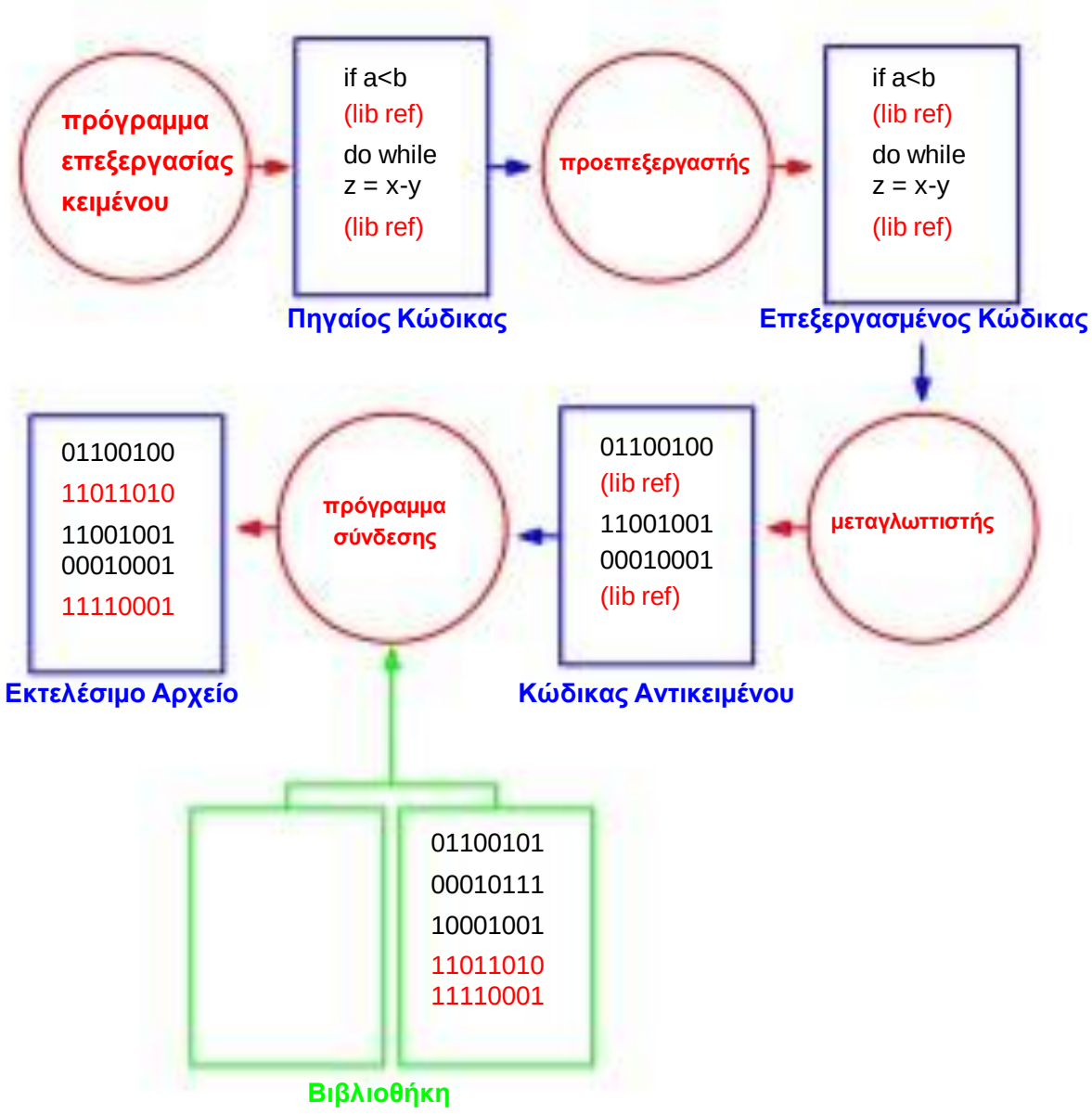
Προγραμματισμός ΙΙ
Προεπεξεργαστής

Κωνσταντίνος Τσερπές
(βασισμένο στις διαφάνειες του κ. Δημήτρη Μιχαήλ)

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Διαδικασία Μεταγλώττισης



Ο προεπεξεργαστής

Ο προεπεξεργαστής αλλάζει το πρόγραμμα μας, ανάλογα με τις εντολές που αρχίζουν με #.

Η έξοδος του είναι ενα άλλο πρόγραμμα γλώσσας C.

Το πρόγραμμα αυτό δεν το βλέπουμε επειδή δίνεται αυτόματα στον μεταγλωττιστή.

Η εντολή `#include` λέει στον προεπεξεργαστή να αντικαταστήσει την εντολή με τα περιεχόμενα του αρχείου που ακολουθεί.

```
1  #include  <stdio.h>
2
3  int  main ( )
4  {
5        printf ( "Hello world!\n" ) ;
6  }
```

Ο προεπεξεργαστής βρίσκει το αρχείο **stdio.h** που βρίσκεται σε κάποια προκαθορισμένη θέση στο σύστημα αρχείων και αντικαθιστά την γραμμή 1 με το περιεχόμενο του αρχείου αυτού.

Το πρόγραμμα που τελικά φτάνει στον μεταγλωττιστή είναι:

```
1  #ifndef  _STDIO_H
2
3  #if  !defined __need_FILE &&  !defined __need___FILE
4  # define  _STDIO_H  1
5  # include  <features.h>
6
7  /* a lot  of code here*/
8
9
10 #endif       /*    <stdio.h> included.                */
11
12 #endif       /*    ! _STDIO_H    */
13
14 int  main ( )
15 {
16         printf ( "Hello  world!\n" ) ;
   }
```

Εκτός από την προηγούμενη μορφή, μπορούμε να κάνουμε:

```
1  #include  "myheader.h"
2
3  int  main ( )
4  {
5        printf ( "Hello world!\n" ) ;
6  }
```

Σε αυτή την μορφή ο προεπεξεργαστής προσθέτει στους φακέλους που ψάχνει το αρχείο **myheader.h** και τον τρέχον φάκελο (τον φάκελο που βρίσκεται το πρόγραμμα μας). Αφού βρεθεί το αρχείο **myheader.h** το περιεχόμενα του αντικαθιστούν την εντολή **include**.

Ο προεπεξεργαστής μας επιτρέπει να κάνουμε αντικαταστάσεις λέξεων στα προγράμματα μας.

Με την εντολή

```
#define  PRINTFUNCTION  printf
```

λέμε για παράδειγμα στον προεπεξεργαστή να αντικαταστήσει το αναγνωριστικό PRINTFUNCTION με την λέξη printf όπου αυτή εμφανίζεται στον κώδικα.

```
1  #include  <stdio.h>
2
3  #define  PRINTFUNCTION  printf
4
5  int  main ( )
6  {
7        PRINTFUNCTION ( "Hello world\n" ) ;
8  }
```

Το παραπάνω πρόγραμμα αντικαθίσταται με το

```
1  #include  <stdio.h>
2
3  int  main ( )
4  {
5        printf ( "Hello world\n" ) ;
6  }
```

πριν δωθεί στον μεταγλωττιστή για μεταγλώττιση.

Η πιο συνηθισμένη χρήση είναι η απόδοση συμβολικών ονομάτων σε αριθμητικές σταθερές:

- #define BUFFER_SIZE 1024
- #define PI 3.1459

Ο προεπεξεργαστής θα αναγνωρίσει και θα αντικαταστήσει το συμβολικό όνομα με την δοσμένη τιμή. Η εντολή:

```
foo  =  ( char*) malloc ( BUFFER_SIZE ) ;
```

θα γίνει

```
foo  =  ( char*) malloc (1024);
```

μετά την προεπεξεργασία.

Κατά σύμβαση τα ονόματα μακροεντολών είναι με κεφαλαία για να μπορούμε με μια ματιά να ξεχωρίσουμε τις μακροεντολές από τις μεταβλητές.

Πολλαπλές Γραμμές

Στην πιο απλή μορφή μια εντολή στον προεπεξεργαστή τελειώνει με το τέλος της γραμμής.

Άμα θέλουμε να συνεχίζει μπορούμε να χρησιμοποιήσουμε ενα backslash.

```
1  #define NUMBERS  1,    \
2                   2,    \
3                   3
4
5  int  x [ ]     =   { NUMBERS };
```

Το παραπάνω κομμάτι κώδικα είναι ισοδύναμο με

```
1  int  x [ ]     =   {1,    2,    3};
```

Συνήθη Λάθη

Χρήση του =

```
1  #define VALUE  =  100
2
3  if  (i  ==  VALUE )
4  {
5        /* some code here */
6  }
```

οδηγεί στο πρόγραμμα

```
1  if  ( i  == = 100 )
2  {
3        /* some code here  */
4  }
```

Το οποίο δεν περνάει από τον μεταγλωττιστή.

Συνήθη Λάθη

Χρήση του ;

```
1  #define VALUE  =  100;
2
3  if  (i  ==  VALUE )
4  {
5        /* some code here */
6  }
```

οδηγεί στο πρόγραμμα

```
1  if  ( i  == 100; )
2  {
3        /* some code here  */
4  }
```

Το οποίο δεν περνάει από τον μεταγλωττιστή.

Αντικατάσταση μακροεντολών

Ο προεπεξεργαστής επεξεργάζεται το πρόγραμμά σας και αντικαθιστά τα συμβολικά ονόματα σειριακά, μετά το σημείο που εμφανίζεται η #define.

π.χ

```
foo  =  X ;
#define  X  100
bar  =  X ;
```

⇒

```
foo  =  X ;
bar  =  1 0 0 ;
```

Αντικατάσταση μακροεντολών

Αντικατάσταση της αντικατάστασης

Η αντικατάσταση ερευνάται και αυτή για αντικατάσταση.

π.χ

```
#define  TABLESIZE  BUFSIZE
#define  BUFSIZE  1024

int  buffer [ TABLESIZE ] ;
```

⇒

```
int  buffer [ 1 0 2 4 ] ;
```

Σε όλον τον κώδικα: αρχικά το TABLESIZE θα αντικατασταθεί με το BUFSIZE και κατόπιν το BUFSIZE με το 1024. Προσέξτε ότι το BUFSIZE δεν ορίζεται, οταν χρησιμοποιείται, απλά αντικαθίσταται με το αλφαριθμητικό.

Είναι δυνατό να ορίσουμε μακροεντολές που μοιάζουν με κλήση συνάρτησης:

```
1
2
3
4
5
6
7
8
```

```c
#include  <stdio.h>

#define hello( ) printf( "Hello world\n" ) ;

int  main ( )
{
        hello ( ) ;
}
```

Προσοχή οι παρενθέσεις είναι υποχρεωτικές και δεν πρέπει να υπάρχει κενό μεταξύ του ονόματος και των παρενθέσεων.

Οι μακροεντολές μπορούν να πάρουν και ορίσματα:

- Τα ορίσματα πρέπει να είναι identifiers της C και να χωρίζονται από κόμματα και προαιρετικά κενά.
- Όλος ο ορισμός πρέπει να είναι σε μια λογική γραμμή.

π.χ η μακροεντολή

```
#define  min ( x , y )        ((x)<(y)?(x):(y))
```

αλλάζει όπως παρακάτω:

- **x** = **min**(**a**,**b**); ⇒ **x** = ((**a**)<(**b**)?(**a**):(**b**));
- **y** = **min**(1,2); ⇒ **y** = ((1)<(2)?(1):(2));
- **z** = **min**(**a**+28,*p); ⇒ **y** = ((**a**+28)<(*p)?(**a**+28):(*p));

Λάθη στα Ορίσματα

- Προσοχή γιατί στις μακροεντολές δεν γίνεται συντακτικός έλεγχος.
- Προσοχή με τις προτεραιότητες μετά την αντικατάσταση.

```
#define  assign(a, b) a = ( char ) b
```

Έτσι το

```
assign ( x ,  y> >8)
```

γίνεται

```
x = (char)y>>8;
```

Λάθη στα Ορίσματα

Προτεραιότητα

Προσοχή γιατί η παρακάτω μακροεντολή

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

γίνεται

- **RADTODEG**(**a**+**b**) ⇒ (**a**+**b***57.29578)

που δεν είναι σωστή.

Λάθη στα Ορίσματα

Προτεραιότητα

Προσοχή γιατί η παρακάτω μακροεντολή

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

γίνεται

- 1 / **RADTODEG**(**a**) ⇒ 1 / (**a**) * 57.29578

που δεν είναι σωστή.

Αντικατάσταση Ορισμάτων

Όλα τα ορίσματα σε μια μακροεντολή αντικαθίστανται πριν γίνει η τελική αντικατάσταση:

- αφού αντικατασταθούν τα ορίσματα, όλο το κείμενο της μακροεντολής σκανάρεται και για άλλες αντικαταστάσεις (και τα ορίσματα!)
- μπορεί να φαίνεται περίεργο, αλλά είναι χρήσιμο για να συμπεριφέρονται οι μακροεντολές όπως οι κλήσεις συνάρτησης

π.χ έστω η μακροεντολή

```
#define  min ( x , y )      ((x)<(y)?(x):(y))
```

Ο κώδικας **min(min(a,b),c)** γίνεται

- **min**(((**a**)<(**b**)?(**a**):(**b**)), **c**) και μετά
- ((((**a**)<(**b**)?(**a**):(**b**)))<(**c**)?(((**a**)<(**b**)?(**a**):(**b**))):(**c**))

Παράλειψη Ορισμάτων

Σε μια μακροεντολή μπορούμε να αφήσουμε ένα η περισσότερα ορίσματα κενά.

- ο προεπεξεργαστής χτυπάει άμα δεν δώσουμε σωστό αριθμό ορισμάτων
- εαν έχω 2 ορίσματα πρέπει να έχω ακριβώς ένα κόμμα μέσα στις παρενθέσεις

Προσοχή στις αντικαταστάσεις:

1. **min**(,**b**) ⇒ (( )<(**b**)?( ):(**b**))
2. **min**(,) ⇒ (( )<( )?():())
3. **min** ((,),) ⇒ (((,))<( )?((,)):( ))

4. **min**() ⇒ λάθος
5. **min**(,,) ⇒ λάθος

Αποφυγή Αντικατάστασης

Για να αποφύγουμε την αντικατάσταση μπορούμε να γράψουμε μια παράμετρο μέσα σε διπλά εισαγωγικά "".

π.χ

```
#define  foo(x)     x ,"x"
```

Η χρήση της **foo**(**bar**) έχει ως αποτέλεσμα:

```
bar , "x"
```

και όχι

```
bar , "bar"
```

Προκαθορισμένες Συμβολικές Σταθερές

- **__FILE__**
  - Αντικαθίσταται με το όνομα και το path του τρέχοντος αρχείου που επεξεργαζόμαστε
  - Είναι σταθερά αλφαριθμητικού
  - π.χ "/usr/include/stdio.h"
- **__LINE__**
  - Αντικαθίσταται με τον αριθμό γραμμής του τρέχοντος αρχείου

Πολύ χρήσιμες σταθερές για μηνύματα λάθους,

π.χ

```
fprintf ( stderr ,  "Error at file %s, line number %d\n" , __FILE__ , __LINE__);
```

Προκαθορισμένες Συμβολικές Σταθερές

- **__DATE__**
  - Αντικαθίσταται με την τρέχουσα ημερομηνία
  - "Feb 2 1996"
- **__TIME__**
  - Αντικαθίσταται με την τρέχουσα ώρα
  - "23:59:59"

Τελεστής

Για να δημιουργήσουμε strings από παραμέτρους μακροεντολής μπορούμε να γράψουμε:

```
#define QUOTEME( x )  #x
```

ο κώδικας

```
printf ( "%s\n" ,  QUOTEME ( 1 + 2 ) ) ;
```

γίνεται

```
printf ( "%s\n" ,  "1+2"            );
```

Τελεστής

Για να βάλουμε εισαγωγικά στην τιμή μιας άλλης μακροεντολής μπορούμε να κάνουμε:

```
#define _QUOTEME( x )  #x
#define QUOTEME( x )  _QUOTEME( x )
```

Η τεχνική αυτή βολεύει για παράδειγμα όταν θέλουμε να βάλουμε εισαγωγικά στην μακροεντολή **__LINE__**.

Στην γραμμή 34 του κώδικα

1. **QUOTEME**(**__LINE__**) ⇒ "34"
2. **_QUOTEME**(**__LINE__**) ⇒ "__LINE__"

Τελεστής

Συνένωση Συμβόλων

Πολλές φορές υπάρχει ανάγκη να ενώσει κανείς 2 σύμβολα (π.χ αλφαριθμητικά) όταν αντικαθίσταται μια μακροεντολή.

Μπορούμε να γράψουμε:

```
#define concat(x,y)  x ## y
```

Τα δύο σύμβολα αριστερά και δεξιά του τελεστή θα ενωθούν σε ένα. Τα σύμβολα αυτά συνήθως είναι:

- identifiers(π.χ μεταβλητές)
- identifiers και αριθμοί (π.χ foo3)
- αριθμοί (π.χ 1 και 3 σε 13)

Τελεστής

Παράδειγμα Συνένωσης Συμβόλων

```
struct command
{
        char * name ;
        void (*function)( void ) ;
};

struct command commands [ ]        =
{
        { "quit" , quit_command },
        { "help" , help_command   },

};
```

⇒

```
#define COMMAND(NAME)    \
    { #NAME , NAME ## _command }

struct command commands [ ]        =
{
        COMMAND ( quit ) ,
        COMMAND ( help ) ,

};
```

Μεταγλώττιση υπό Συνθήκες

Ο προεπεξεργαστής μας επιτρέπει να μεταγλωττίσουμε επιλεκτικά κάποια κομμάτια κώδικα με την χρήση των εντολών #ifdef και #ifndef.

```
1  int  foo ( )
2  {
3  #ifdef DEBUG
4          fprintf ( stderr ,  "foo()□called..\n" ) ;
5  #endif
6
7          /*  code here       */
8  }
```

Η εντολή **fprintf**() μεταγλωττίζεται μόνο εαν έχουμε κάνει **define** την σταθερά **DEBUG**.

Μεταγλώττιση υπό Συνθήκες

Με την χρήση της #ifdef μπορούμε να κάνουμε επιλεκτική μεταγλώττιση. Μας βοηθάει πολύ όταν θέλουμε ο κώδικας μας να λειτουργεί σε διάφορα λειτουργικά που έχουν διαφορετικές συναρτήσεις.

```
1  int  foo ( )
2  {
3  #ifdef SYSTEM_IS_WINDOWS
4          /*  windows specific code            */
5  #elif   SYSTEM_IS_LINUX
6          /*  linux  specific code            */
7  #else
8          /*  code for  other cases            */
9  #endif
10 }
```

Αρχεία Επικεφαλίδας

Η τεχνική αυτή μας βοηθάει ώστε να μην κάνουμε #include το ίδιο αρχείο πολλές φορές. Παρακάτω φαίνεται μια δυνητική μορφή του **stdio.h**.

```
1  #ifndef  STDIO_H
2  #define  STDIO_H
3
4  /*  contents  of STDIO_H        */
5
6  #endif
```

Την δεύτερη φορά που θα γίνει #include το αρχείο αυτό, θα έχει γίνει **define** η σταθερά **STDIO_H** και άρα ο κώδικας δεν θα συμπεριληφθεί.

Το παράδειγμα των assertions

Η βιβλιοθήκη της C μας παρέχει το αρχείο επικεφαλίδας **assert.h** το οποίο παρέχει την μακροεντολή **assert**().

- η μακροεντολή αυτή παίρνει ως παράμετρο μια συνθήκη
- εαν η συνθήκη είναι αληθής δεν κάνει τίποτα
- εαν η συνθήκη είναι ψευδής τότε σταματάει την εκτέλεση του προγράμματος και μας ενημερώνει για το σημείο και την συνθήκη

Η μακροεντολή **assert**() βολεύει πολύ όταν κάνουμε debugging.

Το παράδειγμα των assertions

```
1  #include <stdio.h>
2  #include <assert.h>
3
4  double divide ( int a , int b )
5  {
6        assert ( b != 0 );
7        return ((double)a)/b;
8  }
9
10 int main ( )
11 {
12       printf ( "3/2 = %lf\n" , divide ( 3 , 2));
13       printf ( "3/2 = %lf\n" , divide ( 3 , 0));
14 }
```

Το παραπάνω πρόγραμμα τυπώνει

```
3/2 = 1.500000
test: test.C:6: double divide(int, int): Assertion `b != 0' failed.
Aborted
```

Το παράδειγμα των assertions

Οι έμπειροι προγραμματιστές βάζουν assertions στον κώδικα τους σε σημεία που θέλουν να ελέγξουν πως κάποια συνθήκη είναι αλήθεια.

Όταν θέλουμε να δώσουμε όμως το κώδικα σε μορφή τελικού προϊόντος σε κάποιον πελάτη, ο κώδικας των assertions καθυστερεί την εκτέλεση.

Για αυτό εάν έχουμε δηλώσει την σταθερά **NDEBUG** πριν την εισαγωγή του **assert.h** στον κώδικα μας η μακροεντολή **assert**() ορίζεται ως:

```
#define assert(ignore)((void) 0)
```

Ο μεταγλωττιστής είναι έξυπνος αρκετά ώστε να αφαιρέσει αυτές τις περιττές εντολές από τον κώδικα.

Η εντολή #if επιτρέπει τον έλεγχο για μια τιμή, αντί για ύπαρξη μιας μακροεντολής.

```
#if expression
      controlled text
#endif
```

Η έκφραση expression μπορεί να περιέχει:

1. ακέραιους και χαρακτήρες
2. αριθμητικούς τελεστές για πράξεις (και κατά bit), συγκρίσεις, λογικούς τελεστές
3. μακροεντολές
4. την εντολή defined (θα την δούμε παρακάτω)
5. identifiers (αν δεν είναι μακροεντολές παίρνουν τιμή 0)

Η εντολή defined

Χρησιμοποιείται με τα #if και #elif για να ελέγξει αν ένα σύμβολο είναι ορισμένο ως μακροεντολή:

- **defined name** και **defined** (**name**) είναι εκφράσεις που παίρνουν την τιμή 1 αν το **name** είναι ορισμένο ως μακροεντολή μέχρι το τρέχον σημείο του προγράμματος, αλλιώς παίρνουν την τιμή 0
- η έκφραση
  - #if defined MACRO είναι ισοδύναμη με την
  - #ifdef MACRO

Παραδείγματα #if και defined

```
#if  defined  ( __vax__ )  ||  defined  (__ns16000__ )
      // code here
#endif
```

ή

```
#if  defined  BUFSIZE && BUFSIZE  >=  1024
      // code here
#endif
```

Το τελευταίο παράδειγμα είναι ισοδύναμο με

```
#if  BUFSIZE  >=  1024
      // code here
#endif
```

αφού στην περίπτωση που το **BUFSIZE** δεν έχει οριστεί θα γίνει 0.

Η μακροεντολή #error εισάγει ένα μήνυμα λάθους στην έξοδο του μεταγλωττιστή.

π.χ

```
#error  "NO!"
```

τυπώνει το μήνυμα στην έξοδο του μεταγλωττιστή και σταματάει την εκτέλεση του.

Πολύ χρήσιμη μακροεντολή για να δούμε εάν εκτελείται ένα κομμάτι κώδικα.

Παράδειγμα #error

```
#ifdef WINDOWS
        /* Windows specific code */
#elif  defined ( UNIX )
        /*  Unix  specific code */
#else
      #error  "What's your operating system?"
#endif
```

Η μακροεντολή #pragma επιτρέπει στον κατασκευαστή ενός μεταγλωττιστή να παραμετροποιεί τον μεταγλωττιστή του. Με αυτό τον τρόπο ο προγραμματιστής έχει μεγαλύτερο έλεγχο πάνω στον μεταγλωττιστή.

Για παράδειγμα η μακροεντολή αυτή χρησιμοποιείται συνήθως για να μην εμφανίζονται συγκεκριμένα μηνύματα λάθους του μεταγλωττιστή, κ.τ.λ

π.χ ο μεταγλωττιστής **gcc** παρέχει την #pragma message

```
#pragma  message  "Compiling "  __FILE__  "..."
```

που τυπώνει μηνύματα αλλά δεν σταματάει την μεταγλώττιση.