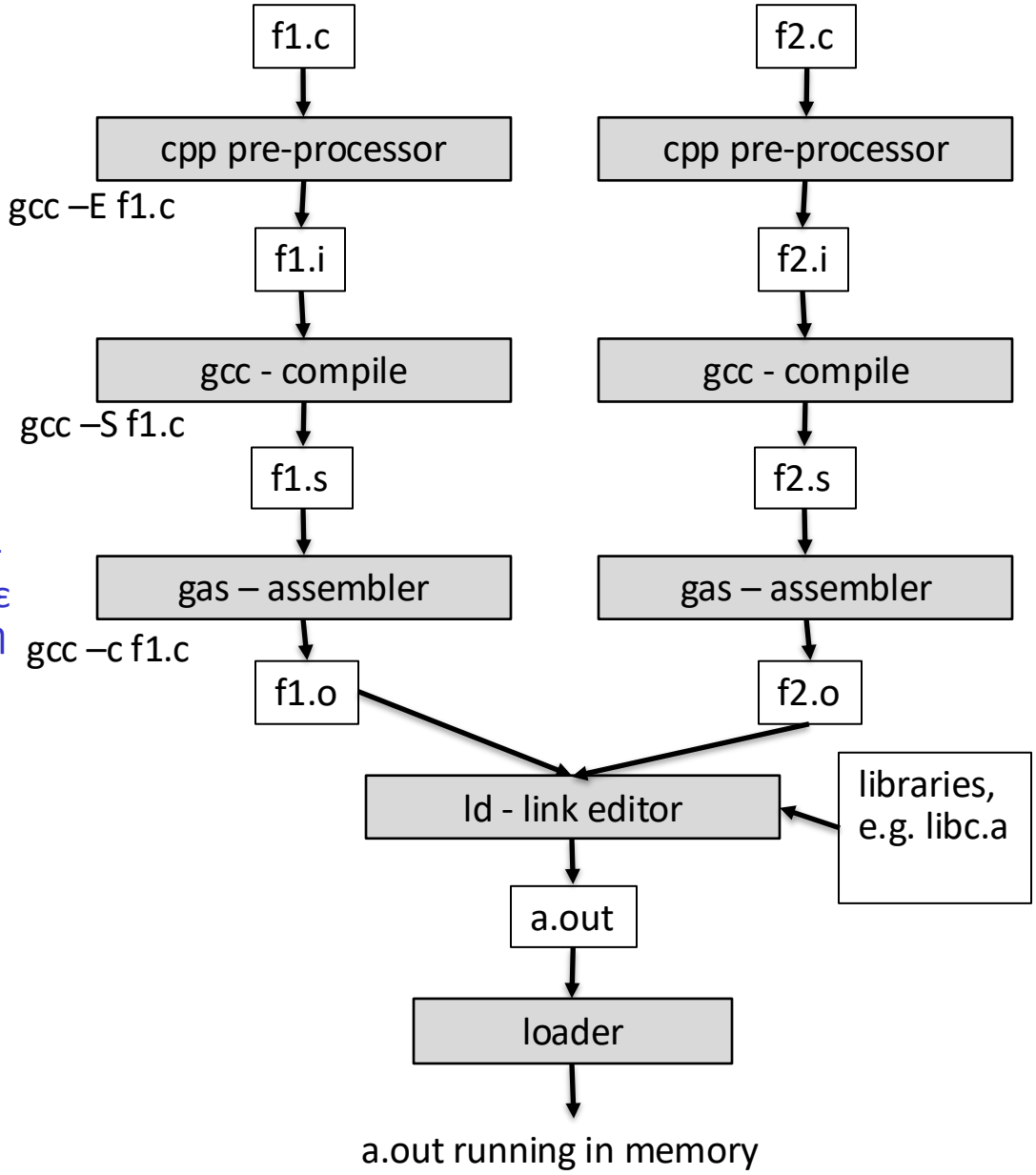
HY255 Εργαστήριο Λογισμικού - L05

(a) Program Building Process
(b) cpp – C Pre-processor

HY255 Εργαστήριο Λογισμικού
Άνοιξη 2021
Άγγελος Μπίλας

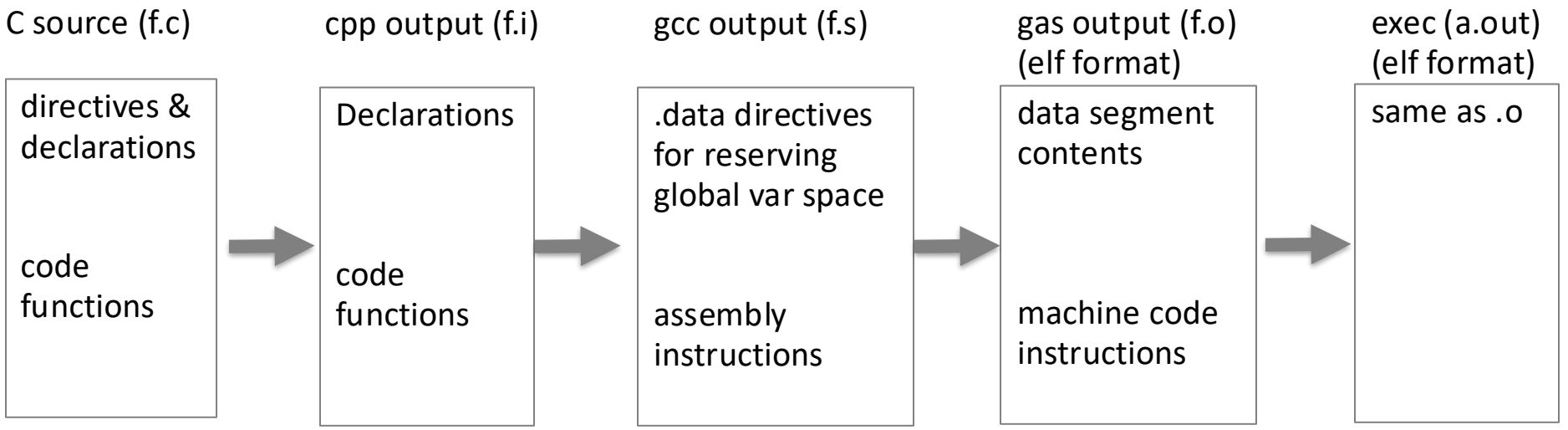
Program Building Process

- Για να σχηματίσουμε το executable (a.out) ενός προγράμματος (f1.c, f2.c) που γράψαμε, χρησιμοποιούμε
  - gcc –Wall –pedantic –ansi f1.c f2.c
- Τι συμβαίνει σε αυτή τη διαδικασία;
- Σημειώσεις
  - Όταν εμείς καλούμε τον gcc ουστιαστικά καλούμε ένα script που εκτελεί όλα τα βήματα με τα αντίστοιχα εργαλεία
  - Οι βιβλιοθήκες βρίσκονται σε προδιαγεγραμμένα directories που τα γνωρίζει ο linker (link editor)
  - Το a.out βρίσκεται στον δίσκο και ο loader είναι το πρόγραμμα που σε συνεργασία με το λειτουργικό σύστημα βάζει το a.out στη μνήμη και αρχίζει να το εκτελεί
- Σήμερα θα μιλήσουμε σε λεπτομέρεια για τον pre-processor cpp
- Για το linking/loading θα μιλήσουμε προς το τέλος του μαθήματος
- Για τον compiler θα κάνετε χωριστό μάθημα (compilers)
- Δοκιμάστε τα διάφορα στάδια με τις αντίστοιχες παραμέτρους



Δομή ενδιάμεσων προγραμμάτων

- Το πρόγραμμα διατηρεί την δομή του αρχικού προγράμματος C με διαφορετικό επίπεδο λεπτομέρειας



- Άραγε μπορούμε ξεκινώντας από το ”τέλος” (από το a.out) να πάμε προς τα ”πίσω“ και να παράγουμε τα source αρχεία (π.χ. το f.c);

Compiler - Assembler

- Η διαδικασία της μετάφρασης (.c -> .s) από source της γλώσσας σε assembly είναι το πιο πολύπλοκο μέρος της διαδικασίας – θα το δείτε στον μάθημα των compilers
- Ο assembler (.c -> .s) είναι ένα σχετικά απλό στάδιο που μεταφράζει τον κώδικα assembly σε κώδικα μηχανής, σχεδόν ένα-προς-ένα (μια εντολή assembly σε μια εντολή κώδικα μηχανής)
  - Ο κώδικας assembly είναι ουσιαστικά ο ίδιος με τον κώδικα μηχανής που εκτελείται, μόνο είναι γραμμένος σαν κείμενο (συμβολική/symbolic μορφή) ώστε να είναι κατανοητός από τον προγραμματιστή
  - Ο κώδικας μηχανής είναι η μετάφραση των συμβόλων σε δυαδική (binary) μορφή που είναι κατανοητή από την CPU
- Ο assembler κάνει τα εξής βήματα
  - (1) Μεταφράζει τις εντολές μια μια από symbols σε binary codes
    - Η διαδικασία αυτή είναι σχετικά απλή – ουσιαστικά ο assembler χρησιμοποιεί έναν πίνακα που περιγράφει πως κάθε opcode και οι παράμετροί του κωδικοποιούνται σε binary code
    - π.χ. add r,r,r → 01001110 01001101 01011111 11010100
  - (2) Δεσμεύει χώρο όπως καθορίζουν οι global δηλώσεις για το global data segment
    - Αυτή η πληροφορία για τον πόσο χώρο χρειάζονται οι global μεταβλητές προέρχεται από τους τύπους του προγράμματος και ακολουθεί το πρόγραμμα μέχρι την δημιουργία του εκτελέσιμου
  - (3) Αντικαθιστά τα labels με πραγματικές διευθύνσεις

```
Label:  add...
...
branch Label
```

- Στη συνέχεια θα μιλήσουμε σε λεπτομέρεια για τον C pre-processor=cpp

cpp – C Pre-processor

- Ο cpp είναι ένα πρόγραμμα που ΔΕΝ σχετίζεται με την γλώσσα C και κάνει συγκεκριμένες ”δουλειές”
  - Οι γραμμές του προγράμματος .c που ξεκινούν με # αποτελούν οδηγίες προς τον cpp και δεν είναι μέρος της γλώσσας C
  - Ουσιαστικά το source πρόγραμμα .c περιέχει ένα μίγμα από οδηγίες προς τον cpp και από εντολές της γλώσσας C
- Ο cpp παίρνει σαν input ένα αρχείο text και:
  - 1. Αφαιρεί όλα τα σχόλια (/*…*/)
    - Με μεγαλύτερη ακρίβεια: Αντικαθιστά κάθε σχόλιο με τον κενό χαρακτήρα ‘ ‘ (decimal 32 στο ANSI character set)
  - 2. Επεξεργάζεται τις οδηγίες που ξεκινούν με #
  - 3. Συνενώνει τις γραμμές του κώδικα που είναι “σπασμένες” σε περισσότερες φυσικές γραμμές κειμένου με την χρήση του συμβόλου ”\”

```
printf(”this is a long line line \
                        line line line of text\n”);
```

    Θα γίνει από τον cpp:

```
printf(”this is a long line line line line line of text\n”)
```

  - Ο cpp δεν ενδιαφέρεται για το αν το source πρόγραμμα είναι C ή κάτι άλλο ή απλό text. Απλά κάνει αυτά τα βήματα.
- Ποιες είναι οι κύριες οδηγίες του cpp που ξεκινούν με # και που τις χρησιμοποιούμε συχνά;

#include

- Έστω ότι ένα f1.c περιέχει την οδηγία #include <filename>
- Πάρε το αρχείο filename και αντικατέστησε την γραμμή του include με τα περιεχόμενα του αρχείου μέσα στο f1.c
  - Μετά συνέχισε την επεξεργασία του αρχείου f1.c από την πρώτη γραμμή του αρχείου filename
- Όταν χρησιμοποιούμε <> ο cpp θα κοιτάξει για το αρχείο filename σε ένα σύνολο από προκαθορισμένα directories
- Αν χρησιμοποιήσουμε #include “filename”, τότε ο cpp θα κοιτάξει για το αρχείο στο directory που βρίσκεται το f1.c
- Ο cpp, όπως είπαμε, θα εκτελέσει (τυφλά) το #include άσχετα από το αν το source αρχείο είναι κώδικας C ή οτιδήποτε άλλο

.c vs. h source files

- Και τα .c και τα .h είναι C source files
  - Επίσης είναι και οι δύο τύποι αρχεία text
- #include uses only .h files – Γιατί;
- Αυτό αποτελεί μια σύμβαση, ως εξής:
  - Στα .h αρχεία τοποθετούμε ΜΟΝΟ δηλώσεις
  - Στα .c αρχεία τοποθετούμε τον κώδικά μας που εκτελείται
  - Το #include χρησιμοποιείται ΜΟΝΟ για να συμπεριλάβουμε δηλώσεις (.h) σε ένα άλλο αρχείο (.c ή .h)
- Αν δούμε κάποιο #include <file.c> αυτό σημαίνει συνήθως ότι προσπαθούμε να συμπεριλάβουμε σε ένα αρχείο μας κώδικα C (όχι δηλώσεις) και κάτι έχουμε κάνει λάθος στον σχεδιασμό του προγράμματός μας
  - Γιατί και πως τα .c και .h αρχεία περιέχουν ότι περιέχουν θα μιλήσουμε σε λεπτομέρεια όταν εξηγήσουμε τα modules, interfaces, και υλοποιήσεις
- Οι συμβάσεις αυτές είναι δική μας ευθύνη – ο cpp δεν μπορεί να τις ελέγξει και απλά θα εκτελέσει ότι οδηγίες του δώσουμε

#ifdef - #else - #endif

- ```
  #ifdef <NAME>
            <part 1>
  #else
            <part 2>
  #endif
  ```

- Παραλλαγή του #ifdef
  ```
  #if <expression>
      <part 1>
  #else
      <part 2>
  #endif
  ```

- `#define <NAME> <expr>`

- Αν το <NAME> έχει οριστεί προηγουμένως με κάποιο τρόπο (π.χ., αλλά όχι μόνο, με #define) τότε ο cpp θα συμπεριλάβει το <part 1> αλλά όχι το <part 2>
  - Αν το <NAME> δεν έχει οριστεί θα συμπεριλάβει μόνο το <part 2>
  - Π.χ.
    ```
    #ifdef LINUX
    #include <sys.h>
    #endif
    ```
- Αν το η έκφραση <expression> έχει αποτέλεσμα 0 τότε περιλαμβάνεται στο output το <part 2> διαφορετικά το <part 1>
  - Το <expression> πρέπει να μπορεί να υπολογιστεί κατά την μετάφραση του προγράμματος (π.χ. δεν μπορεί να περιέχει μεταβλητές, αφού δεν είμαι μέρος της γλώσσας)
  - Χρήση #if 0/#endif για να κάνουμε exclude κομμάτια κώδικα, ειδικά όταν περιέχουν σχόλια
- Ορίζει ότι το NAME θα αντικαθίσταται ”αυτολεξεί” στο υπόλοιπο source αρχείο μέχρι το τέλος του αρχείου με το <expression>, π.χ.
  - #define NCHARS 128 → όπου ο cpp θα βλέπει το NCHARS στο source μέχρι το τέλος του προγράμματος θα το αντικαθιστά με το 128
    ```

#define macros

- #define allows us to also use parameters
- `#define MAX(x,y) ( x >= y ? x : y )`
  - Θα αντικαταστήσει το MAX(x,y) όπου το συναντά με την αντίστοιχη έκφραση, όπου το x,y θα αντικαθίστανται με τις τιμές που χρησιμοποιούνται σαν παράμετροι στο MAX
  - MAX(a,4); → `( a >= 4 ? a : 4 );`
- Προσοχή! Τι θα συμβεί αν χρησιμοποιήσουμε:
  - `#define SUM(x,y) x+y`
  - και στο πρόγραμμά μας γράψουμε SUM(3, 1)*9
  - Ο cpp θα κάνει τυφλά την αντικατάσταση ως: 3+1*9
  - Εκφράζει αυτό όμως την πρόθεση μας όταν γράψαμε το SUM(3,1)*9; Μάλλον όχι… Εμείς θέλαμε το αποτέλεσμα να είναι 36, αλλά θα είναι 12…
  - Το αποτέλεσμα θα ήταν το επιθυμητό αν ορίζαμε:
  - `#define SUM(x,y) ( (x) + (y) )`
- Τα #define θέλουν προσοχή και γενικά αποφεύγουμε τα #define macros
  - Ο κώδικας που εκτελείται διαφέρει από τον κώδικα που γράψαμε γιατί έχει μεσολαβήσει ο cpp
  - Είναι προτιμότερο να χρησιμοποιούμε συναρτήσεις της γλώσσας αντί για macros ώστε να ελέγχει ο compiler τους τύπους και να βλέπουμε στο source αρχείο μας τον κώδικα που εκτελείται
  - Η πρόθεση να μειώσουμε με τα define macros το κόστος της κλήσης των συναρτήσεων είναι για εξεζητημένες μόνο περιπτώσεις

Τελεστής

- Ένωσε τις δύο παραμέτρους σαν strings
- Π.χ. #define TEMP(i) t ## i
  αν στο πρόγραμμά μας χρησιμοποιήσουμε
  x=TEMP(3); ο cpp θα το αντικαταστήσει με x=t3;
- Η C δεν περιλαμβάνει τέτοιο operator και είναι ”φασαρία” να πετύχουμε το ίδιο με άλλο τρόπο

Predefined variables and macros

- Ο cpp διαθέτει μερικές βοηθητικές μεταβλητές και macros. Συχνά χρησιμοποιούμε τα:
- __FILE__: αντικαθίσταται από ένα string που είναι το όνομα του αρχείου στο οποίο περιέχεται το __FILE__
- __LINE__: αντικαθίσταται από έναν αριθμό που είναι ο αριθμός γραμμής του αρχείου στο οποίο βρίσκεται η συγκεκριμένη χρήση της __LINE__, π.χ.

```
#define PRINTERR(s) \
        (“error in file %s, line %d: %s\n”,
__FILE__, __LINE__, s);
```

- Τυπώνει το αρχείο, την γραμμή και ένα string s σαν μήνυμα

Reading

- King, Ch. 14
- Bryant, Ch. 7.1
- More on cpp: The C Book, Ch. 7
- Everything about cpp: https://gcc.gnu.org/onlinedocs/cpp/
- More on the compilation process: Chapter 4, (online) Programming with GNU Software. Gary V. Vaughan, Akim Demaille, Paul Scott, Bruce Korb, and Richard Meeking. Edition 2, 2002. [pdf]