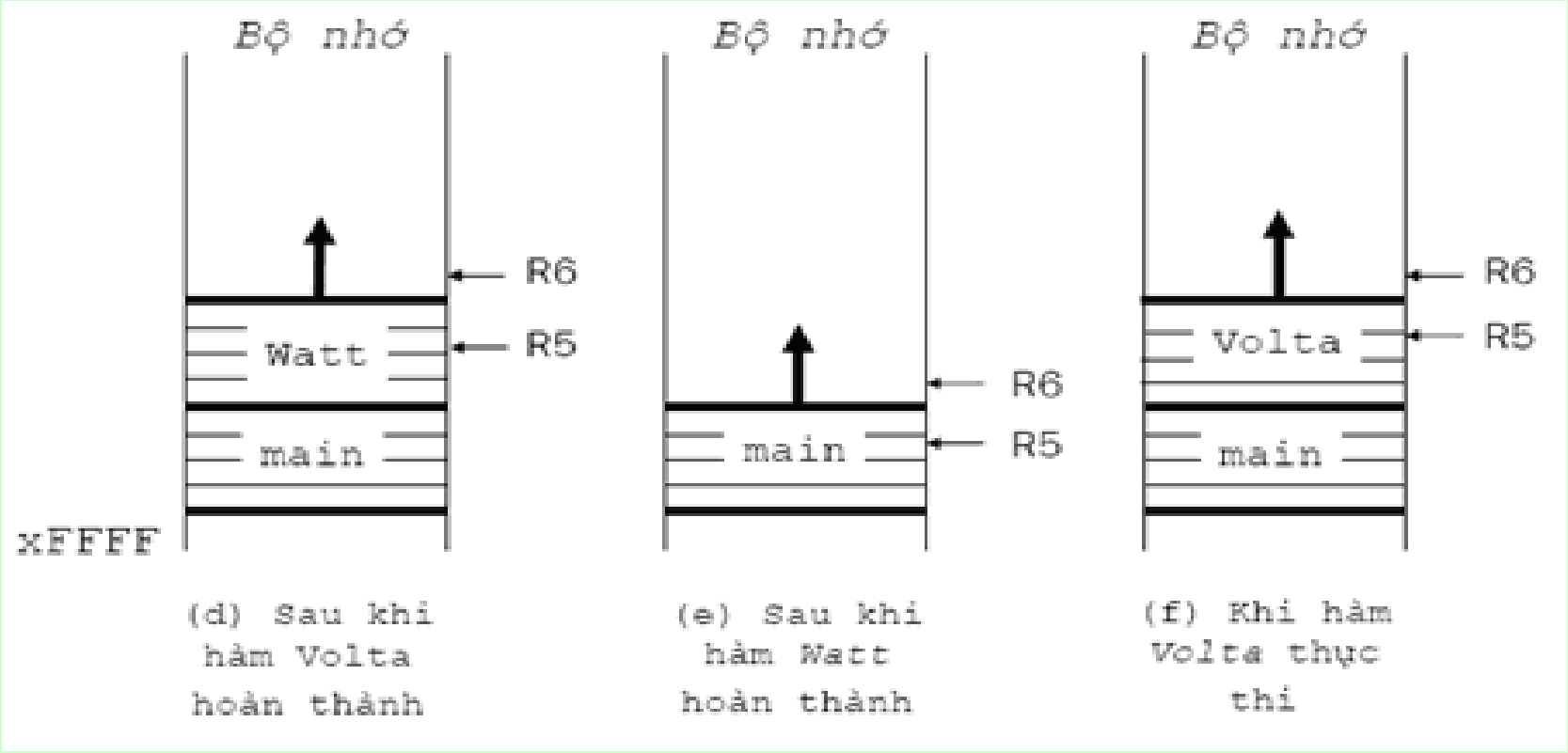
9.7 HIỆN THỰC HÀM TRONG C

9.7.1 Ngăn xếp thực thi (Run-time stack)



9.7 HIỆN THỰC HÀM TRONG C

9.7.1 Ngăn xếp thực thi (Run-time stack)



9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

Để hoàn thành tất cả các việc này, các bước sau cần phải được thực hiện:

•Đầu tiên, code của hàm gọi (caller) sẽ chép các đối số của nó vào vùng bộ nhớ để hàm được gọi (callee) có thể truy xuất được.

•Thứ hai, code ở nơi bắt đầu trong hàm được gọi đẩy mẫu tin kích hoạt của nó vào stack và lưu các thông tin trạng thái của biến cục bộ, thanh ghi, … để khi điều khiển trả về cho nơi gọi, thì đối với nơi gọi mọi thứ như không có gì thay đổi, từ các biến cục bộ tới các thanh ghi.

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- Thứ ba, hàm được gọi thực thi tác vụ của nó.
- Thứ tư, khi hàm được gọi hoàn thành việc của nó, mẫu tin kích hoạt của nó được lấy ra khỏi ngăn xếp thực thi (run-time stack) và điều khiển được trả về cho nơi gọi.
- Sau cùng, một khi điều khiển được trả về cho code nơi gọi, code thực thi sẽ truy tìm trị mà hàm được gọi trả về.

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

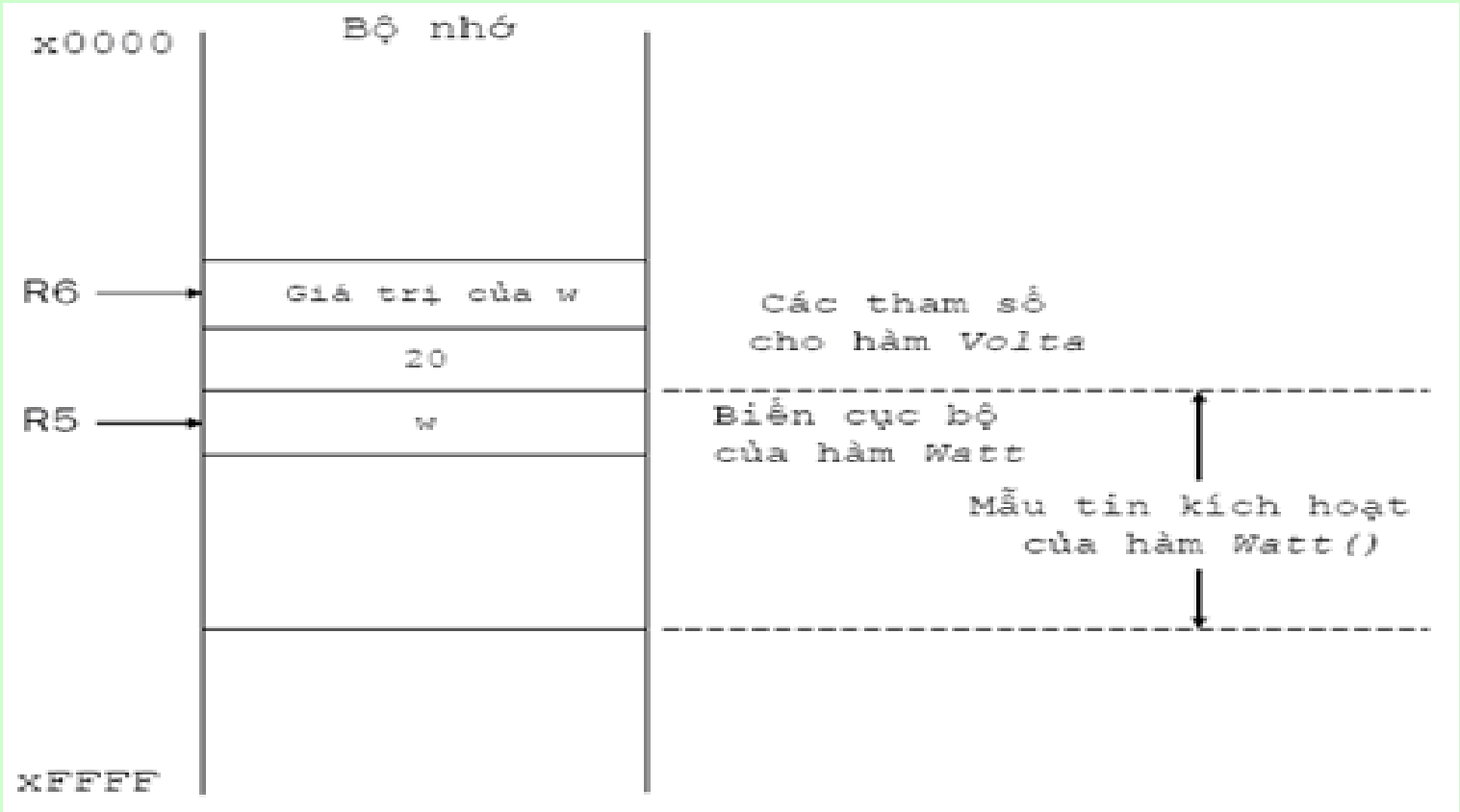
Gọi hàm

Trong mệnh đề w = Volta(w, 20); hàm Volta() được gọi với hai đối số. Sau đó giá trị trả về từ hàm Volta() sẽ được gán cho biến nguyên cục bộ w. Trong khi dịch việc gọi hàm này, bộ dịch tạo ra code LC-3 làm các việc sau:

1) Truyền giá trị của hai đối số của hàm Volta() bằng việc đẩy chúng trực tiếp vào đỉnh của ngăn xếp thực thi (run-time stack) mà địa chỉ đang được chứa trong thanh ghi R6.

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực



9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

Gọi hàm

2) Chuyển điều khiển sang hàm Volta() nhờ lệnh JSR.

Mã LC-3 thực hiện gọi hàm này như sau:

```
; đẩy đối số thứ hai vào stack
AND   R0,  R0,  #0        ; R0 ← 0
ADD   R0,  R0,  #20       ; R0 ← 20
ADD   R6,  R6,  #-1
STR   R0,  R6,  #0        ; Push 20
```

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

Gọi hàm

```
;       đẩy     đối     số      thứ     nhất    vào     stack
        LDR     R0,   R5,   #0   ;  Lấy trị của biến cục bộ w
        ADD     R6,   R6,   #-1
        STR     R0,   R6,   #0    ;  Push  trị này vào stack
    ;  gọi chương trình con
       JSR      Volta
```

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Thực hiện hàm được gọi**

Lệnh được thực hiện ngay sau lệnh JSR trong hàm Watt() là lệnh đầu tiên trong hàm được gọi Volta().

Code ở chỗ bắt đầu của hàm được gọi xử lý một số thao tác liên quan tới việc gọi hàm. Việc đầu tiên là định vị bộ nhớ cho trị trả về bằng cách hàm được gọi sẽ đẩy một ô nhớ vào stack để chiếm chổ qua việc giảm con trỏ stack . Và vị trí này sẽ được ghi vào giá trị cần trả về trước khi điều khiển trả về cho hàm gọi.

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

•Thực hiện hàm được gọi

Kế tiếp, hàm được gọi lưu các thông tin về hàm gọi để khi việc gọi kết thúc, hàm gọi sẽ lấy lại điều khiển chương trình một cách đúng đắn. Đặc biệt, chúng ta cần lưu địa chỉ trở về của hàm gọi đang được chứa trong thanh ghi R7 và con trỏ khung của hàm gọi đang được chứa trong thanh ghi R5. Một điều quan trọng là cần chép sao con trỏ khung của hàm gọi mà ta gọi là liên kết động, để khi điều khiển trả về cho nơi gọi thì nơi gọi sẽ có thể truy xuất trở lại các biến cục bộ của nó.

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Thực hiện hàm được gọi:**

Nếu một trong địa chỉ trở về hoặc liên kết động bị sai, thì chúng ta sẽ gặp sai sót khi tiếp tục thực thi hàm gọi khi hàm được gọi hoàn thành. Nên chúng ta cần phải sao lưu cả hai thứ này vô bộ nhớ.

Sau cùng, khi tất cả điều này được thực thi xong, hàm được gọi sẽ định vị đủ không gian trong stack cho các biến cục bộ của nó bằng việc chỉnh trị cho R6, và nó sẽ đặt R5 chỉ tới nền của các biến cục bộ này.

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Thực hiện hàm được gọi**

Danh sách các tác vụ phải diễn ra lúc bắt đầu một hàm:

- Hàm được gọi lưu không gian trong stack cho trị trả về. Trị trả về được định vị ngay trên đỉnh các tham số của hàm được gọi.
- Hàm được gọi đẩy một bản sao của địa chỉ trở về trong thanh ghi R7 vô stack.
- Hàm được gọi đẩy một bản sao của liên kết động (con trỏ khung của hàm gọi) trong R5 vô stack.

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Thực hiện hàm được gọi**

- Hàm được gọi định vị đủ không gian trong stack cho các biến cục bộ của nó và chỉnh thanh ghi R5 chỉ tới nền của danh sách biến cục bộ và thanh ghi R6 chỉ tới đỉnh stack.

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Thực hiện hàm được gọi**

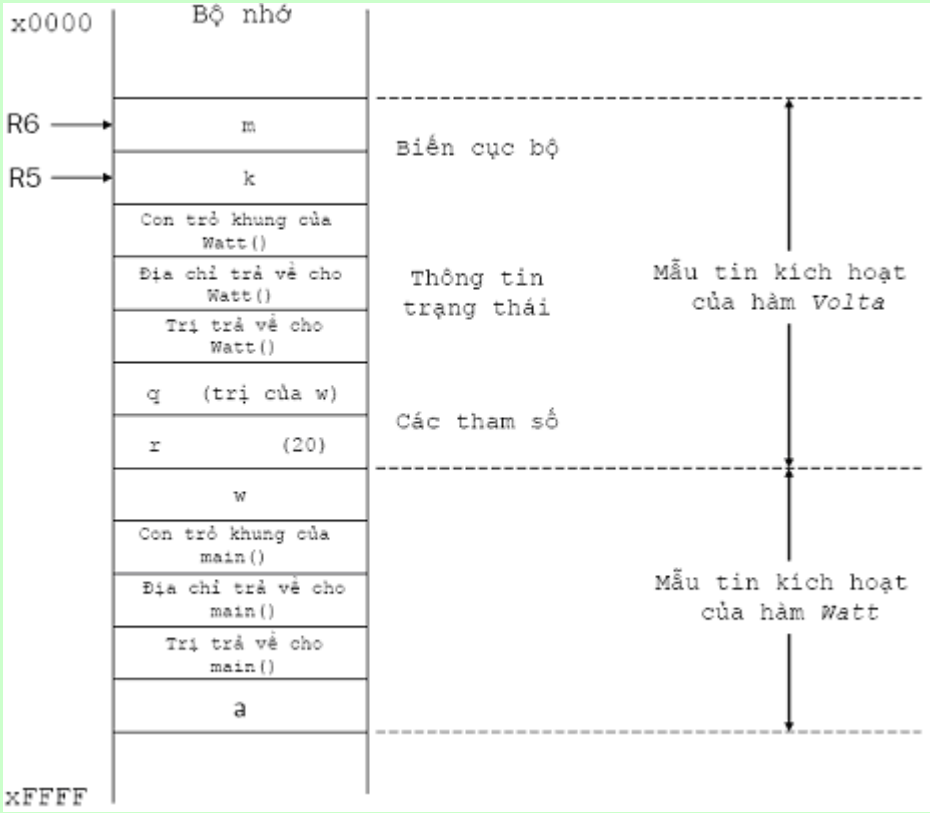
Đoạn code hoàn thành việc này cho hàm Volta() như sau:

```
ADD   R6,   R6,   #-1   ;  dành không gian cho trị trả về
ADD   R6,   R6,   #-1
STR   R7,   R6,   #0    ;  đẩy R7 (địa chỉ trở về) vô stack
ADD   R6,   R6,   #-1   ;  đẩy liên kết động vô stack (con trỏ
khung của hàm gọi)
STR   R5,   R6,   #0    ;  đẩy R5 vô stack
ADD   R5,   R6,   #-1   ;  đặt con trỏ khung mới
ADD   R6,   R6,   #-2   ;  định vị không gian cho các biến cục
bộ
```

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Thực hiện hàm được gọi**



9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Kết thúc hàm được gọi**

Một khi hàm được gọi hoàn thành công việc của nó, nó phải làm thêm một số tác vụ nữa trước khi trả điều khiển về cho hàm gọi. Đầu tiên, ta cần có cơ chế để một hàm trả về trị một cách thích hợp cho nơi gọi. Thứ hai, hàm được gọi phải lấy ra khỏi stack mẫu tin kích hoạt của nó.

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Kết thúc hàm được gọi**

Như vậy, ta có các việc như sau:

1) Nếu có một trị trả về, nó cần phải được ghi vô đầu vào trị trả về của mẫu tin kích hoạt.
2) Các biến cục bộ phải được lấy ra khỏi stack.
3) Liên kết động cần được phục hồi.
4) Địa chỉ trả về phải được phục hồi.
5) Lệnh RET trả điều khiển về cho hàm gọi.

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Kết thúc hàm được gọi**

Hàm Volta() khi lệnh return k; được thi hành như sau:

```
LDR    R0,R5,#0    ;   nạp biến cục bộ k
STR    R0,R5,#3    ;   lưu nó vô đầu vào trị trả về
ADD    R6,R5,#1    ;   pop các biến cục bộ
LDR    R5,R6,#0    ;   pop liên kết động, con trỏ khung -> R5
ADD    R6,R6,#1    ;   R6 chỉ tới ô địa chỉ trả về
LDR    R7,R6,#0    ;   lưu địa chỉ trả về cho R7
ADD    R6,R6,#1    ;   R6 chỉ tới ô trị trả về
RET                ;   trả điều khiển về cho nơi gọi
```

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Trở về hàm gọi**

Sau khi hàm được gọi thực thi lệnh RET, điều khiển được trả ngược về cho hàm gọi. Trong một số trường hợp, không có trị trả về (tức hàm được gọi được khai báo với kiểu *void*) và, trong một số trường hợp khác, hàm gọi bỏ qua các trị trả về (như khi ta gọi getch();).

Đặc biệt, có hai thao tác phải được thi hành:

1) Trị trả về (nếu có) được lấy ra khỏi stack.
2) Các đối số cần được lấy ra khỏi stack.

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Trở về hàm gọi**

Đoạn code LC-3 sau lệnh JSR sẽ như sau:

```
JSR  Volta

LDR     R0,    R6,    #0     ;   nạp trị trả về ở đỉnh stack
STR     R0,    R5,    #0     ;   w = Volta(w, 20);
ADD     R6,    R6,    #1     ;   pop  trị trả về
ADD     R6,    R6,    #2     ;   pop các đối số
```

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Sao lưu ở nơi gọi và sao lưu ở nơi được gọi**

  Trong khi thực thi một hàm, R0 tới R3 có thể chứa các giá trị tạm thời, là một phần của các thao tác tính toán đang chạy. Thanh ghi từ R4 tới R7 được dành cho các mục đích khác: thanh ghi R4 là con trỏ chỉ tới vùng dữ liệu toàn cục, R5 là con trỏ khung, R6 là con trỏ stack, và R7 được dùng để giữ địa chỉ trở về. Nếu chúng ta gọi một hàm, dựa theo quy ước gọi hàm chúng ta đã mô tả thanh ghi R4 tới R7 không thay đổi hay thay đổi theo các cách đã được xác định trước.

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Sao lưu ở nơi gọi và sao lưu ở nơi được gọi**

Nhưng cái gì xảy ra cho các thanh ghi R0, R1, R2, và R3? Tổng quát mà nói, chúng ta muốn chắc chắn rằng hàm được gọi sẽ không chép đè chúng. Để làm được điều này, các quy ước gọi hàm cụ thể theo một trong hai phương cách: (1) Nơi gọi sẽ sao lưu các thanh ghi bằng cách đẩy chúng vô mẫu tin kích hoạt của nó. Đây được gọi là sao lưu nơi gọi (*caller save*). Khi điều khiển được trả về cho nơi gọi, nơi gọi sẽ khôi phục lại các thanh ghi này bằng việc lấy chúng ra khỏi stack.

9.7 HIỆN THỰC HÀM TRONG C

9.7.2 Quá trình hiện thực

- **Sao lưu ở nơi gọi và sao lưu ở nơi được gọi**

(2) Hàm được gọi có thể sao lưu các thanh ghi này bằng cách thêm vô bốn vùng tin trong vùng thông tin trạng thái của mẫu tin của nó. Đây chính là sao lưu nơi được gọi (*callee save*). Khi nơi được gọi được khởi tạo, nó sẽ sao lưu R0 tới R3 và R5 và R7 vô vùng thong tin trạng thái và khôi phục các thanh ghi này lại trước khi trở về nơi gọi.

9.7 HIỆN THỰC HÀM TRONG C

9.7.3 Tóm lại (GT)

9.8 KIỂM TRA VÀ SỬA LỖI

9.8.1 Giới thiệu

9.8.2 Các dạng lỗi

9.8.2.1 Lỗi cú pháp

Ví dụ 10.20:

```
#include <stdio.h>
main()
      {  int i, i2
for (i = 0; i < 10; i++)
         {  i2 = i * i;
     printf ("%d x %d  =  %d\n", i, i, i2);
 }     }
```

9.8 KIỂM TRA VÀ SỬA LỖI

9.8.2 Các dạng lỗi

9.8.2.2 Lỗi ngữ cảnh

Ví dụ 10.21:

```
#include <stdio.h>
main()
{
  int i, i2;
  for (i = 0; i < 10; i++)
  i2 = i * i;
  printf (“%d x %d  =  %d\n”, i, i, i2);
}
```

9.8 KIỂM TRA VÀ SỬA LỖI

9.8.2 Các dạng lỗi

9.8.2.2 Lỗi ngữ cảnh

Ví dụ 10.22: Chương trình tính tổng 1+2+ …+ *n*, với *n* là trị nguyên dương được nhập từ bàn phím. Chương trình được thiết kế với hàm tính tổng.

```
#include <stdio.h>
    int tinh_tong (int n);              // prototype của hàm
main()
{   int so;
    int tong;
    printf ("Moi nhap mot so nguyen duong: ");
```

9.8 KIỂM TRA VÀ SỬA LỖI

9.8.2 Các dạng lỗi

9.8.2.2 Lỗi ngữ cảnh

```
scanf ("%d", &so);
        tong = tinh_tong (n);
        printf ("Tong tu 1 + … + %d = %d\n", n, tong);    }
  int tinh_tong (int n)
  {   int tong;
      int i;
      for (i = 1; i <= n; i++)
         tong += i;
      return tong;        }
```

9.8 KIỂM TRA VÀ SỬA LỖI

9.8.2 Các dạng lỗi

9.8.2.3 Lỗi giải thuật

Ví dụ 10.23:

```
#include <stdio.h>
#include <conio.h>
void gptb1 (double a, double b);
main()
{    double a, b;   clrscr();
printf ("Nhap 2 he so phuong trinh bac nhat: ");
scanf ("%lf %lf", &a, &b); gptb1 (a, b);
getch();  }
```

9.8 KIỂM TRA VÀ SỬA LỖI

9.8.2 Các dạng lỗi

9.8.2.3 Lỗi giải thuật

Ví dụ 10.23:

```
void gptb1 (double a, double b)
    {     printf ("Phuong trinh bac nhat ");
   printf ("co 1 nghiem: x = %5.2f \n", -b/a);
    }
```

9.8 KIỂM TRA VÀ SỬA LỖI

9.8.3 Kiểm tra chương trình

9.8.3.1 Kiểm tra hộp đen

Với kỹ thuật hộp đen này, chúng ta có thể kiểm tra xem dữ liệu nhập và xuất của chương trình có phù hợp không, mà không cần phải biết cấu trúc lệnh trong chương trình.

9.8 KIỂM TRA VÀ SỬA LỖI

9.8.3 Kiểm tra chương trình

9.8.3.1 Kiểm tra hộp trắng

Kỹ thuật kiểm tra hộp trắng tổng quát sử dụng các code tìm lỗi được đặt trong chương trình một cách có tính toán. Các code này có thể kiểm tra các điều kiện để chỉ ra lỗi sai làm chương trình không thực thi đúng. Khi có một lỗi sai được tìm ra, code sẽ in ra một thông điệp cảnh báo, hiển thị một vài thông tin có giá trị về lỗi, hoặc là làm cho chương trình kết thúc sớm. Vì các code tìm lỗi này xác nhận các điều kiện cụ thể cần giữ trong suốt quá trình chương trình thực thi, nên chúng ta gọi các code này là sự xác nhận.

CHƯƠNG 9
HÀM

9.8 KIỂM TRA VÀ SỬA LỖI

9.8.4 Gỡ rối chương trình (*debuging*)

9.8.4.1 Kỹ thuật phi thể thức

9.8.4.2 Gỡ rối cấp nguồn

- Điểm ngắt (*Breakpoints*)
- Thực thi từng bước (*Single-Stepping*)
- Hiển thị giá trị

9.8 KIỂM TRA VÀ SỬA LỖI

9.8.5 Tính đúng đắn khi lập trình

9.8.5.1 Xác định thật rõ các chi tiết

Ví dụ 10.25:

```
int Giaithua (int n)
 {    int i;
      int ketqua = 1; // Kiểm tra trị nhập
      if (n < 1 || n > 31)
     {    printf (“Trị sai, cần trị trong khoảng 1-31.\n”);
          ketqua = -1;   }
      else
          for (i = 1; i <= n; i++)     ketqua *= i;
      return ketqua;           }
```

CHƯƠNG 9
HÀM

9.8 KIỂM TRA VÀ SỬA LỖI

9.8.5 Tính đúng đắn khi lập trình

9.8.5.2 Thiết kế theo từng khối

9.8.5.3 Lập trình dự phòng

CHƯƠNG 9
HÀM

BÀI TẬP CUỐI CHƯƠNG

1. Thiết kế hàm giải phương trình
   $ax^2 + bx + c = 0$.
   Viết chương trình chính sử dụng hàm này.
2. Thiết kế hàm in ra màn hình chuỗi số Fibonaci, với thông số nhập là một số ngẫu nhiên trong số từ 1 đến 100.
   Hướng dẫn: Sử dụng hàm random() và randomize() để tạo số ngẫu nhiên. Dãy Fibonacci : 0, 1, 1, 2, 3...

CHƯƠNG 9
HÀM

BÀI TẬP CUỐI CHƯƠNG

3. Thiết kế hàm tính các biểu thức sau đây:

$$T = \frac{1}{1!} + \frac{1+2}{2!} + \ldots + \frac{1+2+\ldots+n}{n!}$$

$$S = (1)! + (1+2)! + \ldots + (1+\ldots+n)!$$

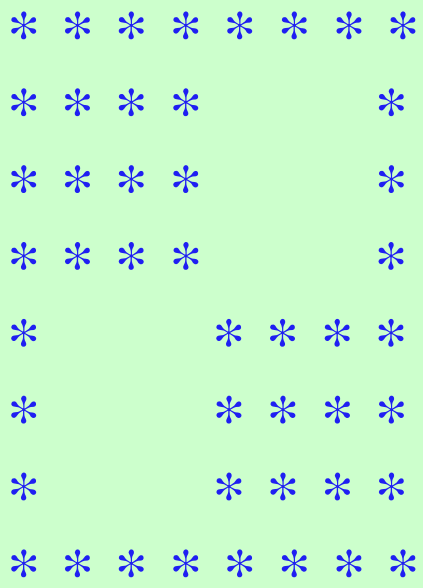
4. Thiết kế hàm vẽ ra màn hình hình sau:

```
        *
      * * *
    * * * * *
  * * * * * * *
```

n nhập

BÀI TẬP CUỐI CHƯƠNG

5. Thiết kế hàm và vẽ ra màn hình hình sau:



n nhập là số chẵn

BÀI TẬP CUỐI CHƯƠNG

6. Viết chương trình thiết kế hai hàm max() và min() cho phép tìm số lớn nhất và nhỏ nhất trong loạt số đã nhập. Loạt số nhập này sẽ kết thúc việc nhập bằng việc nhấn phím <F6>

7. Viết một chương trình cho phép nhập một chuỗi. Hãy thiết kế một hàm cho phép cắt chuỗi đó ra làm nhiều từ và in ra màn hình mỗi từ trên một hàng.

BÀI TẬP CUỐI CHƯƠNG

8. Viết một hàm nhận một chuỗi và cho phép đảo chuỗi đó, in ra kết quả.

9. Viết một hàm cho phép nhận một số nguyên dương. In ra màn hình ký số thứ n tính từ bên phải qua của số đó.

Ví dụ:

Nhập: 12345 4

Xuất: 2

BÀI TẬP CUỐI CHƯƠNG

10. Thiết kế một hàm đệ quy cho phép nhận một số nguyên dương, in ra màn hình số đó ở dạng nhị phân.

11. Viết hàm đệ quy tính x^n.

12. Viết một hàm nhận một số dương có phần lẻ và in ra màn hình phần nguyên và phần lẻ riêng biệt.

13. Viết chương trình với hàm tính tổng sau:

$S=1-x^2/2!+x^4/4!-...x^n/n!$

14. Tương tự như bài 10.13, nhưng tính tổng sau:

$S=1-x^3/3!+x^5/5!-...xn/n!$