Laboratorio di Programmazione 1

Docente: dr. Damiano Macedonio

Lezione 18 – 31/03/2014

Funzioni: Dichiarazione e Definizione

- La *dichiarazione* di una funzione serve a comunicare al compilatore quali sono gli argomenti ed il tipo di ritorno di una funzione.
  - La dichiarazione di una funzione è detta *segnature*.

- Il compilatore richiede solo che una funzione sia dichiarata prima di utilizzarla, non è interessato alla sua definizione.

- Per utilizzare un insieme di funzioni in un programma senza curarsi dell'ordine con cui sono definite è possibile spostare le loro dichiarazioni all'inizio del file.

Funzioni: Header File

- Una soluzione migliore consiste nello spostare tutte le dichiarazioni in un file separato chiamato *header file*.
  - `#include <stdio.h>`
  - Include le dichiarazioni contenute nel file `stdio.h`.
  - Tale istruzione ha lo stesso effetto di specificare le dichiarazioni all'inizio del file.

- L'utilizzo di un header file per le dichiarazioni richiede di:
  - Creare un file con estensione `.h`
    - Se si sta scrivendo un programma chiamato `program.c`, l'header file sarà chiamato `program.h`.
    - All'interno di questo file vanno incluse le dichiarazioni di tutte le funzioni contenute nel programma, ad eccezione della funzione `main()`.
  - Includere l'header file nel programma principale.
    - `#include "program.h"`

Funzioni: Header File

- Ricordarsi di usare le virgolette per includere gli header file contenuti nella stessa directory (o in una sottodirectory) del vostro programma!
  - Le parentesi angolari < > sono usate per le librerie che si trovano all'esterno della directory contenente il vostro programma (es. Librerie standard).
  - All'interno delle virgolette " " si può specificare un path relativo (sottodirectory), anche se solitamente gli header file sono messi nella stessa directory del programma.
    - `#include "header/program.h"` POCO USATO!

Programmi Composti da più File

- Per poter riutilizzare facilmente delle funzioni è utile isolarle in file distinti da quello contenente il programma principale.
  - Suddividere il programma in parti auto-contenute.
- Il programma finale sarà ottenuto combinando i sorgenti contenuti nei vari file.
  - Per prima cosa i vari file vengono compilati in file oggetto separati, quindi il compilatore gli unisce (*linking*) in un unico programma eseguibile.
- Supponiamo di isolare le funzioni di un programma in un unico file `functions.c`, le cui definizioni sono contenute in un file `functions.h`, e di porre il `main()` in un altro sorgente `program.c`.

Programmi Composti da più File

```
int sum( int a, int b);
int pow( int a, int b);
```

functions.h

```
#include "functions.h"

int sum( int a, int b )
{
   ...
}

int pow( int a, int b){
   ...
}
```

functions.c

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

int main(){
   ...
   c = sum( a, b );
   d = pow( a, b );
   ...
}
```

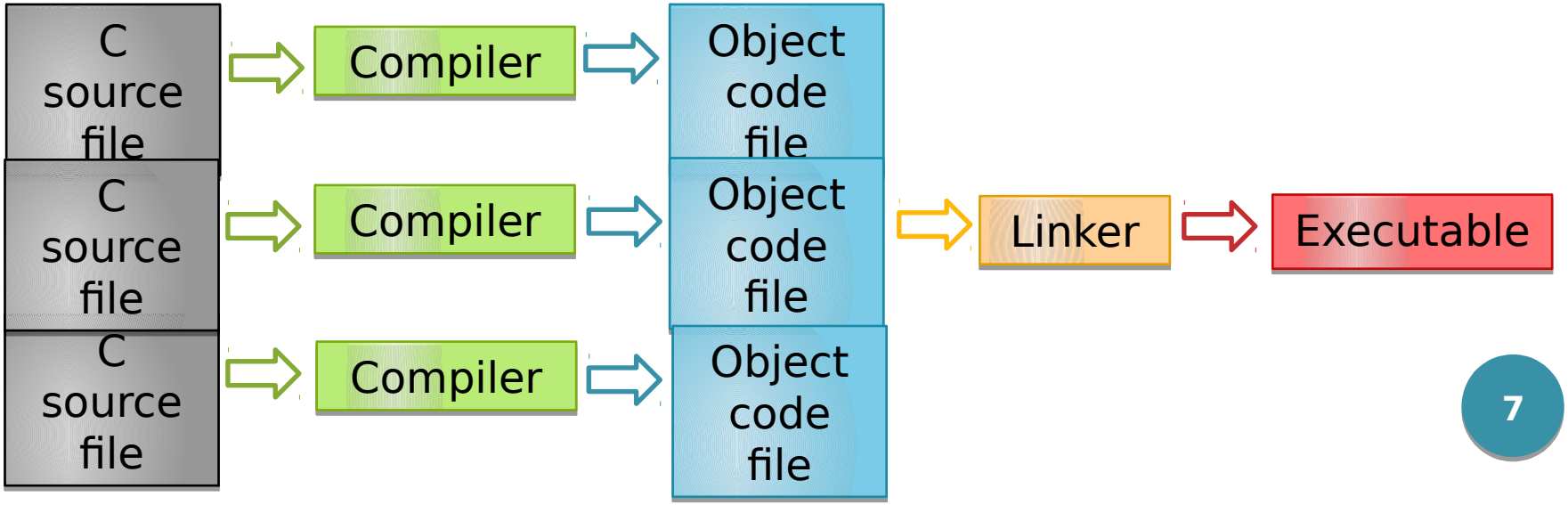
main.c

```
gcc functions.c main.c -o main
```

Programmi Composti da più File: Compilazione

- È possibile evitare di ricompilare ogni volta *tutti* i file che compongono un programma, ma limitarsi solo a quelli che hanno subito modifiche.

- Per fare questo è necessario salvare i file oggetto intermedi, in modo che il compilatore possa utilizzarli durante il linking.



Programmi Composti da più File: Compilazione

- Creare un file oggetto per ogni file sorgente:
  - `gcc -c *.c`
  - `-c` crea un file oggetto (estensione `.o`) per ogni sorgente compilato
  - `*.c` compila tutti i file .c contenuti nella directory.
- Linkare i file oggetto ottenuti:
  - `gcc *.o -o program_name`
  - Il compilatore riconosce che si tratta di file oggetto, e non di file sorgente, e quindi salta il passo di compilazione ed esegue solo il linking.
- Ad ogni successiva modifica è sufficiente eseguire il comando `gcc -c file.c` solo sul file modificato e poi eseguire il linking (`gcc *.o -o program_name`).

Librerie Standard di C

- `stdio.h`: operazioni di input/output
- `stdbool.h`: valori booleani
- `math.h`: operazioni matematiche
- `time.h`: operazioni per la manipolazione del tempo
- `string.h`: operazioni per la manipolazione di stringhe

Libreria Standard stdio.h

- Due funzioni già note:
  - `int scanf(const char *format, ...);`
    - Legge un dato (con un certo formato) dallo standard input e lo mette in una variabile passata come secondo parametro.
  - `int printf(const char *format, ...);`
    - Stampa una stringa (con un certo formato) nello standard output.
- Un'altra funzione utile:
  - `int getchar(void);`
    - Legge un carattere dallo standard input. Ritorna il carattere letto (eseguendo un cast ad intero).
    - Notare che il carattere digitato viene effettivamente letto solo dopo che l'utente ha premuto invio (carattere `'\n'`).

Lettura di una Riga di Input

```
// s e' un array di caratteri di lunghezza length + 1
void readline(char s[], int length) {
  int pos = 0;
  // consumo tutti i caratteri '\n' iniziali
  while((s[pos] = getchar()) == '\n');

  do
     s[++pos] = getchar();
  while(s[pos] != '\n' && pos < length);

  s[pos] = '\0';
}
```

Libreria Standard string.h

- L'header file `string.h` contiene le dichiarazioni di tipi e funzioni della libreria standard del C per la manipolazione delle stringhe.
- Le funzioni in `string.h` lavorano solamente con caratteri ASCII.
- In realtà la libreria contiene anche la definizione di tipi e funzioni per la gestione della memoria.
- Per una lista completa delle funzioni disponibili: `http://it.wikipedia.org/wiki/String.h`

Funzioni di string.h

- `char *strcat(char *dest, const char *src);` Concatena la stringa `src` alla stringa `dest` modificandola. Ritorna la stringa risultante. Il carattere `'\0'` della prima stringa è scartato.

- `char *strncat (char *dest, const char *src, size_t n);` Concatena al massimo *n* caratteri di `src` alla stringa `dest` e ritorna la stringa risultato.

- `int strcmp(const char *s1, const char *s2);` Confronta la stringa `s1` con `s2`, ritorna 0 se le due stringhe sono uguali, -1 se `s1 < s2`, oppure 1 altrimenti (il segno del risultato corrisponde al risultato della differenza tra `s1` e `s2`).

- `int strncmp (const char *, const char *, size_t);` Confronta al massimo *n* caratteri delle due stringhe.

Funzioni di string.h

- `char *strcpy(char *s1, const char *s2);` Copia la stringa `s2` nella stringa `s1`, incluso il carattere di terminazione `'\0'`.

- `char *strncpy(char *s1, const char *s2, size_t n);` Copia al massimo `n` caratteri della stringa `s2` nella stringa `s1`.

- `size_t strlen(const char *s);` restituisce la lunghezza della stringa `s`.