2021_12_09

December 10, 2021

1 2021-12-09

1.0.1 Corso ITS

1.1 Magento & e-commerce software

1.2 ### Fondamenti di Programmazione (Andrea Ribuoli)

1.3 Percorso librerie dinamiche

Le variabili d'ambiente differiscono in base al sistema operativo adottato: **DYLD_LIBRARY_PATH** .vs. **LIBPATH** .vs. **LD_LIBRARY_PATH**

1.3.1 in MacOS

```
export DYLD_LIBRARY_PATH=./math_api:$DYLD_LIBRARY_PATH
./test_driver 66 121
quiz(66, 121) = 11
```

1.3.2 in AIX/PASE

```
export LIBPATH=./math_api:$LIBPATH
./test_driver 66 121
quiz(66, 121) = 11
```

1.3.3 in Linux

```
export LD_LIBRARY_PATH=./math_api:$LD_LIBRARY_PATH
./test_driver 66 121
quiz(66, 121) = 11
```

1.3.4 in Windows (10)

- In Search, search for and then select: System (Control Panel)
- Click the **Advanced system settings** link.
- Click **Environment Variables**. In the section **System Variables** find the `PATH` environment variable and select it. Click **Edit**. If the `PATH` environment variable does not exist, click `New`.
- In the **Edit System Variable** (or **New System Variable**) window, specify the value of the `PATH` environment variable. Click **OK**. Close all remaining windows by clicking **OK**.
- Reopen Command prompt window.

[1]:
```
!cd ../2021_12_06/EserciziC/20211206; \
 export LIBPATH=./math_api:$LIBPATH; \
 ./test_driver 66 121
```

```
quiz(66, 121) = 11
```

1.4 Sequenza opzioni in GCC per Linux

Attenzione ad indicare le shared library alla fine:

```
gcc -o test_driver -L./math_api test_driver.o -lmath
```

1.5 I misteri della opzione -Wl,-rpath

Se con `-L./math_api` aggiungiamo il percorso delle nostre API nella ricerca delle shared library cosa succede in esecuzione?

Abbiamo già mostrato il ruolo della variabile d'ambiente **LD_LIBRARY_PATH** ma si può fare qualcosa di più in fase di build.

```
gcc -o test_driver -L./math_api -Wl,-rpath="./math_api" test_driver.o -lmath
```

In genere tuttavia non è consigliato impacchettare il software con percorsi di ricerca delle shared library ma bensì affidarsi ai default della piattaforma.

Inoltre ha poco senso impostare una percorso di ricerca **relativo alla directory corrente** (ad eccezione delle fasi di test) perchè *questa facilmente non coincide con quella di installazione del programma.*

Alcune piattaforme (ad esempio **Linux**) supportano la variabile `$ORIGIN` a cui può avere un senso più compiuto legarsi:

```
gcc -o test_driver -L./math_api -Wl,-rpath="\$$ORIGIN/math_api" test_driver.o -lmath
```

Per evitare il problema noto col nome di **dependecy hell** ogni distribuzione Linux adotta un criterio molto rigoroso per definire le proprie linee guida per l'installazione delle dipendenze.

1.5.1 stages

- **-E** Preprocess only; do not compile, assemble or link.
- **-S** Compile only; do not assemble or link.
- **-c** Compile and assemble, but do not link.

opzione	Pre-process	Compile	Assemble	Link
-E	Sì	No	No	No
-S	Sì	Sì	No	No
-c	Sì	Sì	Sì	No
	Sì	Sì	Sì	Si

[6]:
```
!gcc -E -o prepro.txt prepro.c
```

[7]:
```
!gcc -S -o prepro.s prepro.c
```

[8]:
```
!gcc -c -o prepro.o prepro.c
```

1.6 Parole riservate del C già viste insieme

- **char**
- **double**
- **else**
- **float**
- **for**
- **if**
- **int**
- **long**
- **return**
- **short**
- **sizeof**
- **struct**
- **typedef**
- **union**
- **unsigned**
- **void**
- **while**
- **_Packed**

1.7 Parole riservate del C presentate oggi

- **break**
- **case**
- **const**
- **continue**
- **decimal**
- **default**
- **do**
- **enum**
- **extern**
- **goto**
- **static**
- **switch**

1.8 Parole riservate del C che non spiego

- **auto**
- **digitsof**
- **precisionof**
- **register**
- **volatile**

```c
if (c) {

} else {

}
```

```c
switch (c) {
  case 1:
     istruzione1();
     break;
  case 2:
     istruzione2();
     break;
  default:
     istruzioneD();
}
```

```c
while (espressione) {  ...  }

do {  ...  } while (espressione);

for (espressione_iniziale; test; epressione_per_incremento) {  ...  }
```

[18]:
```
!gcc -o casi casi.c
```

[19]:
```
!./casi 4
```

```
Giovedì
4
3
2
1
```

1.9 Uso combinato di struct e puntatori: le liste

Un gruppo importante di strutture dati sono le **strutture lineari dinamiche**, in queste un insieme di elementi viene organizzato in modo da poter distinguere gli elementi in base alla loro **posizione** (primo, secondo, terzo, ecc).

N.B.: In C non si includono gli array in questo gruppo perchè non è possibile farne crescere il numero dinamicamente durante l'esecuzione del programma.

Le **liste** sono sequenze di lunghezza variabile nelle quali è possibile inserire e/o cancellare elementi in qualsiasi posizione.

```c
typedef struct _Cella {
  int numero;
  struct _Cella *puntatore;
} Cella, *pCella;
```

```c
  int main(int argc, char *argv[]) {
    pCella a,b;
```

```
  a = (pCella) malloc(sizeof(Cella));
  a->numero = 1;
  a->puntatore = NULL;
  ///  ho solo una cella
  b = (pCella) malloc(sizeof(Cella));
  b->numero = 2;
  b->puntatore = NULL;
  ///  ho solo due celle disgiunte
  a->puntatore = b;
  ///  ho due celle legate
}
```