CHƯƠNG 14:
GIỚI THIỆU LẬP TRÌNH C++

15.1 Lập trình hướng đối tượng
15.2 Constructor và Destructor
15.3 Toán tử New và Delete
15.4 Sự thừa kế dữ liệu
15. 5 Từ khóa static
15.6 Hàm ảo
15.7 Tham khảo trong C++
15.8 Một số điểm khác biệt chính giữa C và C++
15.9 Một số chương trình ví dụ

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Việc khác nhau chủ yếu giữa C++ và C là C++ đưa ra khái niệm **đối tượng**.

C++ đưa ra khái niệm "module mã" bằng các lớp, ***class***, được gọi là đối tượng, ***object***.

Một ***class*** là một cấu trúc ***những thông tin thêm*** vào bao gồm: sự giấu thông tin và tầm vực truy cập, các khai báo prototype hàm và khai báo biến tỉnh.

C++ cho phép lớp khai báo các ***phần thấy được***, tức bên ngoài truy cập được, hay còn được gọi là toàn cục, và ***các phần riêng*** của lớp, còn được gọi là cục bộ.

Thấy được (toàn cục)	Riêng (cục bộ)
Sự tồn tại của các cấu trúc Các hàm giao tiếp	Các dữ liệu trong cấu trúc Các hàm trong đối tượng Các khởi tạo (constructor) Các bộ hủy (destructor)

Đoạn mã nguồn viết bằng mã C :

Xét cấu trúc ***player_t*** mang các thông tin của một game thủ trong trò chơi:

```
struct player_t
{
    char * name;          /*  tên đăng nhập về game thủ   */
    char *password;       /*  password   */
    int num_palyed;       /*  số lần chơi  */
    int win_guesses[13]; /* số lần đoán trúng là thắng */
    double win_percent; /* % của số lần thắng */
};
```

Trong C, chúng ta khai báo một hàm được gọi khi game thủ thắng trò chơi:

void player_win (player_t * p, int num_guesses);

Hàm này sẽ được gọi như sau:

```
player_t  player1;
        /*   một vài đoạn mã ở đây   */
player_win (&player1, 5); /*   game thủ player1 thắng trò chơi sau 5 lần đoán   */
```

Đoạn mã nguồn trên viết lại bằng đối tượng lớp mã C++ :

```
class player_t
{
    public:
        void player_win (int num_guesses);
    private:
        char * name;              /*  tên đăng nhập của  game thủ   */
        char *password;           /*  password   */
        int num_palyed;           /*  số lần chơi  */
        int win_guesses[13];      /*  số lần đoán trúng là thắng  */
        double win_percent;       /*  phần trăm của số lần thắng  */
};
```

Đoạn mã nguồn trên viết lại bằng đối tượng lớp mã C++ :

Để thực hiện việc gọi hàm trong C++:

```
        player_t * p;       /*   pointer tới class player  */
/*  một vài đoạn mã ở đây  */
        p->player_win (5);
```

CONSTRUCTOR VÀ DESTRUCTOR

C++ cho phép chúng ta khai báo các đoạn mã để bất cứ khi nào muốn chúng ta đều có thể khởi tạo hay hủy một đối tượng.
Đoạn mã như vậy được gọi là bộ tạo, ***constructor***, dùng để khởi tạo đối tượng, và bộ hủy, ***destructor***, để hủy đối tượng.

CONSTRUCTOR VÀ DESTRUCTOR

Thí dụ:

```
class player_t
      {
          public:
              player_t (); /*  constructor  */
              ~player_t (); /*   destructor   */
              void player_win (int num_guesses);
           private:
             char * name;                /*  tên đăng nhập của  game thủ   */
             char *password;            /*  password   */
             int num_palyed;            /*  số lần chơi  */
             int win_guesses[13];       /*  số lần đoán trúng là thắng  */
             double win_percent;        /*  phần trăm của số lần thắng  */
      };
```

CONSTRUCTOR VÀ DESTRUCTOR

```
/*  Constructor*/
player_t :: player_t ()
      {   num_played] = 0;
          for (int i = 0; i < 13; i++ )
                win_guesses [i] = 0;
          win_percent = 0;    name = new char [10];
          password = = new char [10];      }
/*  Destructor */
       player_t :: ~player_t ()
       {
          delete []  name; /*  xóa vùng nhớ động xin trong constructor  */
          delete []  password;
       }
```

CONSTRUCTOR VÀ DESTRUCTOR

Thí dụ, xét hàm sau đây:

```
void play_round ()
{
    player_t  player1;  /*  mặc nhiên gọi constructor player_t  */
    player_t  player2;  /*  mặc nhiên gọi constructor player_t  */

        /*   đoạn mã chơi trò chơi  */
}
```

TOÁN TỬ NEW VÀ DELETE

Bên cạnh việc dùng ***malloc*** và ***free***, chúng ta có thể định vị bộ nhớ động bằng việc dùng các toán tử ***new*** và ***delete***.

Toán tử ***new*** sẽ gọi hàm ***malloc*** khi bộ tạo được thực hiện, toán tử ***delete*** sẽ gọi hàm ***free*** khi bộ hủy được gọi.

SỰ THỪA KẾ DỮ LIỆU

C++ cung cấp khái niệm ***thừa kế dữ liệu***. Việc thừa kế này rất hữu ích khi ta muốn tạo ra các đối tượng mà có thể chia sẻ thông tin với nhau.

Xem thêm sách giáo khoa

TỪ KHÓA STATIC

Từ khóa ***static*** khi được sử dụng trước một khai báo hàm trong đối tượng sẽ báo hàm đó không phải là phương thức của đối tượng.

```
class reference_t
  {       public:
          static reference_t * find_author (const char * find);
        private:
          author_list * author_list;
          char * title;
          int year;
          reference_t * next;
  };
```

HÀM ẢO

Việc khai báo hàm ảo sẽ được thực bằng từ khoá ***virtual*** :

```
virtual void print_all_cites ();
reference_t *  r;
r->print_all_cites ();
```

Bây giờ nếu chúng ta viết

```
book_t * b;
b->print_all_cites ();
```

thì hàm **book_t::print_all_cites** sẽ được gọi thay vì hàm **reference_t::print_all_cites**.

HÀM ẢO

Thí dụ dưới đây trình bày cách khai báo các lớp reference_t và book_t.

```
class reference_t
    {public:
            void print_reference_cites ()
            {printf ("Title: %s\n", title);
              printf ("Year: %d\n", year);}
            virtual void print_all_cites ()
            {printf_reference_cites (); }
        private:
            author_list *  author_list;
            char *  title;
            int  year;
            reference_t *  next; };
```

HÀM ẢO

```
class book_t : public refernce_t
    {
        public:
            virtual void print_all_cites ()          /*  (2)  */
            {   printf_reference_cites ();        /*  phương thức được thừa
kế từ lớp reference_t  */
                printf ("Publisher: %s\n", publisher);
                printf ("ISBN: %d\n", ISBN);
            }
        private:
            char *  publisher;                       /*  tên nhà xuất bản */
            char *  address;                      /*  địa chỉ nhà xuất bản */
            double  ISBN;                            /*  số ISBN */
     };
```

HÀM ẢO

Chúng ta hãy xem mỗi hàm sẽ in ra cái gì trong 2 trường hợp sau:

Trường hợp 1:

```
reference_t * r;
r-> print_all_cites ();
```

Trường hợp 2:

```
book_t * b;
b-> print_all_cites ();
```

THAM KHẢO TRONG C++

Trong C++, một tham khảo là một con trỏ mặc nhiên ám chỉ.

Thí dụ:

```
 int val;
int & val_ref = &val/*  Khai báo một tham khảo  */
val_ref = 10;        /*   tương tự như *val_ptr = 10 */
```

Một tham khảo thường được sử dụng trong khái niệm quá tải của toán tử (operator overload).

THAM KHẢO TRONG C++

Thí dụ, chúng ta tạo ra một lớp mới để biểu diễn một số phức ***complex_t***, như sau:

```
complex_t  x, y, z;
z  = x + y;
```

Nếu chúng ta làm quá tải thao tác cộng (+), chúng ta có thể khai báo hàm quá tải như sau:

```
complex_t operator+(const complex_t & a, const complex_t & b);
```

Một số điểm khác biệt chính giữa C và C++

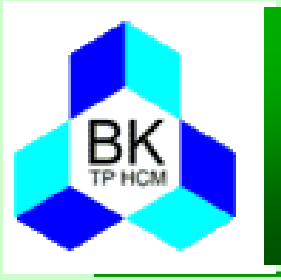
Phép gán ngầm từ con trỏ *void**

Ta không thể gán ngầm một con trỏ thuộc kiểu ***void**** trong C++ cho bất kỳ biến thuộc kiểu nào khác.

Chẳng hạn, thí dụ sau hoàn toàn hợp lệ trong C:

int *x = malloc(sizeof(int) * 10);

nhưng sẽ không hợp lệ trong C++



Một số điểm khác biệt chính giữa C và C++

Xin và giải phóng biến động

Trong C, chỉ có một hàm chính xin bộ nhớ động là ***malloc*** :

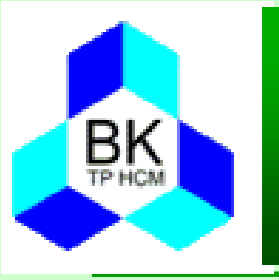
```
int *x = malloc( sizeof(int) );
int *x_array = malloc( sizeof(int) * 10 );
```

và ta luôn giải phóng bộ nhớ theo cách như sau:

```
free( x );
free( x_array );
```

Trong C++, ta sử dụng toán tử ***new[]*** để định vị và dùng toán tử ***delete[]*** để xóa nó.

```
int *x = new int;
int *x_array = new int[10];
delete x;
delete[] x_array;
```



Một số điểm khác biệt chính giữa C và C++

Xin và giải phóng biến động

Việc sử dụng hàm ***malloc***, cũng như ***calloc***, trong C++ là hoàn toàn có thể, tuy nhiên chúng không được khuyến khích do đặc tính không sử dụng khái niệm đối tượng của chúng.

Một số điểm khác biệt chính giữa C và C++

Khai báo hàm trước khi sử dụng

Với C++, chúng ta phải khai báo hàm ở dạng prototype trước khi sử dụng.

Với C, việc khai báo prototype không phải là bắt buộc, dù được khuyến khích sử dụng để tránh xảy ra những lỗi luận lý đáng tiếc.

Một số điểm khác biệt chính giữa C và C++

Khai báo hàm trước khi sử dụng

Thí dụ 15.2: Với ngôn ngữ C:

```
#include <stdio.h>
  int main()
  {foo();
  return 0;}
 int foo()
{printf( "Hello world" ); }
```

Thí dụ 15.3: Với C++, ta phải viết:

```
#include <stdio.h>
  int foo (void);
  int main()
  {        foo();        return 0;}
 int foo(void)    {        printf( "Hello world" );    }
```

Một số điểm khác biệt chính giữa C và C++

Struct và Enum

Khi khai báo kiểu ***struct***, cả C và C++ đều sử dụng từ khoá ***struct***, khi khai báo biến thì C++ không cần dùng lại từ khóa struct như C.

Thí dụ với C++:

```
struct a_struct
{
int x;
};
a_struct struct_instance;
```

Còn trong C, chúng ta phải sử dụng lại từ khoá struct để khai báo biến như sau:

```
struct a_struct struct_instance;
```

Một số điểm khác biệt chính giữa C và C++

Struct và Enum

Tương tự như vậy với kiểu **enum**, trong C chúng ta phải bao gồm từ khóa **enum** khi khai báo kiểu và biến; còn trong C++, chúng ta chỉ dung từ khóa **enum** cho khai báo kiểu, còn khi khai báo biến thì không cần.

Một số điểm khác biệt chính giữa C và C++

C++ có một thư viện lớn hơn C

Thư viện của C++ lớn hơn của C nhiều, nhiều thứ khi viết với C ta không liên kết được hay liên kết rất khó khăn với thư viện, thì với C++ ta có thể liên kết chúng với thư viện này rất dể dàng.

Một số chương trình ví dụ

Xem sách giáo khoa.

KẾT THÚC CHƯƠNG 14