

Laboratorio reti AA 2006/2007

Dott. Matteo Roffilli

roffilli@csr.unibo.it

**Ricevimento in ufficio
dopo la lezione**

Laboratorio reti AA 2006/2007

Per esercitarvi fate SSH su:

`alfa.csr.unibo.it`

`si-tux00.csr.unibo.it`

....

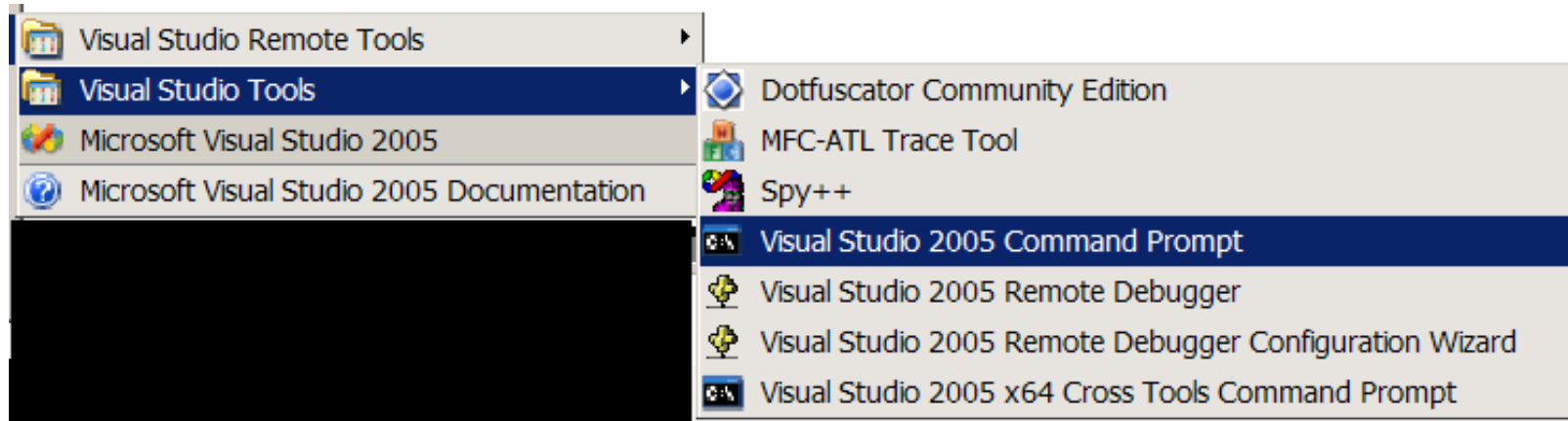
`si-tux15.csr.unibo.it`

Eventuali variazioni di orario/giorno verranno comunicate in anticipo via mail.

Laboratorio reti AA 2006/2007

- **Marzo**
- 6 Intro,SSH,VI/VIM,GCC base
- **19 Richiami di C e Compilazione**
- 26 Socket

Command line .NET



```
C:\WINDOWS\system32\cmd.exe - cl -?
D:\BIBLIO\BIBLIOTECHE\Reti_calcolatori_2006-07\lab2>cl -?
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 14.00.50727.42 for 80x86
Copyright (C) Microsoft Corporation. All rights reserved.

C/C++ COMPILER OPTIONS

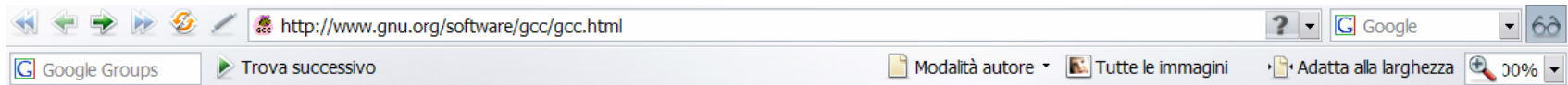
-O1 optimize for speed
-O2 optimize for space
-O3 optimize for both speed and space
-O4 optimize for both speed and space
-O5 optimize for both speed and space
-O6 optimize for both speed and space
-O7 optimize for both speed and space
-O8 optimize for both speed and space
-O9 optimize for both speed and space
-O10 optimize for both speed and space
-O11 optimize for both speed and space
-O12 optimize for both speed and space
-O13 optimize for both speed and space
-O14 optimize for both speed and space
-O15 optimize for both speed and space
-O16 optimize for both speed and space
-O17 optimize for both speed and space
-O18 optimize for both speed and space
-O19 optimize for both speed and space
-O20 optimize for both speed and space
-O21 optimize for both speed and space
-O22 optimize for both speed and space
-O23 optimize for both speed and space
-O24 optimize for both speed and space
-O25 optimize for both speed and space
-O26 optimize for both speed and space
-O27 optimize for both speed and space
-O28 optimize for both speed and space
-O29 optimize for both speed and space
-O30 optimize for both speed and space
-O31 optimize for both speed and space
-O32 optimize for both speed and space
-O33 optimize for both speed and space
-O34 optimize for both speed and space
-O35 optimize for both speed and space
-O36 optimize for both speed and space
-O37 optimize for both speed and space
-O38 optimize for both speed and space
-O39 optimize for both speed and space
-O40 optimize for both speed and space
-O41 optimize for both speed and space
-O42 optimize for both speed and space
-O43 optimize for both speed and space
-O44 optimize for both speed and space
-O45 optimize for both speed and space
-O46 optimize for both speed and space
-O47 optimize for both speed and space
-O48 optimize for both speed and space
-O49 optimize for both speed and space
-O50 optimize for both speed and space
-O51 optimize for both speed and space
-O52 optimize for both speed and space
-O53 optimize for both speed and space
-O54 optimize for both speed and space
-O55 optimize for both speed and space
-O56 optimize for both speed and space
-O57 optimize for both speed and space
-O58 optimize for both speed and space
-O59 optimize for both speed and space
-O60 optimize for both speed and space
-O61 optimize for both speed and space
-O62 optimize for both speed and space
-O63 optimize for both speed and space
-O64 optimize for both speed and space
-O65 optimize for both speed and space
-O66 optimize for both speed and space
-O67 optimize for both speed and space
-O68 optimize for both speed and space
-O69 optimize for both speed and space
-O70 optimize for both speed and space
-O71 optimize for both speed and space
-O72 optimize for both speed and space
-O73 optimize for both speed and space
-O74 optimize for both speed and space
-O75 optimize for both speed and space
-O76 optimize for both speed and space
-O77 optimize for both speed and space
-O78 optimize for both speed and space
-O79 optimize for both speed and space
-O80 optimize for both speed and space
-O81 optimize for both speed and space
-O82 optimize for both speed and space
-O83 optimize for both speed and space
-O84 optimize for both speed and space
-O85 optimize for both speed and space
-O86 optimize for both speed and space
-O87 optimize for both speed and space
-O88 optimize for both speed and space
-O89 optimize for both speed and space
-O90 optimize for both speed and space
-O91 optimize for both speed and space
-O92 optimize for both speed and space
-O93 optimize for both speed and space
-O94 optimize for both speed and space
-O95 optimize for both speed and space
-O96 optimize for both speed and space
-O97 optimize for both speed and space
-O98 optimize for both speed and space
-O99 optimize for both speed and space
-O100 optimize for both speed and space

-CODE GENERATION-
/GF enable read-only string pooling
/Gm[-] enable minimal rebuild
/GS[-] enable security checks
/GX[-] enable C++ EH (same as /EHsc)
/EHs enable C++ EH (no SEH exceptions)
/EHa enable C++ EH (w/ SEH exceptions)
```

GCC

- La GNU Compiler Collection (solitamente abbreviata in GCC) è un insieme di compilatori creato inizialmente dall'hacker americano **Richard Stallman** come parte del Sistema GNU, un sistema operativo libero compatibile con Unix.
- Le versioni recenti di GCC, sviluppate dalla comunità **Open Source**, sono incorporate nelle principali distribuzioni del sistema operativo GNU/Linux, e di molti altri sistemi basati su fondamenta Unix, come per esempio Mac OS X.
- Nata inizialmente come un compilatore per il **linguaggio C** (il nome in origine era GNU C Compiler), GCC dispone oggi di vari front end per altri linguaggi, tra cui Java, C++ , Objective_C ,Fortran e Ada, ed è in grado di generare eseguibili per molte architetture, tra le quali x86 (intel compatibili), x86 a 64 bit (AMD/intel a 64 bit), PowerPC, s390 (IBM), Sparc (Sun Microsystems).
- L'ultima versione rilasciata è la **4.1.2**

GCC homepage



GCC, the GNU Compiler Collection

The GNU Compiler Collection includes front ends for C, C++, Objective-C, [Fortran](#), [Java](#), and Ada, as well as libraries for these languages ([libstdc++](#), [libgcj](#),...).

We strive to provide regular, high quality [releases](#), which we want to work well on a variety of native and cross targets (including GNU/Linux), and encourage everyone to [contribute changes](#) and [help testing](#) GCC. Our sources are readily and freely available via [SVN](#) and [weekly snapshots](#).

Major decisions about GCC are made by the [steering committee](#), guided by the [mission statement](#).



News

March 9, 2007

All m68k targets now support ColdFire processors and offer the choice between ColdFire and non-ColdFire libraries at configure time. There have been [several other significant changes](#) to the m68k and ColdFire support. This work was contributed by Nathan Sidwell of CodeSourcery and others.

February 13, 2007

[GCC 4.1.2](#) has been released.

January 25, 2007

Interprocedural optimization passes have been reorganized to operate on SSA. This enables more precise function analysis and optimization while inlining, significantly improving the performance of programs with high abstraction penalty. Code from [ipa-branch](#) contributed by Jan Hubicka, SUSE labs and

Status

Current release series: [GCC 4.1.2](#)

Status: [2007-02-14](#) (regression fixes and docs only).
[Serious regressions](#). [All regressions](#).

Previous release series: [GCC 4.0.4](#)

Status: The branch has been closed after the release of GCC 4.0.4.
[Serious Regressions](#).

Next release series: [GCC 4.2.0](#) ([changes](#))

Status: [Stage 3](#); [2007-03-12](#) (regression fixes & docs only).
[Serious regressions](#). [All regressions](#).

Active development: [GCC 4.3.0](#) ([changes](#))

Status: [Stage 1](#); [2006-10-17](#)
[Serious regressions](#). [All regressions](#).

Search our site

About GCC

[Mission Statement](#)
[Releases](#)
[Snapshots](#)
[Mailing lists](#)
[Contributors](#)
[Steering Committee](#)

Documentation

[Installation](#)
· [Platforms](#)
· [Testing](#)
[Manual](#)
[FAQ](#)
[Wiki](#)
[Further Readings](#)

Download

[Mirror sites](#)
[Binaries](#)

"Live" Sources

[SVN read access](#)
[Rsync read access](#)
[SVN write access](#)

Development

[Development Plan](#)
· [Tentative Timeline](#)
[Contributing](#)
[Why contribute?](#)
[Open projects](#)
[Front ends](#)
[Back ends](#)
[Extensions](#)
[Benchmarks](#)

Bugs

GCC

Il gcc fornisce al programmatore un controllo esteso del processo di compilazione.

Il processo di compilazione può comprendere fino a quattro stadi distinti:

1. Fase di preprocessing
 2. Compilazione vera e propria
 3. Assemblaggio
 4. Linking
- E' possibile interrompere il processo di compilazione in uno qualsiasi di questi stadi per esaminare il risultato della compilazione.
 - Con gcc è anche possibile controllare la quantità e il tipo di informazioni di **debugging** così da includerle nel file binario prodotto.
 - Come la maggior parte dei compilatori consente di eseguire l'ottimizzazione del codice.
 - gcc è anche un **cross compilatore**, per cui è possibile sviluppare codice utilizzando un particolare processore e farlo poi girare su un altro.

GCC comandi base

gcc

ld

ar

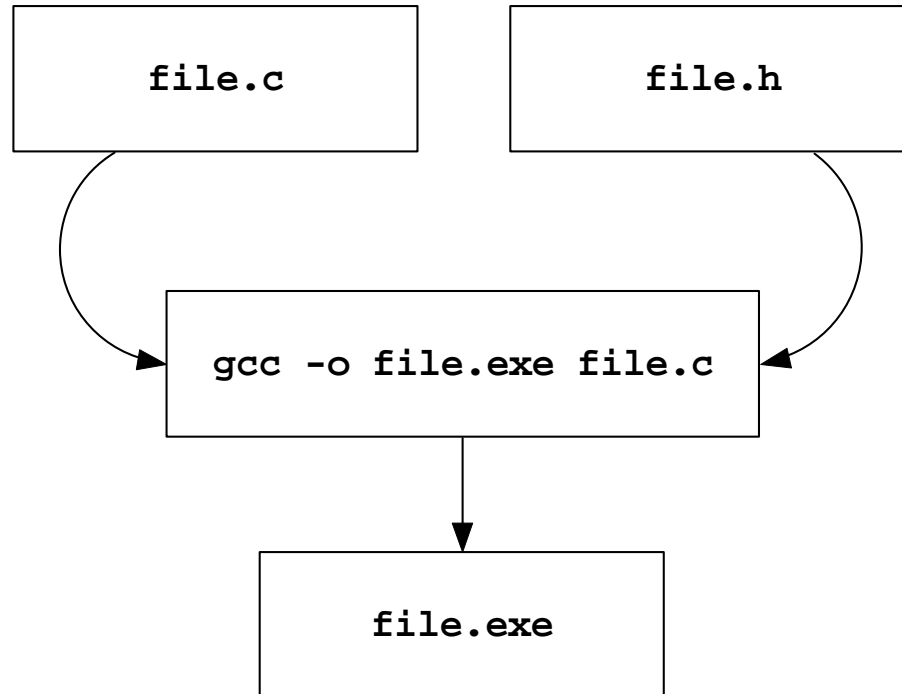
make

gdb

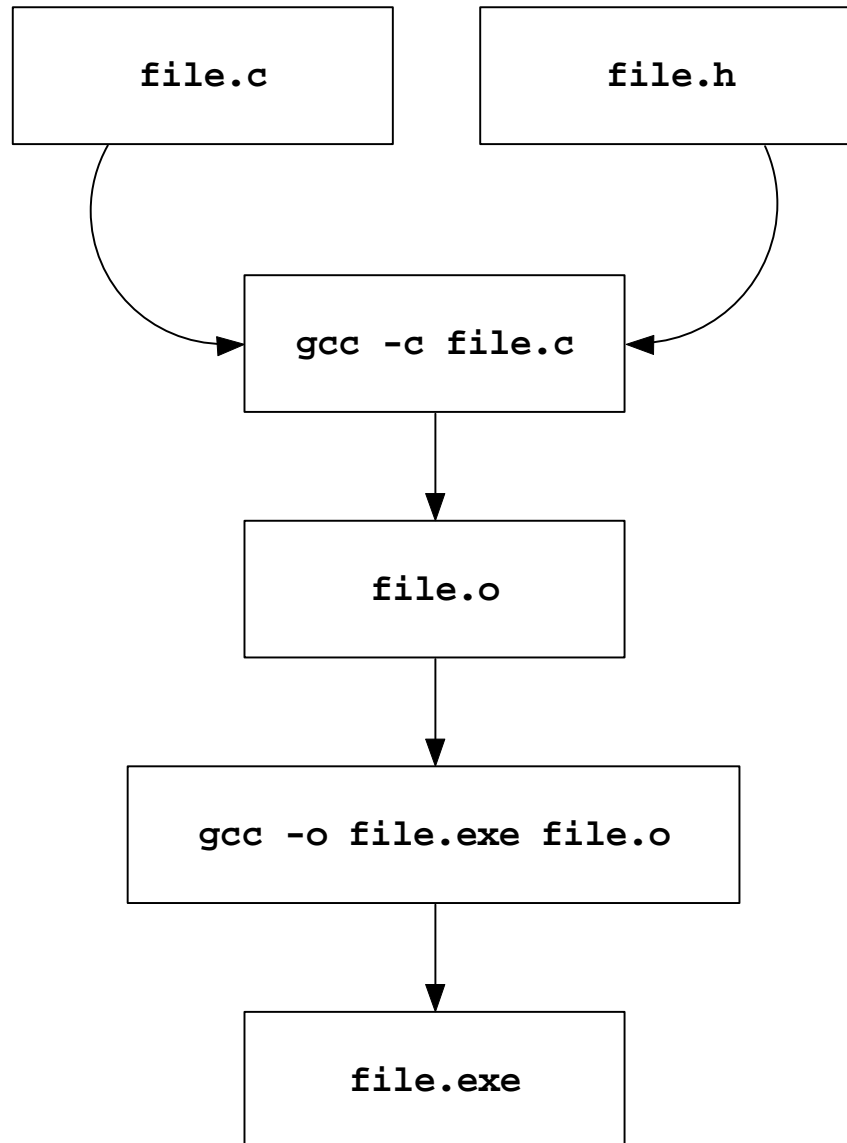
Caro buon vecchio C

```
/* first.c */  
  
#include <stdio.h>  
int main()  
{  
    printf("Hello, world\n");  
    return 0;  
}
```

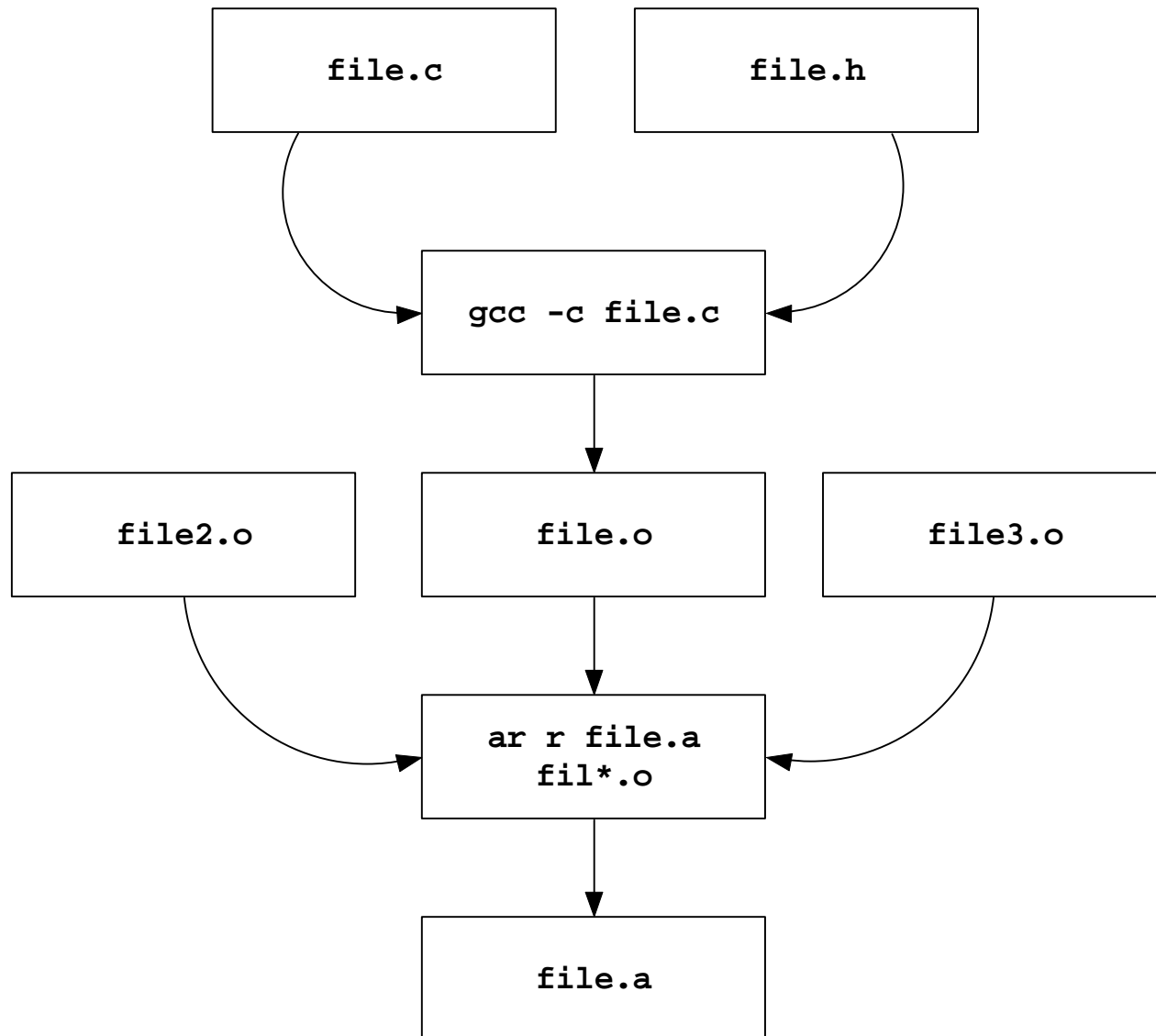
GCC compilazione



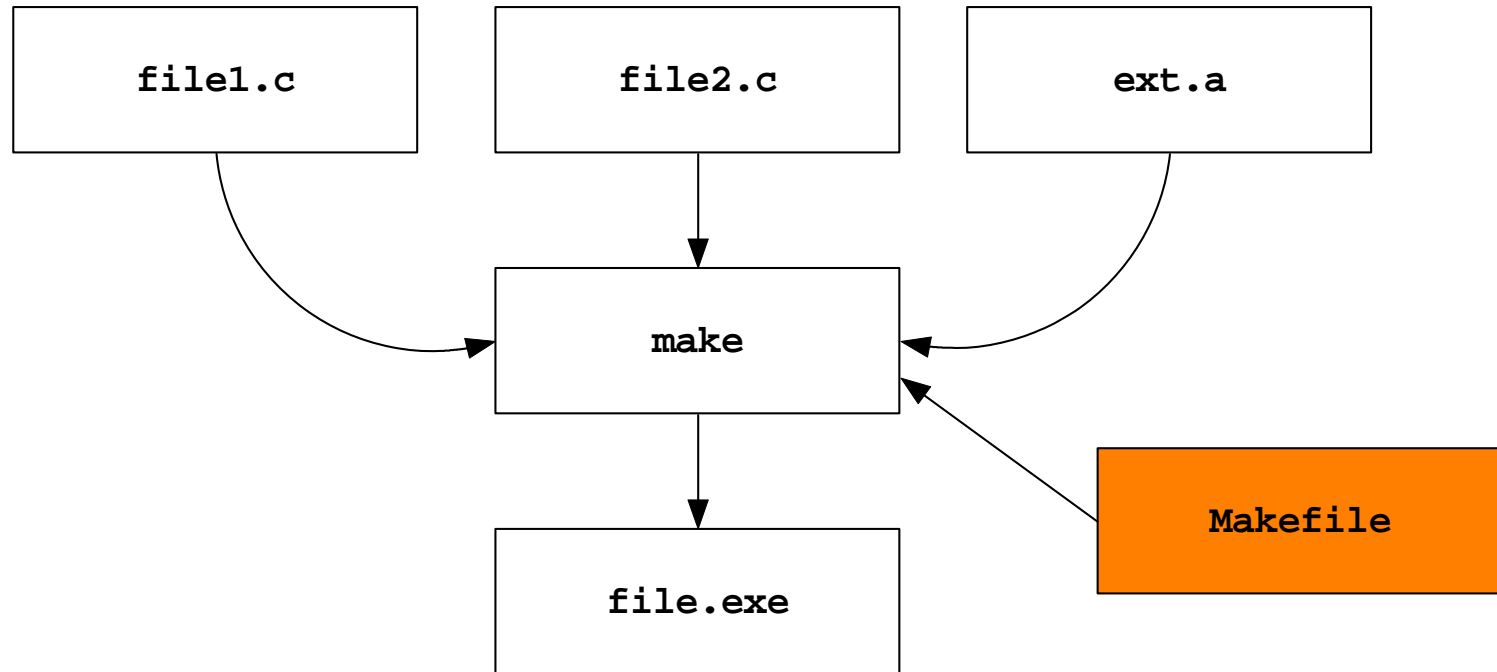
GCC linking



AR creare librerie



MAKE gestire un progetto



Un esempio di Makefile

```
# Makefile
COMPILER = gcc -Wall
LIBS = -lm
EXECUTABLE = first
OBJECT = first.o
$(EXECUTABLE): $(OBJECT)
$(COMPILER) -o $(EXECUTABLE) $(OBJECT)
$(LIBS)
%.o: %.c
$(COMPILER) -o $*.o -c $*.c
```

Un po' di esercizio

Goal:

realizzare un programma che legge a riga di comando 2 stringhe (A e B) e crea una nuova stringa C concatenando A a B: $C=AB$

Esempio:

1. Input: “capo” “colle”
2. Output: “capocolle”

Tempo a disposizione:

15 minuti

Un po' di esercizio 2

Goal:

realizzare un programma che legge a riga di comando 2 numeri a virgola mobile in singola precisione e stampa a video la loro somma

Requisiti:

1. La funzione di somma deve essere implementata in un file esterno chiamato **filext.c**
2. Con file filext.c compilato deve essere creata una libreria **functions.a**
3. Il file del main, chiamato **main.c**, deve essere compilato
4. Il main compilato e la libreria **functions.a** devono essere linkati in un eseguibile chiamato **somma.exe**
5. L'eseguibile deve funzionare!!!

Tempo a disposizione:

25 minuti

main.c

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

extern float somma(float a, float b);

int main(int argc, char *argv[])
{
    float a=0.0,b=0.0;
    if(argc!=3) exit(-1);

    a=(float)atof(argv[1]);
    b=(float)atof(argv[2]);
    printf("\n%2.3f + %2.3f = %2.3f\n",a,b,somma(a,b));

    return 0;
}
```

filext.c

```
float somma(float a, float b)
{
return(a+b);
}
```

sequenza comandi

```
gcc -Wall -c filext.c
```

```
gcc -Wall -c main.c
```

```
ar r functions.a filext.o
```

```
gcc -o somma.exe main.o functions.a
```