IL PREPROCESSORE
lunedì 26 dicembre 2022 16:31

IL PREPROCESSORE

È UN PROGRAMMA che esegue una TRASFORMAZIONE da codice SORGENTE a codice SORGENTE.

- Le sue funzionalità fondamentali sono:
  - inclusione di file di intestazione
  - espansione di macro
  - compilazione condizionale
  - controllo di riga

INCLUSIONE DI FILE.

- Uno degli usi più comuni del preprocessore è includere un altro file.
- Il preprocessore sostituisce, ad esempio, la riga

  ```
  #include <stdio.h>
  ```

  con il contenuto testuale del file 'stdio.h'.
- Se il nome del file è racchiuso tra parentesi angolari, il file viene cercato nei percorsi standard di inclusione del compilatore
- Se il nome del file è racchiuso tra virgolette doppie, il percorso di ricerca viene espanso per includere la directory del file sorgente corrente

COMPILAZIONE CONDIZIONALE

- Le direttive `#if`, `#ifdef`, `#ifndef`, `#else`, `#elif` e `#endif` possono essere utilizzate per la compilazione condizionale.

Chiedo di eliminare alcune porzioni del mio file sorgente, posso dare delle espressioni condizionali e vedere se sono vere o false.

A seconda della direttiva che utilizzo, il mio preprocessore eliminerà una parte del mio codice!
Una porzione di file deve essere inclusa nel sorgente che stiamo andando a utilizzare!

↓

AL VERIFICARSI DI UNA DETERMINATA CONDIZIONE!

Posso dichiarare una MACRO O VARIABILE di PREPROCESSORE, chiamata "VERBOSE":

```
#if VERBOSE >= 2
  printf("trace message");
#endif
```

↑

IN ALTO NEL .C

```
#ifdef __unix__
# include <unistd.h>
#elif defined _WIN32
# include <windows.h>
#endif
```

SE STO COMPILANDO SU UNIX

E STO COMPILANDO SU WINDOWS

SE VERBOSE È ≥2 allora compila tutto quello che c'è fino a #ENDIF!
Se stiamo compilando su un sistema UNIX, i compilatori definiscono una MACRO __UNIX__.
- WIN32 viene definita automaticamente per sistemi Windows.

GUARDIE DI INCLUSIONE

file.h
#include file2.h
STRUCT PIPPO
{

};

file.c
#include "file.h"

file2.h
#include "file.h"

→ hai definito due volte la STRUCT PIPPO;

NE HO DUE, IL PREPROCESSORE LI SOSTITUISCE.

COME RISOLVO? Se non è definita qualche MACRO del mio processore C, allora definiscila:

- Se si include uno stesso file header più volte, si possono generare degli errori
- È utile inserire delle *guardie di inclusione*, che impediscono la ridefinizinoe di simboli e tipi

file.h

```
#ifndef _FILENAME_H
#define _FILENAME_H
...
#endif
```

→

```
#pragma once
```

DEFINIZIONE di MACRO

Posso definire una MACRO di TIPO oggetto scrivendo: `#define <identifier> <replacement token list>`

↓

INSIEME di caratteri che VERRANNO SOSTITUITI dal mio PREPROCESSORE ogni volta che questo identificatore VIENE TROVATO all'INTERNO del mio codice.

Esempio:

```
#define PI 3.14159
```

Posso anche utilizzare le macro per definire la mia MACROFUNZIONE: `#define <identifier>(<parameter list>) <replacement token list>`

funzionano come le MACROoggetto con l'aggiunta che possono accettare dei parametri.

Esempio:

```
• #define RADTODEG(x) ((x) * 57.29578)
```

↓

Parametro. ogni volta che nel codice chiamo RADTODEG(NUMERO), verrà effettuata quell'operazione!

Da questo punto in poi non effettuare più la sostituzione: `#undef <identifier>`

Un esempio

confronto ternario!

- `#define MAX(a,b) ((a) > (b) ? (a) : (b))`
- `#define MIN(a,b) ((a) < (b) ? (a) : (b))`

- Cosa succede se scrivo del codice di questo tipo?

```
int x = 20, y = 10;
int max = MAX(x++, y);
```

CORRETTO!

x++ > y ? x++ : y

+1 → +2

MAX = 22 MA È UN RISULTATO SBAGLIATO.

- *Double evaluation side effect:*
  - `int max = ((x++) > (y) ? (x++) : (y));`

ATTENZIONE: IL PREPROCESSORE SI LIMITA SOLO A SOSTITUIRE CARATTERI.

COME RISOLVO QUESTA COSA?

STATEMENT EXPRESSION

```
#define max(a,b)                \
({                              \
    __typeof__ (a) _a = (a);    \
    __typeof__ (b) _b = (b);    \
    _a > _b ? _a : _b;          \
})

#define min(a,b)                \
({                              \
    __typeof__ (a) _a = (a);    \
    __typeof__ (b) _b = (b);    \
    _a < _b ? _a : _b;          \
})
```

CARATTERE DI ESCAPE ↓ A CAPO È IMPLEMENTATO CON \n.

Dico al compilatore di interpretare tutto come uno statement.

__typeof__ viene valutato a tempo di compilazione e assume il tipo della variabile che passo come argomento

x++ viene eseguito una volta sola!

Sperimentate sul codice!

ERRORI A TEMPO DI COMPILAZIONE

- Sono presenti due direttive speciali:
  - `#warning Message`: stampa "Message" a schermo durante la compilazione, come warning
  - `#error Message`: stampa "Message" a schermo e fa fallire la compilazione

UTILE PER CODICE MULTIPIATTAFORMA

- È un supporto utile se unito, ad esempio, alla compilazione

- È un supporto utile se unito, ad esempio, alla compilazione condizionale:

```
#ifdef __unix__
# include <unistd.h>
#elif defined _WIN32
# error Unsupported operating system
#endif
```

MACRO VARIADICHE

- È possibile definire una macro variadica:
  - `#define macro(M, ...)` → Tutto quello che il PREPROCESSORE SOSTITUIRÀ è quello che scrivo qui.
  - L'elenco dei parametri può essere recuperato utilizzando la direttiva di preprocessore `##__VA_ARGS__`

- Ad esempio:

```
#define print(fmt, ...) printf(fmt, ##__VA_ARGS__)
```

VIENE SOSTITUITA PRINT CON PRINTF A TEMPO di COMPILAZIONE.

IO SCRIVERÒ PRINT NEL CODICE ;

- Sono presenti alcune macro speciali, definite a tempo di compilazione automaticamente:
  - `__FILE__`: Il nome del file in cui viene utilizzata la macro
  - `__LINE__`: La riga del file in cui viene utilizzata la macro
  - `__func__`: Il nome della funzione in cui viene utilizzata la macro
  - `__STDC_VERSION__`: La versione del C utilizzata dal compilatore
  - `__DATE__`: La data di compilazione
  - `__TIME__`: L'ora di compilazione

APPROFONDISCI !