Introduzione al Linguaggio C...

...per chi già mastica Java

Francesco Versaci

DEI – Università di Padova

5 maggio 2009

DEPARTMENT OF INFORMATION ENGINEERING

UNIVERSITY OF PADOVA

Sommario

1. Il Linguaggio C
2. Il Compilatore GCC
3. Come fare i Makefile

Il libro sacro. . .

Brian W. Kernighan, Dennis M. Ritchie
Il Linguaggio C (II Edizione)
Pearson – Prentice Hall, 27 €

Un po' di storia...

Dall'ALGOL al C

1999 ISO C99

1989-90 Ansi C (C89, ISO C90)

1978 Esce *The C Programming Language* (K&R)

1969-73 Nasce il C (Ritchie, AT&T Bell Labs)

∼1969 B (Thompson)

1966 BCPL (Richards, University of Cambridge, in visita all'MIT)

1963 CPL (University of Cambridge and London)

1958 ALGOL (AA. VV., ETH Zurigo)

Dennis Ritchie e Ken Thompson – *Turing Award 1983*

Caratteristiche

> *C is a language that combines all the elegance and power of assembly language with all the readability and maintainability of assembly language*
>
> Jargon file

- Curly bracket language (come C++, Java, . . .)
- Molto semplice
- *Facile* da compilare
- Vicino alla macchina
- Ideale per ottenere alte prestazioni
- Programmazione a oggetti non esplicita

Hello, World!

hello.c

```
1  #include <stdio.h>
2
3  main(){
4     printf("hello, world\n");
5  }
```

Compilazione

```
$ cc hello.c
$ ./a.out
hello, world

$ gcc -O3 -o hello hello.c
$ ./hello
hello, world
```

Tipi di dati

ANSI C

- `char` 1 byte
- `int` una word
- `float` numero in virgola mobile in precisione singola
- `double` numero in virgola mobile in precisione doppia
- `unsigned int` intero non negativo

C99

- `int8_t` intero a 8 bit
- `int16_t` intero a 16 bit
- `uint32_t` intero a 32 bit senza segno
- `int64_t` intero a 64 bit
- `bool` booleano
- `complex` numero complesso

Note

- In C i dati non hanno dimensione fissata (per adattarsi alla macchina mantenendo le prestazioni)
- I booleani sono semplici interi (FALSO=0, VERO=diverso da 0)
- Le stringhe sono array di char, terminate da NULL (`\0=0x00`)

Formattazione dell'output: `printf`
Sintassi ed esempi

int printf (**char** *format, arg1, arg2, ...)

- scrive sullo stdout gli argomenti, formattati secondo `format`
- restituisce il numero di caratteri visualizzati

Esempi

```
int x=13; char c[]="cows";
printf("number␣of␣%s:␣%d\n", c, x );
/* number of cows: 13 */
printf("%.3s\n", c);
/* cow */

const double pi=3.141592653589;
printf("%10.5f\n", pi);
/*    3.14159 */
```

CARATTERE	TIPO DELL'ARGOMENTO; STAMPATO COME
d,i	`int`; numero in notazione decimale
x,X	`unsigned int`; numero in notazione esadecimale (senza `0x` in testa)
c	`int`; singolo carattere
s	`char*`; stampa la stringa fino a raggiungere `0x00` o la lunghezza indicata
f	`double`; [-]i.dddddd, con i parte intera e dddddd cifre decimali
g	`double`; stampa i double in notazione esponenziale
e	`double`; sceglie fra f e g a seconda della grandezza del numero
p	`void *`; puntatore

- Un puntatore è una variabile che contiene l'indirizzo di un'altra variabile.
- I puntatori possono essere incrementati per scorrere dei vettori (gli iteratori in C++ sono una generalizzazione dei puntatori)

Esempio

```
int x=10;
int *p=&x;   /* p punta a x */
(*p)++;   /* incrementa x */

double d[]={1.2, 2.3, 3.4};
double* p=d;
p++;
printf("%g\n", *p); /* 2.3 */
```

swap.c – Sbagliato

```
#include <stdio.h>

void swap(int a, int b){
  int c=b;
  b=a; a=c;
}

main(){
  int a=10; int b=20;
  printf("a: %d  b: %d\n",a,b);
  swap(a,b);
  printf("a: %d  b: %d\n",a,b);
}
```

```
$ gcc swap.c && ./a.out
a: 10  b: 20
a: 10  b: 20
```

swap.c – Corretto

```
#include <stdio.h>

void swap(int *a, int *b){
  int c=*b;
  *b=*a; *a=c;
}

main(){
  int a=10; int b=20;
  printf("a: %d  b: %d\n",a,b);
  swap(&a,&b);
  printf("a: %d  b: %d\n",a,b);
}
```

```
$ gcc swap.c && ./a.out
a: 10  b: 20
a: 20  b: 10
```

Puntatori e Vettori

scan.h

```
void v_scan(int x[], int n);
void p_scan(int *p);
```

scan.c

```
#include <stdio.h>

void v_scan(int x[], int n){
  int i=0;
  for(; i<n; ++i){
    int buf=x[i];
    if (buf>900)
      printf("%d\n",buf);
  }
}
void p_scan(int *p){
  int buf;
  while ((buf=*p++)>0)
    if (buf>900)
      printf("%d\n",buf);
}
```

main.c

```
#include <stdlib.h>
#include "scan.h"

#define N 100

int main(){
  int x[N];

  int i=0;
  /* inizializzazione valori */
  for(; i<N-1; ++i)
    x[i]=rand()%1000;
  /* terminatore */
  x[N-1]=-1;

  v_scan(x, N);
  p_scan(x);

  return 0;
}
```

Argomenti della Riga di Comando

args.c

```c
#include <stdio.h>

int main(int argc, char* argv[]){
  int i=0;
  for(; i<argc; ++i)
    printf("Argomento %d: %s\n", i, argv[i]);

  return 0;
}
```

Compilazione ed esecuzione

```
$ cc -o args args.c && ./args foo bar
Argomento 0: ./args
Argomento 1: foo
Argomento 2: bar
```

Strutture

complex.c

```
1  #include <stdio.h>
2
3  typedef struct {
4    double re;
5    double im;
6  } co;
7
8  co prod(co x, co y){
9    co z;
10   z.re=x.re*y.re-x.im*y.im;
11   z.im=x.im*y.re+x.re*y.im;
12   return z;
13 }
14
15 main(){
16   co x,z;
17   x.re=0; x.im=1; /* x=i */
18   z=prod(x,x); /* z=i^2=-1 */
19   printf("(%g + i %g)\n",z.re, z.im);
20 }
```

- Definiscono nuovi tipi composti
- Antesignani delle classi C++

Allocazione Dinamica e I/O

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char* argv[]){
5    int n, i, *dyn; FILE* fout;
6    if (argc!=2)
7      return 1;
8
9    sscanf(argv[1], "%d", &n);
10
11   dyn=malloc(n*sizeof(int));
12
13   for(i=0; i<n; ++i)
14     dyn[i]=rand()%1000;
15
16   fout=fopen("data", "w");
17   fwrite(dyn, sizeof(int), n, fout);
18   fclose(fout);
19
20   free(dyn);
21
22   return 0;
23 }
```

- La funzione `sscanf` legge i valori da una stringa
- `malloc` alloca dinamicamente un'area di memoria
- `free` la libera
- `fopen` apre un file
- `fclose` lo chiude
- `fwrite` scrive sul file

Ricevere un messaggio

```
int MPI_Irecv(void* buf, int count, \
MPI_Datatype datatype, int source, \
int tag, MPI_Comm comm, MPI_Request *request)
```

- Inizializza il processo di ricezione (recv posting)
- Ritorna quando il richiesta di ricezione è stata registrata
- `buf` non puo' essere usato finche' la richiesta è pendente
- Occorre controllare lo stato della richiesta di ricezione
- Può corrispondere a una spedizione bloccante

Comunicazioni punto-punto
Nearest neighbor exchange in ring topology

```
int numtasks, rank, next, prev, buf[2], tag1=1, tag2=2;
MPI_Request reqs[4];
MPI_Status stats[4];

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

prev = (rank-1)%numtasks;
next = (rank+1)%numtasks;

MPI_Irecv(&buf[0], 1, MPI_INT, prev, tag1, MPI_COMM_WORLD, &reqs[0]);
MPI_Irecv(&buf[1], 1, MPI_INT, next, tag2, MPI_COMM_WORLD, &reqs[1]);
MPI_Isend(&rank, 1, MPI_INT, prev, tag2, MPI_COMM_WORLD, &reqs[2]);
MPI_Isend(&rank, 1, MPI_INT, next, tag1, MPI_COMM_WORLD, &reqs[3]);

{  do some work  }

MPI_Waitall(4, reqs, stats);

MPI_Finalize();
```

Sommario

1 Il Linguaggio C

2 Il Compilatore GCC

3 Come fare i Makefile

Passi per la creazione dell'eseguibile

preprocessing Vengono espanse le macro (fra cui le direttive `#include`)

compilation Il sorgente viene convertito in linguaggio *assembly*

assembling Dall'*assembly* al linguaggio macchina

linking Si uniscono i vari pezzi per formare l'eseguibile finale

runtime Vengono caricate in memoria le eventuali librerie condivise

Estensioni riconosciute

Le estensioni riconosciute dal compilatore come file C sono le seguenti:

header `.h`

codice `.c`

Opzioni per l'output

`gcc [-c] [-o output] nome.c`

-c Compila senza collegare, non crea l'eseguibile

-o *nomeoutput* Chiama il file di uscita "nomeoutput"

scan.h

```
void v_scan(int x[], int n);
void p_scan(int *p);
```

scan.c

```
#include <stdio.h>

void v_scan(int x[], int n){
  int i=0;
  for(; i<n; ++i){
    int buf=x[i];
    if (buf>900)
      printf("%d\n",buf);
  }
}
void p_scan(int *p){
  int buf;
  while ((buf=*p++)>0)
    if (buf>900)
      printf("%d\n",buf);
}
```

main.c

```
#include <stdlib.h>
#include "scan.h"

#define N 100

int main(){
  int x[N];

  int i=0;
  /* inizializzazione valori */
  for(; i<N-1; ++i)
    x[i]=rand()%1000;
  /* terminatore */
  x[N-1]=-1;

  v_scan(x, N);
  p_scan(x);

  return 0;
}
```

```
$ gcc -c scan.c
$ gcc -c main.c
$ gcc -o runme main.o scan.o
$ ./runme
[output]
```

Spiegazione

- I primi due comandi compilano i file sorgenti creando i file oggetto `.o`.
- Il terzo li collega creando l'eseguibile `runme`, lanciato alla quarta riga.

Le prime tre righe si sarebbero potute sostituire con questa:

```
gcc -o runme scan.c main.c
```

Queste opzioni segnalano in fase di compilazione delle probabile sviste del programmatore che non danno errori in compilazione.
È consigliabile usarle sempre.

Warnings

-Wall Abilita molti avvisi utili (variabili usate senza inizializzazione, classi polimorfe senza distruttore virtuale, ecc.)

–ansi Seleziona l'ANSI C

–pedantic Avvisa se si devia dallo standard ISO

-W (-Wextra) Abilita altri avvisi (confronto di unsigned con 0, funzioni che possono o meno restituire un valore, ecc.)

Il gcc può utilizzare diverse tecniche per ottimizzare l'eseguibile prodotto.

Opzioni per l'ottimizzazione

- -01 Ottimizza salti condizionali e cicli, prova a eliminare alcuni if, srotola i cicli, ...
- -02 Ottimizza l'uso di sottoespressioni, riordina il codice minimizzando i salti, ...
- -03 Rende inline le funzioni piccole, ...

Opzioni di linking

-l*nome* Usa la libreria esterna “nome”

-L*dir* Cerca le librerie nelle directory “dir”

-shared Crea un file oggetto condiviso, da usare come libreria dinamica

-static Collega tutte le librerie in modo statico (anche le condivise)

-s Togli la tavola dei simboli dall’eseguibile

pi.c

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

int main(){
   printf("pi=%.10f\n", 4*
       atan(1));

   return 0;
}
```

Compilazione

```
$ gcc -o runme pi.c -lm
$ ./runme
pi=3.1415926536
```

Per usare una libreria esterna è necessario installare il pacchetto con gli header (in Linux *libqualcosa-dev*), includere l'header e dichiarare al linker in quali librerie recuperare gli oggetti (si veda il manuale della libreria usata).

Sommario

1 Il Linguaggio C

2 Il Compilatore GCC

3 Come fare i Makefile

Per automatizzare la compilazione dei sorgenti lo strumento standard è il `make`. Noi analizzeremo la versione GNU di questo programma.

Il `Makefile`

In ogni directory che contiene sorgenti è necessario creare un file di testo, chiamato *Makefile*, che contiene le istruzioni per la compilazione.

Come avviare la compilazione

Una volta creato il Makefile è sufficiente lanciare il comando `make` (o `make obiettivo`) per avviare la compilazione.

Sintassi delle regole

```
obiettivo: prerequisiti
        comando1
        comando2
        ...
```

Spiegazione

obiettivo Il file da creare

prerequisiti I file che servono per crearlo

comando I comandi (si noti il *tab* obbligatorio) per creare l'*obiettivo*

Cosa fa il make?

Controlla se qualcuno frai *prerequisiti* è stato modificato piú recentemente dell'*obiettivo*. In caso affermativo esegue i *comandi*.

Makefile

```
grosso.txt:  piccolo1.txt piccolo2.txt
        cat piccolo1.txt piccolo2.txt > grosso.txt
```

Il comando `make grosso.txt` controlla che il file *grosso.txt* non esista oppure sia piú vecchio dei file *piccolo1.txt* e *piccolo2.txt*.
Quindi esegue il comando *cat* che crea il file *grosso.txt*.
Nel caso l'obiettivo sia già aggiornato make notifica la cosa con la seguente stringa:

```
make: 'grosso.txt' is up to date.
```

Spesso è comodo poter creare dei comandi svincolati da particolari file. Si usano allora degli obiettivi *phony*.

Makefile

```
.PHONY: clean
clean:
        rm *.o
```

Spiegazione

Eseguire il comando `make clean` è equivalente ad eseguire `rm *.o`

Nei Makefile è possibile utilizzare delle variabili per evitare di riscrivere piú volte gli stessi comandi.

Makefile

```
CFLAGS = -Wall -W -pedantic -ansi

scan.o : scan.c
        gcc $(CFLAGS) -c scan.c

main.o : main.c
        gcc $(CFLAGS) -c main.c
```

Spesso si deve eseguire lo stesso comando su piú file.

Makefile

```
CFLAGS = -Wall -W -pedantic -ansi
%.o : %.c
        gcc $(CFLAGS) -c -o $@ $<
```

Regole implicite

La seconda riga fornisce una regola per trasformare qualunque file `.c` in un file `.o`

Variabili automatiche

- `$@` Bersaglio
- `$<` Primo prerequisito
- `$^` Tutti i prerequisiti

```
CXXFLAGS = -Wall -W -pedantic

%.o : %.cc
        g++ $(CXXFLAGS) -o $@ -c $<

runme :  test.o saluti.o
        g++ $(CXXFLAGS) -o $@ $^
```

EOF

Buona programmazione e buona serata. . .