Kỹ Thuật Lập Trình

(Ngôn Ngữ Lập Trình C)

Bộ tiền xử lý

Bộ Tiền Xử Lý

- Các chỉ thị tiền xử lý xuất hiện trước khi chương trình được biên dịch.
- Các thao tác có thể thực hiện
  - Gộp thêm các tệp khác vào tệp đang biên dịch
  - Định nghĩa các hằng, macro
  - Biên dịch có điều kiện mã chương trình
  - Thực hiện có điều kiện các chỉ thị tiền dịch
- Các chỉ thị tiền dịch đều bắt đầu bằng ký tự #
  - Chỉ có các ký tự dấu cách được phép đứng trước các chỉ thị tiền xử lý trong dòng đó

Bộ Tiền Xử Lý

- Các chỉ thị tiền xử lý chỉ có ý nghĩa trong phạm vi file mà nó được định nghĩa.
- Chỉ thị tiền xử lý không phải là cú pháp cơ bản của C.
- Tuy nhiên việc sử dụng chúng có thể làm thay đổi cấu trúc của chương trình.

Chỉ Thị **#include**

- Chỉ thị tiền dịch **#include** thường được sử dụng với các file tiêu đề

  #include <standard.h>

  #include "myheader.h"

- Chỉ thị này thực hiện việc sao chép file vào vị trí của chỉ thị.

Định Nghĩa Các Hằng

- Sử dụng từ khóa **#define** để định nghĩa các hằng.
- Thường được dùng để loại bỏ “magic numbers \ hard code” trong mã nguồn.
- Dạng thường gặp

  #define name text

- Trước khi chương trình được biên dịch, tất cả **name** sẽ được thay tự động bằng **text**

Định Nghĩa Các Hằng

```
#define BUFFERSIZE  256
#define MIN_VALUE   -32
#define PI          3.14159
```

- Chú ý
  - Không có dấu **=** và dấu**;**
  - Tên được định nghĩa bởi **#define** có thể được loại bỏ bằng sử lệnh **#undef** (có thể được định nghĩa lại sau đó)
- Trong thực tế, các biểu tượng hằng thường dùng chữ IN HOA để phân biệt nó với tên biến và tên hàm.

#define, const Và enum

- Có thể thay thế **#define**

```
#define ARRAYSIZE     10
const int ArraySize = 10;
double array[ARRAYSIZE]; /* Valid. */

#define PI            3.14159
#define ARRAYSIZE     10
const double Pi = 3.14159; /* Preferred */
enum {ARRAYSIZE = 10};     /* Preferred */
```

Enum example

```
// example program to demonstrate working of enum in C

#include<stdio.h>

enum week{Mon, Tue, Wed, Thur, Fri, Sat,
Sun};

int main()
{
    enum week day;
    day = Wed;
    printf("%d",day);
    return 0;
}
```

Enum example

```
// Another example program to demonstrate working
#include<stdio.h>

enum year{Jan, Feb, Mar, Apr, May, Jun, Jul,
          Aug, Sep, Oct, Nov, Dec};

int main()
{
    int i;
    for (i=Jan; i<=Dec; i++)
        printf("%d ", i);

    return 0;
}
```

Định Nghĩa Các Macro

- Lệnh **#define** thường được sử dụng trong việc tạo ra các macro

```
#define  MAX(x,y)  ((x)>(y) ? (x) : (y))
```

- Macro giống như hàm, nhưng thực chất không phải là hàm. Thực tế tên macro (MAX) sẽ được thay thế bằng dòng lệnh tương ứng với các đối số trước khi chương trình được biên dịch

```
int a = 4, b = -7, c;
c = MAX(a,b);
```

thay bằng:

```
c = ((a)>(b) ? (a) : (b));
```

Tại Sao Dùng Macro

- Tốc độ
  - Thực hiện như hàm nhưng sử dụng lời gọi hàm, đoạn mã được chèn vào trong chương trình trước khi biên dịch
  - Vấn đề này ít có ý nghĩa với chương trình viết trên PC
- Mã chung
  - Macro cho phép làm việc với mọi loại kiểu (**int**, **double**…)

```
int max(int x, int y) {
return x > y ? x : y;
}
```

Ví Dụ Về Macro

```
#define SQR(x)          ((x)*(x))
#define SGN(x)          (((x)<0) ? -1 : 1)
#define ABS(x)          (((x)<0) ? -(x) : (x))
#define ISDIGIT(x)      ((x) >= '0' && (x) <= '9')
#define NELEMS(array) sizeof(array)/sizeof(array[0]))

#define CLAMP(val,low,high) \
((val)<(low) ? (low) : (val) > (high) ? (high) : (val))

#define ROUND(val) \
((val)>0 ? (int)((val)+0.5) : -(int)(0.5-(val)))
```

Một Vài Nhược Điểm

- Dễ dàng mắc lỗi với macro đơn giản.
- Ba nhược điểm chính
  - Sử dụng dấu ngoặc không chính xác
  - Dùng các toán tử ++, --
  - Không kiểm tra kiểu

Cạm Bẫy Macro

```
#define SQR(x)  x * x
```

- Ví dụ

```
int a = 7;
b = SQR(a+1);
```

dẫn tới

```
b = a+1 * a+1;
```

- Giải pháp: dùng dấu ngoặc khi có thể

Cạm Bẫy Macro

- Ví dụ

  b = ABS(a++);

  trở thành

  b = (((a++)<0) ? -(a++) : (a++));

- Giải pháp: không sử dụng các toán tử này trong macro

Cạm Bẫy Macro

- Không kiểm tra kiểu là con dao hai lưỡi, có thể dẫn tới các lỗi tính toán

```
int a = 7, b;

double c = 5.3, d;

d = SQR(a);

b = SQR(c);
```

Macro Chiếm Nhiều Dòng

- Bạn có thể viết các maco nhiều dòng với cuối dòng kết thúc bằng `\`

```
#define ERROR(condition, message) \
    if (condition) printf(message)
```

Ví Dụ

```
#define TIMELOOP(CODE) { \
  t0 = clock(); \
  for (i = 0; i < n; ++i) { CODE; } \
  printf("%7d ", clock() - t0); \
}
```

Sử dụng như sau

```
TIMELOOP(y = sin(x));
```

Hằng Xâu Ký Tự Trong Macro

- Nếu đối số của macro có ký tự # đứng trước, thì đối số đấy sẽ được chuyển thành xâu ký tự hằng.

```
#define PRINT_DEBUG(expr) \
        printf(#expr " = %g\n", expr)
```

Ví dụ:

```
PRINT_DEBUG(x/y);
printf("x/y" " = %g\n", x/y);
```

Macro Có Sẵn

- Vài macro có sẵn trong chương trình dịch

```
__LINE__
__FILE__
__DATE__
__TIME__
__STDC__
```

Ví Dụ

```
#define PRINT_DEBUG(expr, type) \
  printf(__FILE__ "[%d](" #expr "): \
        %" type##Conv "\n", __LINE__, (expr))
```

- Đoạn chương trình sau

```
#define intConv          “d”
#define doubleConv       “f”
PRINT_DEBUG(x/y, int);
```

Sẽ in ra tên file, thứ tự dòng và kết quả

Dịch Có Điều Kiện

- Các chỉ thị dịch có điều kiện trong C

  `#if, #elif, #else, #endif`

  `#ifdef, #ifndef`

- Mục đích
  - Thêm vào các đoạn mã gỡ lỗi chương trình
  - Thêm vào các đoạn mã không phãi mã chuẩn
  - Tránh việc chèn các file header nhiều lần

Gỡ Lỗi

- Khi dịch chương trình trong chế độ debug

```
//#define DEBUG
```

- Bạn có thể chèn vào các đoạn mã gỡ lỗi

```
#ifdef DEBUG
  printf("Pointer %#x points
          to value %f", pd, *pd);
#endif
```

Đoạn Mã Không Chuẩn

- Sử dụng trong trường hợp đoạn mã chỉ dùng cho các vi xử lý khác nhau

```
#ifdef __WIN32__
  return WaitForSingleObject(Handle,0)==
                                WAIT_OBJECT_0;
#elif defined(__QNX__)||defined(__linux__)
  if(flock(fd,LOCK_EX|LOCK_NB) == -1)
     return 0;
  else
     return 1;
#endif
```

Chống Chèn File Nhiều Lần

- File header chỉ nên được chèn vào đúng một lần trong chương trình (mặc dù được nhiều file sử dụng).
- Chèn nhiều lần sẽ làm cho các biến, các hàm, các nhãn được định nghĩa lại và chương trình sẽ không thể dịch thành công
- Phương pháp chống: sử dụng "*header guards*"

```
#ifndef A_HEADER_H_
#define A_HEADER_H_

/* Contents of header file is here. */

#endif
```