Alcuni Richiami sul C

Maurizio Palesi
DIIT—Università di Catania
Viale Andrea Doria 6, 95125 Catania
mpalesi@diit.unict.it
http://www.diit.unict.it/users/mpalesi

Sommario

In questo documento saranno richiamati alcuni degli argomenti fondamentali trattati nel corso di Fondamenti di Informatica. Tali argomenti costituiscono il punto base del corso di Laboratorio di Informatica e lo studente ne deve avere un'assoluta padronanza.

Indice

Capitolo 1

Introduzione

Il corso di Fondamenti di Informatica è stato rivolto allo studio dei fondamenti della programmazione utilizzando il linguaggio C come linguaggio di programmazione di riferimento. In questo corso si continuerà ad utilizzare il linguaggio di programmazione C e suppone che ogni studente abbia una perfetta padronanza dei seguenti argomenti:

- Tipi di dato (semplici e strutturati).
- Costrutti base (if, for, while, switch).
- Manipolazione delle stringhe.
- Puntatori.
- Funzioni e passaggio dei parametri.

Nei capitoli successivi faremo un breve richiamo per ognuno dei suddetti argomenti.

Capitolo 2

Tipi di Dato in C

Possiamo dividere i tipi di dato in C in *semplici* e *complessi*. I tipi di dato complessi sono un'aggregazione di dati di tipo semplice.

2.1 Tipi Semplici

La Tabella 2.1 riassume alcuni dei tipi di dato semplici messi a disposizione dal linguaggio C insieme alla loro rappresentazione e insieme di variazione[1]. La definizione (o dichiarazione)

Tipo	bits	min	max
char	8 in C2	-128	+127
unsigned char	8	0	255
int	16 in C2	-32768	32767
unsigned int	16	0	65535
float	16 SPFP		
double	32 DPFP		

Tabella 2.1: Tipi di dato semplici in C.

di una variabile in C si effettua specificando il tipo ed il nome della variabile:

```
<tipo_di_dato> <nome_della_variabile>;
```

Per esempio le seguenti sono tutte dichiarazioni di variabili corrette:

[1]Tali valori dipendono dal compilatore utilizzato.

```
int             a, b=7, c=-44;
unsigned char   q=21;
char            r='A';
float           f=12.5;
```

2.2 Tipi Complessi

Tra i tipi di dati complessi in C richiamiamo i vettori (o array) e le strutture (o struct).

Vettori

Un vettore è un insieme ordinato di dati tutti dello stesso tipo indirizzabile in modo diretto. Un vettore si definisce specificando il tipo base di ogni elemento e il numero maasimo di elementi che può memorizzare:

```
<tipo_di_dato> <nome_del_vettore>[<numero_di_elementi>];
```

Per esempio la seguente dichiarazione:

```
int v[10];
```

definisce un vettore di 10 interi. Il generico elemento i-esimo del vettore `v` è `v[i]`. Il primo elemento di `v` ha indice 0, l'ultimo ha indice 9. Se per esempio volessimo inizializzare il vettore `v` con tutti zeri basta fare:

```
{
  int i, v[10];
  for (i=0; i<10; i++)
    v[i] = 0;
}
```

Strutture

A differenza dei vettori, le strutture, consentono di aggregare dati di tipo diverso. Una struttura si definisce specificando la parola chiave `struct` e la lista dei *campi* che la compongono insieme ai loro tipi:

```
struct <nome_struttura> {
   <tipo1> <campo1>;
   <tipo2> <campo2>;
   ...
   <tipoN> <campoN>;
};
```

Per esempio la seguente struttura definisce il tipo di dato *Persona*:

```
struct Persona {
    char nome[16];
    char cognome[16];
    int  giorno, mese, anno;
};
```

Per accedere al generico campo della struttura basta specificare il nome della variabile struttura e del campo separati da un punto. Per esempio le seguenti istruzioni:

```
struct Persona persona;
persona.giorno = 10;
persona.mese = 6;
persona.anno = 1974;
```

dichiarano una variabile `persona` di tipo `struct Persona` ed inizializzano i campi `giorno`, `mese` ed `anno` rispettivamente a 10, 6, 1974. Il campo di una struttura può anch'esso essere una struttura. Per esempio:

```
struct Data {
    int  giorno, mese, anno;
};

struct Persona {
    char        nome[16];
    char        cognome[16];
    struct Data data;
};
```

Per accedere ad un campo struttura di una struttura basta percorrere la stessa in modo gerarchico:

```
struct Persona persona;
persona.data.giorno = 10;
persona.data.mese = 6;
persona.data.anno = 1974;
```

Capitolo 3

Costrutti Base

3.1 Sequenza

Una sequenza di istruzioni è racchiusa sempre tra parentesi graffe {}.
Per esempio: leggere e sommare due numeri

```
{ int a, b, somma;

  scanf(''%d%'',&a);
  scanf(''%d'', &b);
  somma = a + b;
  printf(''%d'', somma);
}
```

3.2 Istruzione `if-else`

Questa istruzione serve a indicare quale azione intraprendere sulla base del valore di una certa condizione. La forma generale dell'istruzione `if` è la seguente:

```
if (expressione)
  istruzione_if;
else
  istruzione_else;
```

dove `istruzione_if` e `istruzione_else` possono essere istruzioni singole, sequenze di istruzioni o nulle. La clausola `else` può essere opzionale. Se l'espressione fornisce un risultato vero viene eseguita l'`istruzione_if`; altrimenti, se la clausola `else` esiste, viene eseguita l'`istruzione_else`.

Esempi `if-else`

```
/*************************************
 Versione 1
*************************************/
 {
   int a, b, differenza, va;

   scanf(''%d'', &a);
   scanf(''%d'', &b);
   differenza = a - b;

   if (differenza > 0)
     va = differenza;
   else
     va = -differenza;

   printf(''%d'', va);
 }

/*************************************
 Versione 2
*************************************/
 {
   int a, b, differenza;

   scanf(''%d'', &a);
   scanf(''%d'', &b);
   differenza = a - b;

   if (differenza > 0)
     printf(''%d'', differenza);
   else
     printf(''%d'', -differenza);
 }


/*************************************
 Versione 1
```

```
**************************************/
 {
   int a, b, max;

   scanf(‘‘%d’’, &a);
   scanf(‘‘%d’’, &b);
   max = a;
   if (b > a)
     max = b;
   printf(‘‘%d’’, max);
 }

/**************************************
 Versione 2
**************************************/
 {
   int a, b, max;

   scanf(‘‘%d’’, &a);
   scanf(‘‘%d’’, &b);
   if(b > a)
     max = b;
   else
     max = a;
   printf(‘‘%d’’, max);
 }


/**************************************
 Versione 1
**************************************/
 {
   int a, val_assoluto;

   scanf(‘‘%d’’, &a);
   if (a > 0)
     val_assoluto = a;
   else
```

```
    val_assoluto = -a;
  printf(''%d'', val_assoluto);
}

/*************************************
 Versione 2
*************************************/
{
  int a;

  scanf(''%d'', &a);
  if (a < 0)
    a = -a;
  printf(''%d'', a);
}


{
  int  x, y;
  char operazione;

  printf("Inserire i due operandi");
  scanf(''%d%d'', &x, &y);
  printf("Quale operazione ? (+ o -)");
  scanf(''%c'', &operazione);

  if (operazione == '+')
    risultato = x + y;
  else
    risultato = x - y;
  printf(''%d'', risultato);
}


{
  int a, b;

  scanf(''%d%d'', &a, &b);
```

```
  if (a < 0 && b < 0)
    {
      qa = a * a;
      qb = b * b;
      somma = qa + qb;
    }
  else
    somma = a + b;
  printf(‘‘%d’’, somma);
}


{
  int  x, y;
  char operazione;

  printf("Inserire i due operandi");
  scanf(‘‘%d %d’’, &x, &y);
  printf("Quale operazione (+ – * / )");
  scanf(‘‘%d’’, &operazione);

  if (operazione == ’+’)
    risultato = x + y;
  else if (operazione == ’–’)
    risultato = x – y;
  else if (operazione == ’*’)
    risultato = x * y;
  else if (y != 0)
    risultato = x / y;

  if (operazione == ’/’ && y == 0)
    printf("Divisione per zero");
  else
    printf(‘‘%d’’, risultato);
}
```

3.3 Istruzione `while`

Questa istruzione permette la ripetizione di un'azione fino a quando una certa condizione è vera. Non appena la condizione diventa falsa, viene eseguita l'istruzione successiva al `while`. La forma generale dell'istruzione `while` è la seguente:

```
while (condizione)
  istruzione;
```

dove `istruzione` può essere un'istruzione vuota, una singola istruzione o un blocco di istruzioni. L'istruzione eseguita deve essere tale che dopo un numero finito di volte la condizione diventi falsa.

Nel seguente esempio:

```
{
 int i=0;
 while (i<10)
  printf(''%d'', i);
}
```

poichè il valore di `i` non viene modificato, il ciclo verrebbe eseguito all'infinito.

Esempi `while`

```
{
   int contatore=0, numero;

   while (contatore < 10)
     {
       scanf(''%d'', &numero);
       contatore = contatore + 1;
     }
}
```

```
{
   int contatore=0, somma=0, numero;
   while (contatore < 100)
     {
```

```
        scanf(‘‘%d’’, &numero);
        somma = somma + numero;
        contatore = contatore + 1;
      }

   printf(somma);
}


{
   int i=0, somma=0, N;

   scanf(‘‘%d’’, &N);
   while (i < N)
     {
       somma = somma + i;
       i = i + 1;
     }

   printf(‘‘%d’’, somma);
}


{
   int contatore=0, somma=0, numero;

   scanf(‘‘%d’’, &N);
   while (contatore < N)
     {
       scanf(‘‘%d’’, &numero);
       somma = somma + numero;
       contatore = contatore + 1;
     }

   printf(‘‘%d’’, somma);
}
```

```
{
  int numero, massimo, contatore;

  scanf(‘‘%d’’, &numero);
  massimo = numero;
  contatore = 1;

  while (contatore < 10)
    {
      scanf(‘‘%d’’, &numero);
      if(numero > massimo)
        massimo = numero;

      contatore = contatore + 1;
    }
  printf(‘‘%d’’, massimo);
}
```

```
{
  int numero, minimo;

  printf("Inserire il primo numero");
  scanf(‘‘%d’’, &numero);
  if (numero > 0)
    {
      minimo = numero;
      while (numero >= 0)
        {
          printf("Inserire un altro numero");
          scanf(‘‘%d’’, &numero);
          if (numero > 0 && numero < minimo)
            minimo = numero;
        }
      printf(‘‘%d’’, minimo);
    }
  else
    printf("Il primo numero non e’ positivo");
```

```
}


 {
    int a, b, mcma, mcmb;

    scanf(''%d'', &a);
    scanf(''%d'', &b);

    mcma = a;
    mcmb = b;

    while (mcma != mcmb)
       if (mcma < mcmb)
         mcma = mcma + a;
       else
         mcmb = mcmb + b;

    printf(''%d'', mcma);
}
```

3.4 Istruzione `do-while`

Questa istruzione permette la ripetizione di un'azione fino a quando una certa condizione è vera. Non appena la condizione diventa falsa, viene eseguita l'istruzione successiva al `do-while`. La forma generale dell'istruzione `do-while` è la seguente:

```
do
  istruzione;
while (condizione);
```

dove istruzione può essere un'istruzione vuota, una singola istruzione o un blocco di istruzioni. Mentre nel ciclo while la condizione viene vaulata prima dell'esecuzione dell'istruzione, nel ciclo `do-while` la condizione viene valutata dopo l'esecuzione dell'istruzione. Così come per il ciclo `while`, bisogna stare attenti alla condizione da testare in modo da evitare che il ciclo venga eseguito all'infinito. L'istruzione o la sequenza di istruzioni deve essere tale che entro un numero finito di passi la condizione diventi falsa.

Esempi do-while

```
{
 int contatore=0, somma=0;
 do
 {
   scanf(‘‘%d’’, &numero);
   somma = somma + numero;
   contatore = contatore + 1;
 }
 while (contatore < 100)
 printf(‘‘%d’’, somma);
}
```

```
{
  int max, numero;

  scanf(‘‘%d’’, &max);
  if (max == 0)
    printf("E’ stato inserito uno zero");
  else {
    do {
      scanf(‘‘%d’’, &numero);
      if (numero != 0)
        {
          if (numero > max)
            max = numero;
        }
      else
        printf("Fine della sequenza");
    } while (numero != 0);
    printf(‘‘%d’’, max);
  }
}
```

```
/*************************************
```

```
 Versione 1
****************************************/
{
  int conta = 0, max = 0;
  do {
    do
      scanf(‘‘%d’’, &numero);
    while (numero <= 0);

    if (numero > max)
      max = numero;

    conta = conta + 1;
  } while (conta < 20);
  printf(‘‘%d’’, max);
}
/****************************************
 Versione 2
****************************************/
{
  int conta=0, max=0;
  do {
    if (numero > 0)
      {
        if (numero > max)
          max = numero;
        conta = conta + 1;
      }
  } while (conta < 20);
  printf(‘‘%d’’, max);
}
```

3.5 Ciclo `for`

La forma generale dell’istruzione for è la seguente:

```
for (inizializzazione; condizione; incremento)
```

```
istruzione;
```

Il ciclo for viene eseguito fino a quando la condizione è vera. L'inizializzazione normalmente è una istruzione di assegnamento di una variabile di controllo usata nella condizione; Nella sezione incremento viene definito come deve variare la variabile di controllo.

Esempi `for`

```
{
  int somma=0, indice, numero;

  /* inizializzazione; condizione; incremento */
  for (indice=0; indice<10; indice++)
    {
      printf("Inserire un numero\n");
      scanf(''%d'', &numero);
      somma = somma + numero;
    }
}
```

```
{
  int numero, fattoriale, indice;

  scanf(''%d'', &numero);
  fattoriale = 1;
  indice = 0;
  for(indice=2; indice<=numero; indice++)
    fattoriale = fattoriale*indice;
}
```

```
{
  int max = 0, conta, numero;

  for(conta=0; conta<20; conta++)
    {
```

```
        do
          scanf(‘‘%d’’, &numero);
        while (numero <= 0);

        if (numero > max)
          max = numero;
      }
    printf(‘‘%d’’, max);
}


 {
    int indice, A[10];
    for(indice=0; indice<10; indice++)
      scanf(‘‘%d’’, &A[indice]);

    for(indice=9; indice>0; indice--)
      printf(‘‘%d’’, A[indice]);
 }


{
  int i, imin, A[10];

  for(i=0; i<10; i++)
    scanf(‘‘%d’’, &A[i]);

  imin=0;

  for(i=1; i<10; i++)
    if(A[i] < A[imin])
      imin = i;

  printf(‘‘%d’’, A[imin]);
  printf(‘‘%d’’, imin);
}
```

3.6 Istruzione `switch`

Questa istruzione serve per selezionare un'operazione tra diverse alternative. La forma generale dell'istruzione switch è la seguente:

```
 switch (espressione)
   {
   case valore1:
     seq_istruzioni_1;
     break;
   case valore2:
     seq_istruzioni_2;
     break;
     ...
   case valoreN:
     seq_istruzioni_N;
     break;
   default:
     seq_istruzioni_N;
}
```

Il valore dell'espressione viene confrontato, nell'ordine, con i valori specificate dal case.

Quando viene trovata una corrispondenza, vengono eseguite tutte le istruzioni fino al successivo break o alla fine dello `switch`.

Esempi `switch`

```
do {
  int  x, y, risultato;
  char operazione, risposta;

  printf("Inserire il tipo di operazione (+,-,*,/): ");
  scanf(&operazione);
  printf("Inserire il primo operando");
  scanf(&x);
  printf("Inserire il secondo operando");
  scanf(&y);

  switch (operazione)
```

```
    {
    case '+' :
      risultato = x + y;
      break;
    case '-' :
      risultato = x - y;
      break;
    case '*' :
      risultato = x * y;
      break;
    case '+' :
      if (b!=0)
        risultato = x / y;
      break;
    default:
      printf("Operazione non valida");
    }

  if (operazione == '/' && y == 0)
    printf("Divisione per zero");
  else
    printf(``%d'', risultato);

  do {
    printf("Continuare ? (s/n) ");
    scanf("%c", &risposta);
  } while (risposta != 's' && risposta != 'n');
} while (risposta = 's');
```

3.7 Istruzione `break`

Questa istruzione è utilizzata per interrompere l'esecuzione

- di un ciclo while, do-while o for;
- dell'istruzione break;

Nel seguente esempio.

```
{
 int somma=0, i;

 for(i=0; i<10; i++)
  {
   scanf(''%d'', &numero);
   if (numero < 0)
     break;
   somma = somma + numero;
  }
 printf(''%d'', somma);
}
```

l'esecuzione del ciclo for ha temine se vengono letti 10 numero oppure se viene letto un numero negativo.

3.8 Istruzione `continue`

Questa istruzione viene usata per interrompere l'esecuzione della corrente iterazione di un ciclo per passare a quella successiva. Tale istruzione può essere inserita solo in un ciclo while, do-while o for; Nel seguente esempio.

```
{
  int somma=0, conta=0;

  for(i=0; i<10; i++)
    {
      scanf(''%d'', &numero);
      if (numero <= 0)
        continue;
      somma = somma + numero;
      conta++;
    }
  media = somma / conta;
  printf(''%d'', somma);
}
```

vengono letti 10 numeri ma vengono sommati solo quelli positivi. Se la condizione (numero <= 0) è vera, le istruzioni successive all'interno del ciclo for non vengono eseguite.

Capitolo 4

Stringhe

Una stringa è un vettore di caratteri il cui ultimo elemento è un carattere terminatore (o di fine stringa) codificato dal carattere di codice 0 e rappresentato in C dal carattere'$\backslash 0$'.

4.1 Rappresentazione in Memoria

Il vettore di caratteri che rappresenta la stringa sarà perciò formato da un numero di elementi pari al numero di caratteri della stringa più uno (il carattere di fine stringa).

Per esempio La stringa "PLUTO" è rappresentata in C da un vettore di 6 caratteri: 5 caratteri per memorizzare PLUTO più il carattere di terminazione:

P	L	U	T	O	$\backslash 0$

4.2 Dichiarazione

In C per dichiarare una stringa in grado di contenere N caratteri occorre dichiarare un vettore di N+1 caratteri. Se per esempio si volesse dichiarare una variabile stringa atta a contenere un codice fiscale occorre dichiarare un vettore di 17 caratteri:

- 16 caratteri per memorizzare i caratteri e le cifre del codice fiscale;
- 1 carattere per il terminatore di stringa ('$\backslash 0$')

```
char codice_fiscale[17];
```

Per dichiarare una stringa inizializzata con certi caratteri (una parola, una frase, un codice ecc.) basta specificare tra doppi apici il contenuto della stringa omettendo il carattere di terminazione. Sarà il compilatore ad aggiungere il carattere di terminazione e a calcolare la dimensione del vettore.

```
char nome_e_cognome[] = "Giuseppe Rossi";
```

A seguito di questa dichiarazione sarà allocato in memoria un vettore di 15 byte (1 byte per ogni carattere): 14 caratteri per contenere "Giuseppe Rossi" 1 carattere di fine stringa ('\0').

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
G	i	u	s	e	p	p	e		R	o	s	s	i	\0

4.3 Lettura e Scrittura di Variabili Stringa

L'identificatore di formato utilizzato dalle funzioni di lettura (scanf) e scrittura (printf) per manipolare le stringhe è il %s.

Leggere una stringa da tastiera e stamparla a video

```
/* dichiarazione della variabile s */
char s[20];
/* invito l'utente a scrivere qualcosa */
printf("Inserisci una stringa: ");
/* leggo la stringa inserita in s */
scanf("%s", s);
/* visualizzo la stringa s */
printf("Hai inserito la stringa: %s", s);
```

Come si vede si è scritto scanf("%s", s) e non scanf("%s", &s). Il motivo di ciò sarà chiarito quando sarà introdotto il concetto di puntatore.

4.4 Funzioni di Libreria

La libreria standard del C mette a disposizione diverse funzioni per la gestione delle stringhe. Per utilizzare tali funzioni è necessario includere nel proprio file sorgente string.h.

```
#include <string.h>
```

Di seguito sono descritte alcune delle più importanti funzioni per la gestione delle stringhe.

strlen

La funzione `strlen` restituisce la lunghezza della stringa passata come parametro. Per lunghezza della stringa si intende il numero di caratteri contenuti nella stringa escluso il carattere di terminazione.

Scrivere un programma che chieda all'utente di inserire una parola e ne restituisca la lunghezza.

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

void main()
{
   char stringa[50];
   int lunghezza;

   printf("Scrivi una parola:  ");
   scanf("%s", stringa);
   lunghezza = strlen(stringa);
   printf("%s è formata da %d caratteri.",
            stringa, lunghezza);
}
```

Scrivere un programma che chieda all'utente di inserire una parola e ne restituisca la lunghezza senza usare la funzione strlen.

```
#include <stdio.h>

void main()
{
    char stringa[50];
    int i, lunghezza;

    printf("Scrivi una parola:  ");
    scanf("%s", stringa);
    lunghezza = 0;
    for (i=0; stringa[i] ≠ '\0'; i++)
        lunghezza++
```

```
    printf("%s è formata da %d caratteri.",
            stringa, lunghezza);
}
```

Esercizio Convertire il precedente programma in modo che utilizzi il costrutto while piuttosto che il for.

strcmp

La funzione strcmp consente di confrontare due stringhe passate come parametro. Date due stringhe, stringa1 e stringa2, strcmp(stringa1, stringa2) restituisce 0 se stringa1 == stringa2, un numero negativo se stringa1 < stringa2 e un numero positivo se stringa1 > stringa2.

Scrivere un programma che chieda all'utente di inserire due stringhe e che stampi il risultato del confronto

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

void main()
{
    char stringa1[20], stringa2[20];
    int ris;

    printf("inserisci la prima stringa:  ");
    scanf("%s", stringa1);
    printf("inserisci la seconda stringa:  ");
    scanf("%s", stringa2);
    ris = strcmp(stringa1, stringa2);
    if (ris == 0)
        printf("le due stringhe sono uguali");
    else if (ris < 0)
        printf("%s  <  %s", stringa1, stringa2);
    else
        printf("%s  >  %s", stringa1, stringa2);
}
```

Scrivere un programma che chieda all'utente di inserire due stringhe e che stampi il risultato del confronto senza utilizzare la funzione strcmp.

```
#include <stdio.h>

int main(void)
{
    char stringa1[20];
    char stringa2[20];
    int i, fine, len1, len2;

    printf("inserisci la prima stringa:  ");
    scanf("%s", stringa1);
    printf("inserisci la seconda stringa:  ");
    scanf("%s", stringa2);

    i = 0;
    fine = 0;
    len1 = strlen(stringa1);
    len2 = strlen(stringa2);
    while (!fine && i < len1 && i < len2)
        {
            if (stringa1[i] ≠ stringa2[i])
                fine = 1;
            else
                i++;
        }

    if (!fine)
        printf("le due stringhe sono uguali");
    else if (stringa1[i] < stringa2[i])
        printf("%s  <  %s", stringa1, stringa2);
    else
        printf("%s  >  %s", stringa1, stringa2);
}
```

strcpy

Si è detto più volte che le stringhe non sono altro che vettori di caratteri per cui non è possibile assegnare ad una variabile stringa un'altra variabile stringa utilizzando l'operatore =. Le possibili soluzioni sono due:

1. copiare elemento per elemento dalla stringa sorgente alla stringa destinazione fino a quando si incontra il carattere di fine stringa;
2. utilizzare la funzione strcpy.

La funzione strcpy richiede due parametri: il primo parametro rappresenta la variabile stringa di destinazione, il secondo parametro è la variabile stringa sorgente. Ovviamente la stringa destinazione deve essere grande da ospitare tutti gli elementi della stringa sorgente.

Scrivere un programma che letta una stringa immessa da tastiera la copia in un'altra stringa.

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

void main()
{
    char sorgente[20], destinazione[20];

    printf("inserisci una stringa:  ");
    scanf("%s", sorgente);
    strcpy(destinazione, sorgente);
    printf("destinazione = %s", destinazione);
}
```

Scrivere un programma che letta una stringa immessa da tastiera la copia in un'altra stringa senza utilizzare la funzione strcpy.

```
#include <stdio.h>

void main()
{
    char sorgente[20], destinazione[20];
```

```
    printf("inserisci una stringa:  ");
    scanf("%s", sorgente);
    i=0;
    while (sorgente[i] ≠ '\0')
    {
        destinazione[i] = sorgente[i];
        i++;
    }
    destinazione[i] = '\0';
    printf("destinazione = %s", destinazione);
}
```

strcat

La funzione strcat consente di concatenare le due stringhe passate come parametro. Il risultato della concatenzione è memorizzato nel primo parametro. Se per esempio stringa1 contiene "Giuseppe" e stringa2 contiene "Rossi" allora a seguito dell'esecuzione di strcat(stringa1, stringa2) stringa1 conterrà la stringa "GiuseppeRossi".

Scrivere un programa che letti in ingresso nome e cognome di un utente li concateni in un'unica stringa.

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

void main()
{
    char nome[10], cognome[10],
            nome_cognome[20];

    printf("Inserisci il tuo nome:  ");
    scanf("%s", nome);
    printf("Inserisci il tuo cognome:  ");
    scanf("%s", cognome);
    strcpy(nome_cognome, nome);
    strcat(nome_cognome, cognome);
    printf("nome+cognome = %s", nome_cognome);
}
```

Scrivere un programa che letti in ingresso nome e cognome di un utente li concateni in un'unica stringa senza utilizzare la funzione strcat.

```
#include <stdio.h>

void main()
{
    char nome[10], cognome[10],
    nome_cognome[20];
    int i, l1;

    printf("Inserisci il tuo nome:  ");
    scanf("%s", nome);
    printf("Inserisci il tuo cognome:  ");
    scanf("%s", cognome);
    strcpy(nome_cognome, nome);
    l1 = strlen(nome_cognome);
    i = 0;
    while (cognome[i] ≠ '\0')
    {
        nome_cognome[l1+i] = cognome[i];
        i++;
    }
    nome_cognome[l1+i] = '\0';
    printf("nome+cognome = %s", nome_cognome);
}
```

4.5 `gets()` e `puts()`

Le funzioni gets() e puts() consentono l'I/O di stringhe terminate da un carattere di newline o EOF.

`gets()`

Il prototio della funzione `gets()` è:

```
char *gets(char *s);
```

La funzione `gets()` legge una linea dallo standard output nel buffer di memoria puntato da `s` finchè non viene incontrato un carattere di newline o di EOF che è sostituito con il terminatore di stringa.

`puts()`

Il prototio della funzione `puts()` è:

```
int puts(const char *s);
```

La funzione `puts()` scrive una stringa `s` sullo standard output.

4.6 Alcuni Esempi più Complessi

Palindromi

Data una parola immessa da tastiera dire se è un palindromo.

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

void main()
{
    char parola[20];
    int i, len, uguali;

    printf("Immetti una parola:  ");
    scanf("%s", parola);
    len = strlen(parola);
    uguali = 1;
    i = 0;
    while (uguali && i < len/2)
    {
        if (parola[i] ≠ parola[len-i-1])
            uguali = 0;
        else
            i++;
    }
    if (uguali)
```

```
        printf("%s è un palindromo", parola);
    else
        printf("%s non è un palindromo", parola);
}
```

Ricerca

Scrivere un programma che effettui la ricerca binaria su un elenco i N nominativi.

```
#include <stdio.h>

#define N 10

void main()
{
   char elenco[N][50];
   char nome[50];
   int i, trovato, inf, med, sup;

   for (i=0; i<N; i++)
      {
         printf("inserisci un nome:  ");
         scanf("%s", elenco[i]);
      }
   /* si supponga l'elenco ordinato */
   printf("inserisci il nome da cercare:  ");
   scanf("%s", nome);

   trovato = 0;
   inf = 0;
   sup = N-1;
   while (inf ≤ sup && !trovato)
      {
         med = (inf + sup)/2;
         if ( strcmp(nome, elenco[med]) == 0 )
            trovato = 1;
         else if ( strcmp(nome, elenco[med]) < 0 )
            sup = med - 1;
         else
```

```
            inf = med + 1;
        }

    if (trovato)
        printf("%s trovato in posizione %d",
                nome, med);
    else
        printf("%s non trovato", nome);
}
```

Ordinamento

Scrivere un programma che letti N cognomi in un vettore, ordini lo stesso alfabeticamente.

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

#define N 10
void main()
{
    char elenco[N][20];
    char tmp[20];
    int i, sup, ultimo;

    for (i=0; i<N; i++)
        {
            printf("inserisci cognome:  ");
            scanf("%s", elenco[i]);
        }

    sup = N-1;
    while (sup ≠ 0)
     {
        ultimo = 0;
        for (i=0; i<sup; i++)
            if (strcmp(elenco[i],elenco[i+1]) > 0)
                {
                    ultimo = i;
                    strcpy(tmp, elenco[i]);
```

```
                strcpy(elenco[i], elenco[i+1]);
                strcpy(elenco[i+1], tmp);
            }
        sup = ultimo;
    }
  for (i=0; i<N; i++)
    printf("%s\n", elenco[i]);
}
```

Conversione Binario a Decimale

Scrivere un programma che effettui la conversione di un numero binario in un numero decimale.

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

void main()
{
  char binario[17];
  unsigned int decimale, peso;
  int i, nbits;

  printf("inserisci un numero binario:  ");
  scanf("%s", binario);

  decimale = 0;
  peso = 1;
  nbits = strlen(binario);
  for (i=nbits-1; i≥0; i--)
    {
      decimale = decimale +
                    peso ∗ (binario[i] - ’0’);
      peso = peso∗2;
    }

  printf("%s in decimale vale %d", binario, decimale);
}
```

Conversione Decimale a Binario

Scrivere un programma che effettui la conversione di un numero decimale in binario.

```
#include <stdio.h>

void main()
{
   char binario[17];
   unsigned int decimale;
   int i, j, resto;

   printf("inserisci un numero:  ");
   scanf("%d", &decimale);
   printf("%d in binario vale:  ", decimale);

   i = 0;
   do {
      resto = decimale % 2;
      decimale = decimale / 2;
      binario[i] = resto + '0';
      i++;
   } while (decimale ≠ 0);
   binario[i] = '\0';

   for (j=i-1; j ≥ 0; j--)
      printf("%c", binario[j]);
}
```

Capitolo 5

Puntatori

La memoria di un calcolatore è suddivisa in *locazioni di memoria* ciascuna delle quali ha una capienza fissa espressa in bit. Quando si fa riferimento alla memoria si considerano due dimensioni:

- l'estensione della memoria;
- la capacità di ciascuna locazione.

Le operazioni attuabili su una memoria sono 2:

1. scrittura di un dato;
2. lettura di un dato.

Ogni locazione di memoria deve essere identificabile. Ciò avviene attribuendo ad ogni locazione un *indirizzo* usato come etichetta. Esiste quindi una corrispondenza biunivoca tra etichetta e locazione di memoria: cioè ad ogni etichetta corrisponde una locazione e, viceversa, ad ogni locazione corrisponde un'etichetta.

Le operazioni di scrittura in memoria sovrascrivono i dati contenuti nelle locazioni interessate, mentre quelle di lettura non operano alcun cambiamento delle locazioni coinvolte.

5.1 Indirizzi

Per determinare il numero di bit necessario ad "etichettare" N locazioni di memoria si ricorre alla seguente espressione: $\log_2 N$.

Per esempio. Per indirizzare 1024 locazioni di memoria ogni indirizzo deve essere formato da un numero di bit pari a:

$$\log_2 1024 = 10$$

5.2 Associazione Variabile—Indirizzo

Tutte le volte che in un programma si dichiarano delle variabili, l'esecutore associa alla variabile un indirizzo.

```
int a, b, c;
```

Si crea in questo modo una *tabella di associazioni* variabile—indirizzo (Tabella 5.1)

variabile	indirizzo
a	Ia
b	Ib
c	Ic

Tabella 5.1: Tabella di associazioni variabile—indirizzo.

Per esempio, se Ia=63, Ib=79 e Ic=3 date le seguenti assegnazioni:

```
a = 7;
b = 14;
c = 21;
```

la configurazione della memoria sarà come quella illustrata in Tabella 5.2.

Esempio

Quando l'esecutore si trova ad eseguire un'operazione del tipo:

```
a = b + c;
```

egli valuta prima l'espressione alla destra dell'uguale:

1. identifica `b` considerando l'indirizzo che gli è stato associato in fase di dichiarazione e legge il contenuto di tale locazione;
2. identifica `c` considerando l'indirizzo che gli è stato associato in fase di dichiarazione e legge il contenuto di tale locazione;
3. somma `b` e `c`;
4. assegna alla variabile `a` (cioè alla locazione il cui indirizzo è stato associato ad `a`) la somma di `b` e `c`.

indirizzo	locazione di memoria
0	
1	
2	
3	21
4	
...	
62	
63	7
64	
...	
78	
79	14
80	
...	

Tabella 5.2: Configurazione della memoria dopo le assegnazioni a=7; b=14; c=21;

Memoria Riservata per Tipi di Dati

La quantità di locazioni di memoria riservate a ciascuna variabile (`int`, `char`, `float`, ecc.) dipende dal tipo di compilatore e dal tipo di dato adoperato.

Quando si dichiara un vettore:

```
int tab[N];
```

l'esecutore riserva una quantità di locazioni di memoria tale da contenere N variabili di tipo intero.

L'indirizzo associato al vettore `tab` sarà quello della prima locazione di memoria riservata al primo elemento del vettore. Per determinare l'indirizzo della locazione di memoria del generico elemento di posizione `i`, l'esecutore applicherà la seguente formula:

$$\begin{aligned} indirizzo\ elemento\ di\ posizione\ i = & indirizzo\ del\ primo\ elemento + \\ & + i \times numero\ di\ locazioni\ riservate\ per\ un\ int \end{aligned}$$

In linguaggio C la funzione `sizeof(...)` restituisce la dimensione in bytes del dato (o tipo di dato) passato come parametro. Quindi l'elemento di posizione `i` del vettore `tab` si troverà $i \times sizeof(int)$ byte dopo l'indirizzo di `tab` (cioè del primo elemento di `tab`).

5.3 I Puntatori

I puntatori sono variabili atte a contenere l'indirizzo di locazioni di memoria in cui sono memorizzate le variabili di un certo tipo.

Per esempio, per dichiarare una variabile puntatore che dovrà contenere l'indirizzo di una variabile intera basta fare:

```
int *p;
```

Generalizzando, per dichiarare una variabile puntatore che dovrà contenere l'indirizzo di una variabile di tipo `tipo` basta fare:

```
tipo *p;
```

Per ottenere l'indirizzo di una variabile si usa l'operatore &. Per esempio, si consideri il seguente frammento di codice:

```
int *p;
int a;

p = &a;
```

l'ultima istruzione assegna alla variabile puntatore ad intero `p` l'indirizzo della variabile intera `a`. Diremo in questo caso che `p` *sta puntando ad* `a`.

L'accesso al dato contenuto nella locazione di memoria il cui indirizzo è memorizzato in una variabile puntatore, avviene utilizzando l'operatore *. Per esempio si consideri il seguente frammento di codice:

```
int *p;
int a;

p = &a;
a = 7;
printf(''%d'', *p);

*p = 18;
printf(''%d'', a);
```

L'esecuzione di questo codice visualizzerà 7 e poi 18.

Esempio

Si consideri la seguente dichiarazione:

```
int *p;
int a, b;
```

che supponiamo dia luogo alla tabella delle associazioni 5.3.

variabile	indirizzo
a	68
b	80
p	90

Tabella 5.3: Tabella di associazioni variabile—indirizzo.

Per assegnare il valore 47 alla variabile `a` basta fare:

```
a = 47;
```

Un'altro metodo possibile e' fare:

```
p = &a;
*p = 47;
```

in cui la prima istruzione (`p = &a;`) assegna a `p` l'indirizzo della variabile `a` e la seconda istruzione (`*p = 47;`) scrive nella locazione di memoria, il cui indirizzo è memorizzato in `p`, il valore 47. La configurazione della memoria sarà come quella illustrata in Tabella 5.4.

Assegnazioni tra Puntatori

In C non è possibile attribuire ad un puntatore un dato indirizzo in maniera diretta. Cioè non è possibile fare:

```
int *p;
p = 105;
```

La forma corretta è:

```
int *p;
int a;
p = &a;
```

indirizzo	locazione di memoria
...	
67	
68	47
69	
... 79	
80	7
81	
...	
89	
90	68
91	
...	

Tabella 5.4: Configurazione della memoria.

dove si suppone che `a` sia stata allocata all'indirizzo 105.

Si badi a non confondere le seguenti sintassi:

```
int *p, *q;
p = q;
*p = *q;
```

La prima assegna al puntatore `p` l'indirizzo contenuto in `q`. Cioè se `q` punta alla locazione 39 allora a seguito dell'assegnazione `p = q;` anche `p` punterà alla locazione 39. La seconda, invece, assegna alla locazione puntata da `p` il valore contenuto nella locazione puntata da `q`.

Aritmetica dei Puntatori

L'insieme delle operazioni eseguite sugli indirizzi memorizzati nelle variabili puntatore costituisce l'aritetica dei puntatori.

```
int *p;
p = p + 1;
p++;
```

non significa sommare 1 all'indirizzo contenuto in `p` bensì significa incrementare l'indirizzo di una quantità in byte pari alla dimensione del dato puntato. In questo caso se per esempio `p` punta all'indirizzo 30, `p = p + 1` assegna a `p` l'indirizzo `30 + sizeof(int)` e non l'indirizzo 31.

Puntatori e Vettori

Quando si usa un vettore è possibile specificare un suo elemento non solo indicando la sua posizione tramite l'indice tra le parentesi quadre ma anche nel seguente modo. Sia data la seguente dichiarazione:

```
int tab[10];
int *p;
```

Scrivere:

```
tab[3] = 49;
```

equivale a scrivere:

```
p = tab;
p = p + 3;
*p = 49;
```

Quindi `tab[i]` o `*(p+i)` sono equivalenti. Ricordando che `tab` rappresenta anche il puntatore al primo elemento dell'array, per accedere all'elemento i-esimo si può anche usare `*(tab+i)`. Analogamente si può scrivere `p[i]`.

Capitolo 6

Funzioni

Le funzioni sono i blocchi costitutivi del C e il luogo in cui si svolge tutta l'attività del programma. Esse rappresentano una delle più importanti funzionalità del C che danno l'opportunità di generalizzare ed accrescere la leggibilità del codice.

6.1 La Forma Generale di Funzione

La forma generale di funzione è la seguente:

```
SpecificatoreTipo NomeFunzione(ElencoParametri)
{
        CorpoDellaFunzione;
}
```

Lo `SpecificareTipo` specifica il tipo dei dati restituito dalla funzione. L'`ElencoParametri` è un elenco di nomi di variabili separate da virgole e preceduti dai rispettivi tipi. Tali parametri noti come parametri formali ricevono il valore degli argomenti passati (noti come parametri attuali) nel momento dell'invocazione della funzione.

Una funzione può anche essere senza parametri, in tal caso l'elenco dei parametri sarà vuoto. Tuttavia, anche se non vi è alcun parametro, è richiesta la presenza delle parentesi.

Sappiamo che nella dichiarazione di variabili, è possibile dichiarare più variabili di un tipo comune utilizzando un elenco di nomi di variabili separati da virgole:

```
int a, b, c;
```

Al contrario, tutti i parametri della funzione devono essere dichiarati singolarmente e per ognuno si deve specificare sia il tipo che il nome:

```
int Somma(int a, int b)
```

6.2 Regole di Visibilità delle Funzioni

Le regole di visibilità di un linguaggio sono le regole che governano ciò che una parte del codice conosce e le possibilità di accesso cge ha su un'altra parte di codice o dati.

Dichiarazione e Definizione di Funzione

La dichiarazione di una funzione è rappresentata dal prototipo della funzione. Cioè dalla dichiarazione del tipo di dato restituito, del nome della funzione, e dall'elenco dei tipi dei parametri formali. Per esempio con:

```
int Somma(int a, int b);
```

si è dichiarata una funzione di nome Somma che accetta due parametri di tipo intero e restituisce un risultato intero.

La definizione di una funzione rappresenta l'implementazione delle funzionalità (comportamento) della funzione. Cioè l'insieme della dichiarazione della funzione ed il corpo della stessa. Per esempio:

```
int Somma(int a, int b)
{
   return a+b;
}
```

definisce la funzione `Somma` implementando le sue funzionalità (che sono quelle di sommare i due addendi `a` e `b` passati come parametri e restituire la somma utilizzando l'istruzione `return`.

Visibilità delle Funzioni

Una funzione F1 può essere invocata da una funzione F2 se e solo se F1 è stata *dichiarata* o *definita* prima di F2.

```
void F1()
{
        /* corpo della funzione F1 */
}

void F2()
{
        ...
        F1(); /* invocazione della funzione F1 */
        ...
}
```

Se F1 è definita dopo F2, è possibile ancora per F2 invocare F1 se e solo se prima di F2 la funzione F1 è stata dichiarata attraverso il suo prototipo.

```
void F1(); /* dichiarazione di F1 */

void F1()  /* definizione di F2 */
{
        ...
        F1();
        /* F2 òpu invocare F1 èperch F1  */
è        /*  stata dichiarata */
        ...
}

void F1() /* definizione di F1 */
{
        /* corpo della funzione F1 */
}
```

Visibilità delle Variabili

Le istruzioni del corpo di una funzione possono accedere alla variabili dichiarate localmente alla funzione stessa o a quelle esterne solo se queste si trovano nell'ambiente globale del programma.

```
int a; /* aè  una variabile globale */

void F1()
{
        int b; /* bè  locale a F1 */
        printf("%d %d", a, b);
}
```

La Funzione F1 può accedere alla variabile locale b e alla variabile globale a.

```
int a; /* aè  una variabile globale */

void F1()
{
        int b1; /* b1è  locale a F1 */
        printf("%d %d", a, b1);
}
```

```
void F2()
{
        int b2; /* b2è  locale a F2 */
        printf("%d %d", a, b2);
}
```

Le istruzioni all'interno della funzione F2 possono accedere alla variabile locale `b2` e alla variabile globale `a`. Non possono accedere alla variabile locale `b1` di F1 visto che questa si trova in un ambiente non visibile da F2. Allo stesso modo le istruzioni in F1 non possono accedere alla variabile `b2`.

6.3 Gli Argomenti delle Funzioni

Se una funzione deve utilizzare degli argomenti, deve dichiarare delle variabili che accettino i valori degli argomenti. Queste variabili sono chiamate i parametri formali della funzione. Essi si comportano come ogni altra variabile locale della funzione e vengono creati all'ingresso nella funzione e distrutti all'uscita.

Chiamata per Valore e per Indirizzo

In generale è possibile passare argomenti alle funzioni in due modi. Il primo è detto *chiamata per valore*. Questo metodo copia il valore di un argomento nel parametro formale della funzione. In questo caso, ogni modifica eseguita sul parametro non avrà alcun effetto sull'argomento passato.

```
void StampaIncremento(int a)
{
        a = a + 1;
        printf("a=%d\n", a);
}

main()
{
        int b=0;
        StampaIncremento(b);
        printf("b=%d\n", b);
}
```

L'esecuzione di questo programma visualizza:

```
a=1
b=0
```

Il secondo metodo per passare argomenti ad una funzione è la *chiamata per indirizzo*. Con questo metodo nel parametro viene copiato l'indirizzo dell'argomento. All'interno della funzione, è possibile utilizzare l'indirizzo per accedere all'effettivo argomento utilizzato nella chiamata. Questo significa che le modifiche eseguite sul parametro avranno effetto anche sull'argomento.

```
void StampaIncremento(int *a)
{
        *a = *a + 1;
        printf("a=%d\n", *a);
}

main()
{
        int b=0;
        StampaIncremento(&b);
        printf("b=%d\n", b);
}
```

L'esecuzione di questo programma visualizza:

```
a=1
b=1
```

Creazione di una Chiamata per Indirizzo

Anche se la convenzione di passaggio dei parametri del C prevede la chiamata per valore, è possibile creare una chiamata per indirizzo passando alla funzione un puntatore ad un argomento al posto dell'argomento stesso. Poiché alla funzione viene passato l'indirizzo dell'argomento, il codice che si trova all'interno della funzione può modificare il valore dell'argomento che si trova all'esterno della funzione stessa.

Ad esempio la funzione `Scambia()` che scambia i valori delle due variabili intere puntate dai suoi argomenti potrebbe avere il seguente aspetto:

```
void Scambia(int *a, int *b)
{
        int tmp;

        tmp = *a;
        *a  = *b;
```

```
        *b   = tmp;
}
```

`Scambia()` è in grado di scambiare i valori delle due variabili puntate da a e b perchè le vengono passati gli indirizzi delle variabili e non i valori. Pertanto, all'interno della funzione, sarà possibile accedere al contenuto delle variabili utilizzando le comuni operazioni sui puntatori. Il seguente programma mostra il modo corretto per utilizzare la funzione `Scambia()`:

```
void Scambia(int *a, int *b); /* dichiarazione del prototipo */

main()
{
        int x, y;

        x = 10;
        y = 20;
        Scambia(&x, &y); /* passa gli indirizzi di x e y */
}
```