2' Esercitazione

https://politecnicomilano.webex.com/meet/gianenrico.conti

Gian Enrico Conti

- C Programming: 2' part

Esercizi:

1) Struct e Union
2) Cast di puntatori
3) Lib di funzioni - F. sum e sub in 2 file
4) Function Pointers

Make (def)

5) Make e pattern
6) Make e generazioni di lib da piu file
7) Make e dipendenze (prepend)
8) Allineamento e "Pack"
10) Macro
11) Macro: errori piu comuni.
10)"Endianess"

Shell: small recap

- Create sempre cartella per vs. progetti. Esempio: **mkdir** EX01
- Comandi:
  - Entrare in una cartella **cd** es: cd EX01
  - Risalire di un livello **cd..**
  - File presenti ls -la (list **all** even hidden in **list** mode)
  - Editor: nano o vi

- Gcc: opzioni
  -o `outputfile`
  -Wall
  -v
  -lm (es. *Math.. more about later)*
  -save-temps

  Esempio recap

  gcc -save-temps -Wall -lm hello.c -o hello

Struct, union e typedef

- **Struct** Permettono l'aggregazione di più variabili, in modo simile a quella degli array, ma a differenza di questi non ordinata e non omogenea (una struttura può contenere variabili di tipo diverso).

```
struct <name> {
   field1;
   field2;
   …
}
```

- **Union** Serve per memorizzare (in istanti diversi) oggetti di differenti dimensioni e tipo, con, in comune, il ruolo all'interno del programma

```
union {
   .. field1;
   .. field2;
   …
}
```

NOTA: field1 e field2 usano lo stesso spazio, e sizeOf e' il max fra i due sizeof!

- **Typedef** Per definire nuovi tipi di dato viene utilizzata la funzione typedef

Esercizio 1: Using Union(ex01)

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

typedef union MixedData{
    char c;
    int n;
    char s[21];
}MixedData;

int main(){
    MixedData md;
    printf("sizeof: %ld\n", sizeof(md));

    strcpy(md.s, "HELLO");
    md.c = 65;
    printf("%s\n", md.s);

    md.n = 66;
    printf("%s\n", md.s);

    MixedData mdArray[2];
    printf("sizeof: %ld\n", sizeof(mdArray));

    return 0;
}
```

Si noti:

- Sizeof
- sovrascrittura...
- Array...

```
gcc  ex01.c -Wall -o ex1
```

Esercizio 1: Using Union(ex01)

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

typedef union MixedData{
    char c;
    int n;
    char s[21];
}MixedData;

int main(){
    MixedData md;
    printf("sizeof: %ld\n", sizeof(md.s));

    strcpy(md.s, "HELLO");
    md.c = 65; //'A' in Ascii
    printf("%s\n", md.s);

    md.n = 66; // int: 66 | 0 | 0 | 0
    printf("%s\n", md.s);
    return 0;
}
```

Si noti:

- Sizeof
- sovrascrittura...
- Array...

```
sizeof: 24
AELLO
B
sizeof: 48
```

Cast di puntatori

- state forzando la macchina a prendere dati dalla RAM
- Se ci sono problemi di allineamento, risultato "undefined"

The C Standard, 6.3.2.3, paragraph 7 [ISO/IEC 9899:2011], states
A pointer to an object or incomplete type may be converted to a pointer to a different object or incomplete type. If the resulting pointer is not correctly aligned for the referenced type, the behavior is undefined.

- Da "grande a piccolo" safe
- Da "piccolo a grande pericoloso!"
- *(Byte ordering.. more about later..)*

Esercizio 2: Cast di puntatori (ex02)

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



int main(){
    //char dummy = "X";
    char c = 'A';
    char trailing1 = 0;
    char trailing2 = 0;

    int *ip = (int *)&c; /* This can lose information */
    char *cp = (char *)ip;

    /* Will fail on some conforming implementations */
    printf("cp dereferenziato:\n%c\n", *cp);
    printf("ip dereferenziato:\n%d\n", *ip);
    printf("c:\n%c\n", c);

    return 0;
}
```

```
- (S)Commentiamo dummy...
```

Esercizio 2: Cast di puntatori (ex02)

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



int main(){
    //char dummy = "X";
    char c = 'A';
    char trailing1 = 0;
    char trailing2 = 0;

    int *ip = (int *)&c; /* This can lose
information */
    char *cp = (char *)ip;

    /* Will fail on some conforming
implementations */
    printf("cp dereferenziato:\n%c\n", *cp);
    printf("ip dereferenziato:\n%d\n", *ip);
    printf("c:\n%c\n", c);

    return 0;
}
```

```
//char dummy = "X";


cp dereferenziato:
A
ip dereferenziato:
65
c:
A
```

```c
    char dummy = "X";
```

```
cp dereferenziato:
A
ip dereferenziato:
30273
c:
A
```

Esercizio 3: Lib di funzioni

Logica

- main() in suo file
- f. sum() " "
- f. sub() " "
- Liberia binaria contenente sum e sub
- Chiamare sum e sub nel main -> header.

Esercizio 3: Lib di funzioni (II) 1 step:

```c
#include "sum.h"
#include "sub.h"

int main(){

    printf("%d\n", sum(30, 20));
    printf("%d\n", sub(30, 20));
    return 0;
}
```

Gia visto gcc:

```
gcc -Wall ex03.c sub.c sum.c -o exe3
```

A NON e' una lib...

Esercizio 3: Lib di funzioni (III) 2 step:

```
Creo object solo x le f.:


gcc -Wall -c  sub.c sum.c


Opz.  -c   compile BUT not link...



Output:

sum.o
sub.o
```

Esercizio 3: Lib di funzioni 4' step:

Generiamo delle Lib STATICHE (more about static/dynamic later..)

```
gcc -Wall -c  sub.c sum.c
```

Opz. -c   compile BUT not link...

Output:

sum.o
sub.o

uniamole: (ar == archive)

```
 ar -qc libmylib.a *.o
```

opzioni.. man :)

Esercizio 3: Lib di funzioni 5' step:

Controlliamo il contenuto:

```
nm libmylib.a

mylib.a(sub.o):
0000000000000000 T _sub

mylib.a(sum.o):
0000000000000000 T _sum
```

Esercizio 3: Lib di funzioni 6' step;

uniamo tutto:

```
gcc -Wall ex03.c -o ex03
```

Fallisce:

```
"_sub", referenced from:
      _main in ex03-efb8f3.o
  "_sum", referenced from:
      _main in ex03-efb8f3.o
```

Esercizio 3: Lib di funzioni 7: FIX

Manca la ns lib:

```
gcc -Wall ex03.c -L. -lmylib -o ex03
```

Attenzione ai nomi!
Deve iniziare x lib

-L dove cecare le lib: si noti il punto

```
 ./exe03
50
10
```

Puntatori

Puntatori a funzioni

```
#include <stdio.h>

void   myFunc(int i){
      printf("Valore: %d\n",i);
}

void (*foo) (int);

int main(){
   int i=10,j=5;
   myFunc(i);
   foo=myFunc;
   foo(j);
   (*foo)(j);
}
```

Quale delle due chiamate a foo è quella corretta? Cosa stampa foo?

→ Entrambe, una volta assegnato al puntatore di funzione l'indirizzo della funzione che si vuole chiamare non fa differenza il fatto di dereferenziare o meno il puntatore a funzione al momento della chiamata. `foo(j)` stampa 5.

Puntatori a funzioni

A che servono?

- molte librerie hanno callback
- Solo noi sappiamo cosa fare sul dato che le lib ci mandano:
  - Es quick sort: (man qsort)

```
void qsort(void *base, size_t nitems, size_t size,
int (*compar)(const void *, const void*))
```

- Permettono codice altamente configurabile.

Esercizio 4: function pointers

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


void   myFuncA(int i){
      printf("in A: %d\n",i);
}

void   myFuncB(int i){
      printf("in B: %d\n",i);
}

//void (*foo) (int);
typedef void (*FooPtr)(int);


int main(){

    FooPtr f1 = myFuncA;
    f1(10);

    return 0;
}
```

```
Solo creato typedef..
```

Esercizio 4: function pointers II

```c
void   myFuncA(int i){
      printf("in A: %d\n",i);
}

void   myFuncB(int i){
      printf("in B: %d\n",i);
}

typedef void (*FooPtr)(int);

void multipleFoo(FooPtr ff){
    int i = 0;
    for (i=0; i<10; i++) {
        ff(i);
    }
}

int main(){

    FooPtr f1 = myFuncA;
    f1(10);

    multipleFoo(myFuncA);

    return 0;
}
```

Run...

E customizzare..

Esercizio 4: function pointers: customizzare ..

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


int  myFuncA(int i){
//      printf("in A: %d\n",i);
      return i*3;
}

typedef int (*FooPtr)(int);

int  falloPiuVolte(FooPtr ff){
    int i;
    int tot = 0;
    for (i=0;i<10; i++){
        tot+=ff(i);
    }
    return tot;
}

int main(){

    int k = falloPiuVolte(myFuncA);
    printf("%d\n",k);
    return 0;
}
```

Makefile

GNU Make

- Determina automaticamente quali parti di un programma complesso devono essere ricompilate
- Esegue i comandi utili alla loro ricompilazione
- È lo strumento standard usato da chi sviluppa per Linux

I Makefile definiscono

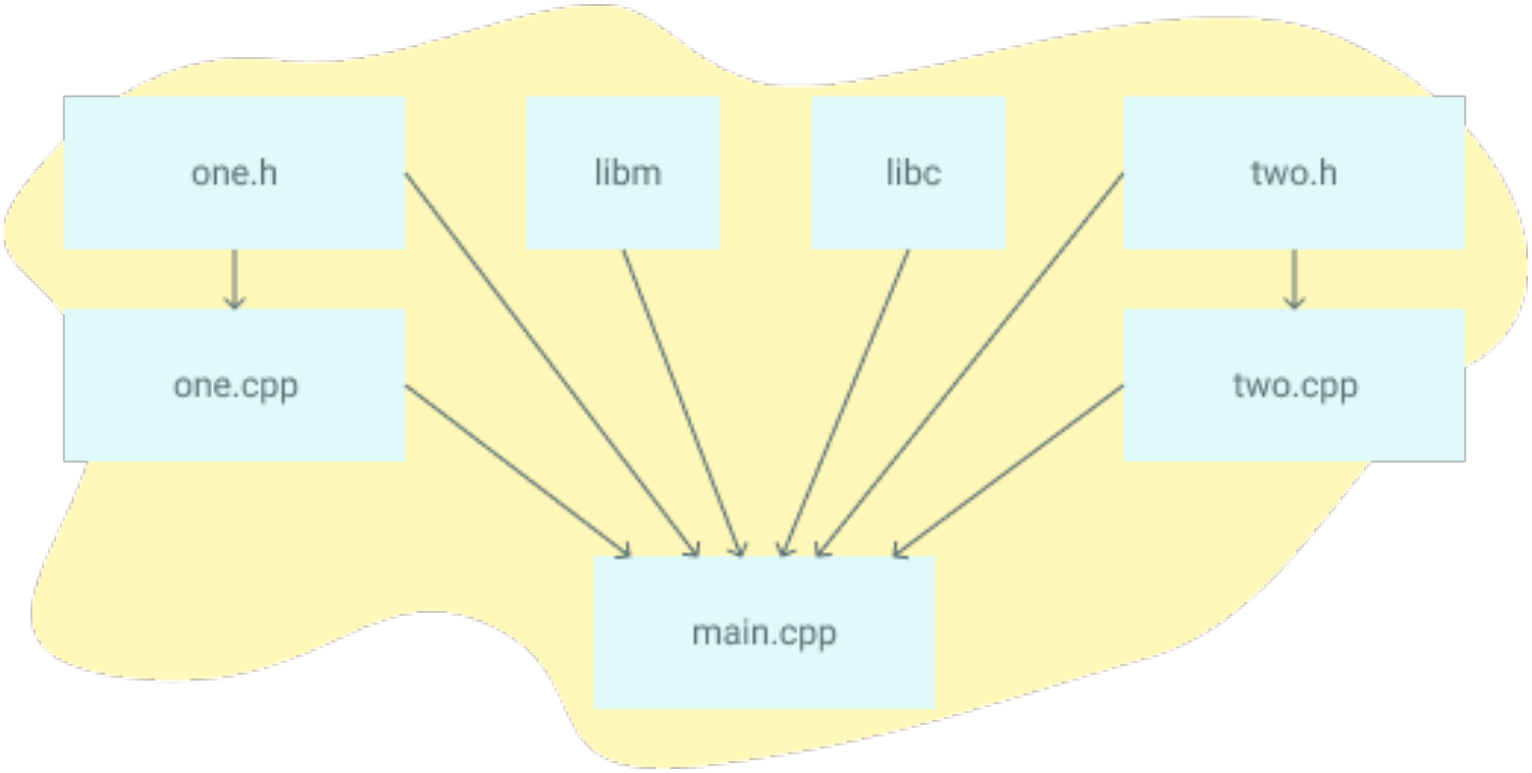
- *Target*: file che vogliamo compilare / ricompilare
- *Goals*: istruzioni per come compilare / ricompilare i *target*
- *Dependencies*: indicano quali *target* devono essere ricompilati a seguito di una modifica

Lanciare il comando *make*

```
make [options] [goal...]
```

`-C dir`	Esegue il comando `cd dir` prima di iniziare
`-f file`	Specifica un makefile diverso da quelli di default
`-j [n]`	Esegue `n` job in parallelo, Se n è omesso, esegue quanti job possibili
`-n`	Stampa i comandi richiesti per aggiornare il goal senza eseguirli

Dependencies



Esercizio 5: MAKE elementi

"make" tool
Un file Makefile

A Makefile consists of a set of *rules*. A rule generally looks like this:

```
targets: prerequisites
    command
    command
    command
```

- The *targets* are file names, separated by spaces. Typically, there is only one per rule.
- The *commands* are a series of steps typically used to make the target(s). These *need to start with a tab character*, not spaces.
- The *prerequisites* are also file names, separated by spaces. These files need to exist before the commands for the target are run. These are also called *dependencies*

Esercizio 5: MAKE esempi:

```
all: hello.exe

hello.exe: hello.o
	gcc -o hello.exe hello.o

hello.o: hello.c
	gcc -c hello.c

clean:
	rm hello.o hello.exe


say_hello:
	@echo "Hello World"
```

Anche cmd standard.

Esercizio 5: MAKE ns file:

```
mylibmake: ex05.c sum.c sub.c sum.h sub.h
	gcc -Wall -c  sub.c sum.c
	ar -qc libmylib.a *.o
	gcc -Wall ex05.c -L. -lmylib -o ex05

clean:
	 rm *.o
	 rm *.a
```

Esercizio 4: MAKE e pattern

https://www.gnu.org/software/make/manual/html_node/Pattern-Examples.html#Pattern-Examples

```
%.o : %.c
        $(CC) -c $(CFLAGS) $< -o $@
```

```
 gcc -Wall  hello.c -o hello.o
```

1' riga: any file *x*.o from *x*.c.

2' riga: ricetta:

"automatic variables '$@' and '$<' to substitute the names of the target file and the source file in each case where the rule applies"

8) Allineamento e "Pack"

Ogni μp / S.O. allinea on a "natural boundary"

```
typedef union MixedData{
    char c;
    int n;
    char s[21];
}MixedData;
```

Spreco?

8) Allineamento e "Pack"

```
typedef union MixedData{
    char c;
    int n;
    char s[21];
}MixedData;

MixedData md = ...
md.n = 0XAABBCCDD;
```

```
        |   1     |   2     |   3   |       4     |
| c (1) |.............................. |
| n (4) | 0xAA(1) | 0xBB(2) | 0xCC(3)   | 0xDD(4) |
| s (21)| ............................. |
|       | ............................. |
```

```
PACK riallinea senza "buchi" (padding)
```

8) Allineamento e "Pack"

```
typedef union MixedData{
    char c;
    int n;
    char s[21];
}MixedData;

MixedData md = ...
md.n = 0xXAABBCCDD;
```

`#pragma pack(1)` ensures that C `struct/union` items are packed in order and on **byte** boundaries

1 == granularity (1 byte)

#pragma arriva al compilatore!

8) Allineamento e "Pack"

```
typedef struct Point{
    char c;
    double x,y;
}MixedData;

MixedData md = ...
```

1. `#pragma pack(`*n*`)` simply sets the new alignment.
2. `#pragma pack()` sets the alignment to the one that was in effect when compilation started (see also command line option `-fpack-struct[=<n>]` see Code Gen Options).
3. `#pragma pack(push[,`*n*`])` pushes the current alignment setting on an internal stack and then optionally sets the new alignment.
4. `#pragma pack(pop)` restores the alignment setting to the one saved at the top of the internal stack (and removes that stack entry). Note that `#pragma pack([`*n*`])` does not influence this internal stack; thus it is possible

9) **MACRO**

Moltissimo usate:

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

Pericolose! Provate:

```
#include <stdio.h>
#define MAX(a,b) ((a) > (b) ? a : b)
#define MIN(a,b) ((a) < (b) ? a : b)

int main(int argc, const char * argv[]) {
   float i = 9.6;
   float w = MAX(i++,9);
   printf("%f\n", w);
   return 0;
}
```

Rida': 10.600000

MI ASPETTEREI: 9.6 !

9) MACRO

Moltissimo usata:

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

Pericolose!

E se usassi:

```
 int max(int a, int b){
  if...
}
```

Ma con optimization flag: -O03

```
Int k = max(10, 20);    ->>>> int k = 20;
```

10) big /little endian

RISC big endian

Intel little-end

ARM bi - endian

0xaabbccdd

Little: 0xAA cella PIU bassa in memoria.
Big : 0xDD cella PIU bassa in memoria.