DIPARTIMENTO
DI INGEGNERIA
DII DELL' INFORMAZIONE

UNIVERSITÀ DEGLI STUDI DI PARMA

Lezione integrativa: Dal C++ al C

Mario Sabbatelli
smario@ce.unipr.it
Palazzina 3 - int. 5792
www.ce.unipr.it/people/smario

Contenuto della lezione

- Nel corso si utilizza l'ANSI C
  - Storia e standard
- panoramica differenze C/C++
  - non esaustiva
- differenze di cui tener conto
  - meno significative
    - commenti
    - funzioni
    - dati
    - passaggi a funzione
    - funzione tipo
  - gestione memoria
  - gestione I/O

C vs C++

- C++ evoluzione del C
  - orientato agli oggetti
  - codice piú leggibile e manutenibile
  - piú utilizzato in sw complessi
  - maggiore difficoltà di programmazione
- C
  - efficiente
  - consolidato
  - piú utilizzato per: driver, programmi semplici, sistemi operativi
  - basso/medio livello

standard C

- il C nasce nel 1969-1973
- differenti standard
  - K&R (da evitare)
  - ANSI C o ISO C:
    - C89/C90 largo supporto
    - C99 diffusione recente
    - C1X lavori in corso (dal 2007)
- Code::Blocks > C89/C90

Compilazione

- Linguaggio C è un linguaggio compilato
- programma che traduce una serie di istruzioni scritte in un determinato linguaggio di programmazione (codice sorgente) in istruzioni di un altro linguaggio (codice macchina)
- Schema compilazione C/C++

Il preprocessore

- Direttive che iniziano per #
- #define
  - Macro e costanti
  - #define ELEMENTS 100
    - Sostituisce in tutto il programma 100 alla parola ELEMENTS
- #include
  - Incorpora quanto incluso nel sorgente

Il compilatore

- Converte il sorgente in linguaggio macchina
  - Oggetti
  - Progetti multi file
  - Riferimenti/indirizzi non completi

linker

- Unisce i file oggetto
  - Risolve indirizzi/riferimenti
  - Associa eventuali librerie esterne
- Risultato: eseguibile

Scheletro di un sorgente C

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

int main(int argc, char **argv){
    /* stampa Hello World! A schermo */
    printf("Hello World!");
    return 0;
}
```

Header file

- In C hanno sempre l'estensione `.h`
- Buona parte utilizzati anche in C++ con c iniziale
- Principali
  - **`stdio.h`** (obbligatorio)
  - **`stdlib.h`** (altamente consigliato)
  - **`math.h`** (funzioni matematiche)
  - **`string.h`** (funzioni di stringa)
  - **`time.h`** (gestione date e tempo)

commenti

- la maggioranza dei compilatori (C89) supporta solo /* ... */
- C99 e Code::Blocks anche //
- problemi di portabilità

funzioni

- In C sono i mattoncini principali
- Differenze con il C++
  - valori predefiniti non possibili
  - non esiste overloading
  - solo passaggio per copia o indirizzo
    - non esistono riferimenti
  - C89: definizione implicita funzioni

Variabili in C

Una variabile è uno spazio di memoria con nome

- Dimensioni
- Posizione

- Scalari
  - Interi
    - char
    - int
    - long
    - short
  - Virgola mobile
    - float
    - double
  - Indirizzi
    - puntatori
- Composite
  - struct
  - union
  - Enum
- Altro
  - void

Definizione variabili

- C89: definibili solo all'inizio di un blocco di codice prima di qualunque istruzione
- C89: non definibili entro `for()`

```
int main(){
   double a;
   a=3.14567;
   char b;
   for(int i=0;i<10;++i)
      …
}
```

tipo di dato bool

- C89 non esiste
  - si usa un qualunque altro tipo di dato intero (anche `char`)
- C99
  - possibile includere `stdbool.h`

riferimenti

- non esistono
- alternativa → puntatori
  - sintassi piú complessa

```
void half3(double *x){
    *x = *x + 3;
    *x /= 2;
}
…
double myvalue=3.14;
half3(&myvalue);
…
```

oggetti

- non esistono oggetti
  - niente costrutto **`class`**
  - esiste **`struct`** ma senza metodi
- conseguenze
  - no **`iostream`** → funzioni specifiche
  - no **`vector`** → array o altri costrutti
  - no **`string`** → array di char
  - ...

Gestione memoria in C

- Non esistono `new[]` e `delete[]`
- Costrutti equivalenti (ANSI C):
  - `void *malloc(size_t size);`
  - `void free(void *ptr);`
  - `void *calloc(size_t nmemb, size_t size);`
  - `void *realloc(void *ptr, size_t size);`

Esempio di allocazione in C

```
#include<stdlib.h>
int *p;
…
p=malloc(20*sizeof(int))// array di 20 int
if(!p){
    // errore di allocazione
}
…
free(p); // rilascio memoria
```

- la `malloc()` non inizializza la memoria
  - `memset(p,0,20*sizeof(int));`
- non necessario cast → `int*` (K&R e C++)

Allocazione con zeri

```
#include<stdlib.h>
int *p;
…
p=calloc(20,sizeof(int))// array di 20 int
if(!p){
    // errore di allocazione
}
…
free(p); // rilascio memoria
```

- la `calloc()` inizializza i dati a 0
- numero elementi e loro dimensione in byte

I/O console in C

- Principali funzioni:
  - `int printf(const char *format, ...);`
  - `int scanf(const char *format, ...);`
  - `char *gets(char *s);`
  - `int getchar(void);`

int printf(const char *format, ...);

```
printf("Hello World!\n");
printf("Il risultato di %d/%d è %2.3f\n",
            a,b,((float)a)/b);
```

- stringa di formato:
  - testo
  - sequenze di escape
  - specificatori di formato
    - caratteri di controllo
- dati da utilizzare in base alla stringa

sequenze di escape

- corrispondono a caratteri “non stampabili” o ad uso specifico
- elenco non esaustivo:

`\n`	newline
`\r`	carriage return
`\f`	form feed
`\a`	bell
`\t`	horizontal tab
`\v`	vertical tab
`\”`	doppie virgolette
`\nnn`	carattere corrispondente all'ASCII nnn (ottale)
`\xNN`	carattere corrispondente all'ASCII NN (esadecimale)
`\\`	backslash
`%%`	singolo %

specificatori di formato

- preceduti da %
- elenco non esaustivo

d	intero decimale
o x X	intero ottale o esadecimale
f	numero a virgola mobile in notazione decimale
e E	numero a virgola mobile in notazione scientifica
g G	numero a virgola mobile (automatico)
c	singolo carattere
s	stringa (C)
p	indirizzo di memoria (void *)
q L	long long (C99)
[]	elenco di caratteri, solo **scanf()**

caratteri di controllo

- anteposti allo specificatore di conversione
- elenco non esaustivo

+	stampa comunque il segno
spazio	antepone a numeri positivi uno spazio
`n`	minima ampiezza campo (riempimento con spazi)
`0n`	minima ampiezza campo (riempimento con 0)
`.n`	numero di cifre decimali (minimo o fisso)
–	allineamento a sinistra

int getchar(void); char *gets(char *s);

- getchar() lettura di un singolo carattere
- gets() lettura di una stringa e terminazione

```
#include<stdio.h>
char sc,str[100];
…
sc=getchar(); // bloccante
…
gets(str); // no limite e no controllo
           // se fallisce restituisce NULL
```

int scanf(const char *format, ...);

- legge input in accordo a stringa di formato
- simile a **printf()**

```
#include<stdio.h>
char str[100];
int a,n;

scanf("%d", &a); // specificare indirizzo
scanf("%s", str); // si ferma al primo "spazio"
scanf("%[^\n]", str); // legge fino a "\n"
n=scanf("%s %d",str,&a); // n=elementi letti
scanf("n: %d",&a); // input problematico!
```

I/O su file

- struttura dati specifica: FILE
- principali funzioni:
  - `FILE *fopen(const char *path, const char *mode);`
  - `int fprintf(FILE *stream, const char *format, ...);`
  - `int fscanf(FILE *stream, const char *format, ...);`
  - `size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);`
  - `size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);`
  - `int fclose(FILE *fp);`

Esempio (file ASCII)

```
FILE *fp;


// apre il file in scrittura
fp=fopen("z:pippo.txt", "w");
if(!fp) exit(1) // errore!
…
fprintf(fp, "Dato %d\n", 33);
…
fclose(fp);
```

Modalità di apertura file

`r`	lettura
`w`	azzeramento e scrittura
`a`	scrittura in aggiunta
`r+`	lettura e scrittura
`w+`	azzeramento, lettura e scrittura
`a+`	lettura e scrittura in aggiunta
`b`	in aggiunta ai precedenti: file binario

Esempio file binario

```
FILE *fp;
 // apre il file in lettura binaria
fp=fopen("c:\\somep0rn.jpg", "rb");
if(!fp) exit(1) // errore!
int jpeghead[100];
 // leggo i primi 4*100 byte
fread(jpeghead,4,100,fp);
…
fclose(fp);
```

stream predefiniti

- esistono 3 stream predefiniti:
  - `stdin` (standard input)
  - `stdout` (standard output)
  - `stderr` (standard output)
- sconsigliato `fclose()`

```
fprintf(stderr,"Errore: %d\n",a);
fscanf(stdin,"%s",miastringa);
fgets(miastringa,80,stdin); // da studiare
```

Le Stringhe

- non esiste la classe **`string`**
- le stringhe si realizzano con array
- stringa = array di **`char`** che contiene una sequenza di **`char`** terminata da un carattere nullo (**`'\0'`**)
  - lunghezza stringa <= lunghezza array-1

C	i	a	o	\0	?	?

Principali funzioni di stringa

- occorre includere **string.h**
- si basano tutte sull'esistenza di **'\0'**

```
#include<string.h>
size_t strlen(const char *s);
char *strcpy(char *dest, const char *src);
int strcmp(const char *s1, const char *s2);

char corso[]="Fondamenti di Programmazione";
char buf[2000];

strcpy(buf,corso);
printf("La lunghezza di %s e' %d\n",buf,strlen(buf));
if(!strcmp(corso,buf))
   printf("le due stringhe sono uguali!\n");
```

DIPARTIMENTO
DI INGEGNERIA
DII DELL' INFORMAZIONE

UNIVERSITÀ DEGLI STUDI DI PARMA

Lezione integrativa: Dal C++ al C

Mario Sabbatelli
smario@ce.unipr.it
Palazzina 3 - int. 5792
www.ce.unipr.it/people/smario