Esempi di programmazione in C

In questo capitolo vengono mostrati alcuni algoritmi elementari o comuni portati in C. Per la spiegazione di questi, se non sono già conosciuti, conviene leggere quanto riportato a partire dalla sezione 62.2.

«

Somma tra due numeri positivi

«

Una copia di questo file dovrebbe essere disponibile presso *allegati/somma.c* .

Listato u3.1. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/0fuGkfkz* , *http://ideone.com/SQBOl* .

```
#include <stdio.h>

int
somma (int x, int y)
{
    int z = x;
    int i;

    for (i = 1; i <= y; i++)
      {
        z++;
      };

    return z;
}

int
main (int argc, char *argv[])
{
    int x;
    int y;
    int z;

    // Converte le stringhe ottenute dalla riga di comando in
    // numeri interi e li assegna alle variabili x e y.
```

```
    sscanf (argv[1], "%i", &x);
    sscanf (argv[2], "%i", &y);

    z = somma (x, y);

    printf ("%i + %i = %i\n", x, y, z);

    return 0;
}
```

In alternativa si può tradurre il ciclo '**`for`**' in un ciclo '**`while`**'.

Listato u3.2. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/TKwppy25* , *http://ideone.com/K33pD33* .

```
int
somma (int x, int y)
{
    int z = x;
    int i = 1;

    while (i <= y)
      {
        z++;
        i++;
      };

    return z;
}
```

Moltiplicazione di due numeri positivi attraverso la somma

«

Una copia di questo file dovrebbe essere disponibile presso *allegati/moltiplica.c* .

Listato u3.3. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/xjV7MZte* , *http://ideone.com/b5Wxx* .

```
#include <stdio.h>

int
moltiplica (int x, int y)
{
    int z = 0;
    int i;

    for (i = 1; i <= y; i++)
      {
        z = z + x;
      }

    return z;
}

int
main (int argc, char *argv[])
{
    int x;
    int y;
    int z;

    // Converte le stringhe ottenute dalla riga di comando
    // in numeri interi e li assegna alle variabili x e y.

    sscanf (argv[1], "%i", &x);
    sscanf (argv[2], "%i", &y);
```

```
    z = moltiplica (x, y);

    printf ("%i * %i = %i\n", x, y, z);

    return 0;
}
```

In alternativa si può tradurre il ciclo '**for**' in un ciclo '**while**'.

Listato u3.4. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/9GQLZmXk* , *http://ideone.com/CoQf0* .

```
int moltiplica (int x, int y)
{
    int z = 0;
    int i = 1;

    while (i <= y)
      {
        z = z + x;
        i++;
      }

    return z;
}
```

Divisione intera tra due numeri positivi

Una copia di questo file dovrebbe essere disponibile presso *allegati/dividi.c* .

Listato u3.5. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/h15j1MVo* , *http://ideone.com/EmJ3X* .

```
#include <stdio.h>

int
dividi (int x, int y)
{
    int z = 0;
    int i = x;

    while (i >= y)
      {
        i = i - y;
        z++;
      }

    return z;
}

int
main (int argc, char *argv[])
{
    int x;
    int y;
    int z;

    // Converte le stringhe ottenute dalla riga di comando
    // in numeri interi e li assegna alle variabili x e y.

    sscanf (argv[1], "%i", &x);
    sscanf (argv[2], "%i", &y);

    z = dividi (x, y);
```

```
    printf ("Divisione intera - %i:%i = %i\n", x, y, z);

    return 0;
}
```

Elevamento a potenza

Una copia di questo file dovrebbe essere disponibile presso *allegati/exp.c* .

Listato u3.6. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/mxkVE5yi* , *http://ideone.com/guhJ6* .

```
#include <stdio.h>

int
exp (int x, int y)
{
    int z = 1;
    int i;

    for (i = 1; i <= y; i++)
      {
        z = z * x;
      }

    return z;
}

int
main (int argc, char *argv[])
{
    int x;
    int y;
    int z;
```

```
    // Converte le stringhe ottenute dalla riga di comando
    // in numeri interi e li assegna alle variabili x e y.

    sscanf (argv[1], "%i", &x);
    sscanf (argv[2], "%i", &y);

    z = exp (x, y);

    printf ("%i ** %i = %i\n", x, y, z);

    return 0;
}
```

In alternativa si può tradurre il ciclo '**for**' in un ciclo '**while**'.

Listato u3.7. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/7uP7uRGg* , *http://ideone.com/4X81h* .

```
int
exp (int x, int y)
{
    int z = 1;
    int i = 1;

    while (i <= y)
      {
        z = z * x;
        i++;
      };

    return z;
}
```

È possibile usare anche un algoritmo ricorsivo.

Listato u3.8. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/mwiZu0T7* , *http://ideone.com/x0ei9* .

```
int exp (int x, int y)
{
    if (x == 0)
      {
        return 0;
      }
    else if (y == 0)
      {
        return 1;
      }
    else
      {
        return (x * exp (x, y-1));
      }
}
```

Radice quadrata

Una copia di questo file dovrebbe essere disponibile presso *allegati/radice.c* .

Listato u3.9. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/8FgV7l1v* , *http://ideone.com/rO8KS* .

```
#include <stdio.h>

int
radice (int x)
{
    int z = 0;
    int t = 0;

    while (1)
```

```
      {
        t = z * z;

        if (t > x)
          {
            //  È stato superato il valore massimo.
            z--;
            return z;
          }

        z++;
      }

    // Teoricamente, non dovrebbe mai arrivare qui.
}

int
main (int argc, char *argv[])
{
    int x;
    int z;

    sscanf (argv[1], "%i", &x);

    z = radice (x);

    printf ("radq(%i) = %i\n", x, z);

    return 0;
}
```

Una copia di questo file dovrebbe essere disponibile presso *allegati/fatt.c* .

Listato u3.10. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/vBGnZfEf* , *http://ideone.com/jYVgR* .

```
#include <stdio.h>

int
fatt (int x)
{
    int i = (x - 1);

    while (i > 0)
      {
        x = x * i;
        i--;
      }

    return x;
}

int
main (int argc, char *argv[])
{
    int x;
    int z;

    sscanf (argv[1], "%i", &x);

    z = fatt (x);

    printf ("%i! = %i\n", x, z);
```

```
    return 0;
}
```

In alternativa, l'algoritmo si può tradurre in modo ricorsivo.

Listato u3.11. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/LIC6Vyxp* , *http://ideone.com/6SfUj* .

```
int
fatt (int x)
{
    if (x > 1)
      {
        return (x * fatt (x - 1));
      }
    else
      {
        return 1;
      }
}
```

Massimo comune divisore

«

Una copia di questo file dovrebbe essere disponibile presso *allegati/mcd.c* .

Listato u3.12. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/ETf2XcdR* , *http://ideone.com/8H1og* .

```
#include <stdio.h>

int
mcd (int x, int y)
```

```
{
    while (x != y)
      {
        if (x > y)
          {
            x = x - y;
          }
        else
          {
            y = y - x;
          }
      }

    return x;
}

int
main (int argc, char *argv[])
{
    int x;
    int y;
    int z;

    sscanf (argv[1], "%i", &x);
    sscanf (argv[2], "%i", &y);

    z = mcd (x, y);

    printf ("Il massimo comune divisore di %i e %i è %i\n",
            x, y, z);

    return 0;
}
```

Numero primo

«

Una copia di questo file dovrebbe essere disponibile presso *allegati/primo.c* .

Listato u3.13. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/T6vjaM2y* , *http://ideone.com/X1oos* .

```
#include <stdio.h>

unsigned int
primo (int x)
{
    unsigned int primo = 1;
    int i = 2;
    int j;

    while ((i < x) && primo)
      {
        j = x / i;
        j = x - (j * i);

        if (j == 0)
          {
            primo = 0;
          }
        else
          {
            i++;
          }
      }

    return primo;
}
```

```
int
main (int argc, char *argv[])
{
    int x;

    sscanf (argv[1], "%i", &x);

    if (primo (x))
      {
        printf ("%i è un numero primo\n", x);
      }
    else
      {
        printf ("%i non è un numero primo\n", x);
      }

    return 0;
}
```

Successione di Fibonacci

«

Gli esempi mostrano una funzione che restituisce l'elemento ***n***-esimo nella sequenza di Fibonacci, mentre la chiamata di questa funzione viene fatta in modo da ottenere l'elenco dei primi ***n*** numeri di Fibonacci. La prima soluzione mostra una funzione ricorsiva. Una copia di questo file dovrebbe essere disponibile presso *allegati/fibonacci.c* .

Listato u3.14. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/6CHi9taB* , *http://ideone.com/0Z1dz* .

```
#include <stdio.h>

unsigned int
fibonacci (unsigned int n)
{
    if (n == 0)
      {
        return 0;
      }
    else if (n == 1)
      {
        return 1;
      }
    else
      {
        return (fibonacci (n-1) + fibonacci (n-2));
      }
}

int
main (int argc, char *argv[])
{
    unsigned int n;
    unsigned int i;

    sscanf (argv[1], "%u", &n);

    for (i = 0; i <= n; i++)
      {
        printf ("%u ", fibonacci (i));
      }
```

```
    printf ("\n");

    return 0;
}
```

L'esempio seguente mostra solo la funzione, in forma iterativa:

Listato u3.15. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/1sH2hhIf* , *http://ideone.com/8z9J8* .

```
unsigned int
fibonacci (unsigned int n)
{
    unsigned int f1 = 1;
    unsigned int f0 = 0;
    unsigned int fn = n;
    unsigned int i;

    for (i = 2; i <= n; i++)
      {
        fn = f1 + f0;
        f0 = f1;
        f1 = fn;
      }

    return fn;
}
```

«

Ricerca sequenziale

«

Una copia di questo file dovrebbe essere disponibile presso *allegati/ricercaseq.c* .

Listato u3.16. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/9p0P3GU9* , *http://ideone.com/J8hQb* .

```
#include <stdio.h>

int
ricercaseq (int lista[], int x, int a, int z)
{
    int i;

    // Scandisce l'array alla ricerca dell'elemento.

    for (i = a; i <= z; i++)
      {
        if (x == lista[i])
          {
            return i;
          }
      }

    // La corrispondenza non è stata trovata.

    return -1;
}

int
main (int argc, char *argv[])
```

```
{
    int lista[argc - 2];
    int x;
    int i;

    // Acquisisce il primo argomento come valore da cercare.

    sscanf (argv[1], "%i", &x);

    // Considera gli argomenti successivi come gli elementi
    // dell'array da scandire.

    for (i = 2; i < argc; i++)
      {
        sscanf (argv[i], "%i", &lista[i-2]);
      }

    // Esegue la ricerca.

    i = ricercaseq (lista, x, 0, argc - 2);

    // Emette il risultato.

    printf ("%i si trova nella posizione %i\n", x, i);

    return 0;
}
```

Al posto di dichiarare l'array ***lista*** con una quantità di elementi definita in fase di funzionamento, si può usare la funzione ***malloc()***, avendo cura di incorporare il file '`stdlib.h`':

Listato u3.17. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/5nP79Nf798* , *http://ideone.com/YdMdC* .

```
#include <stdio.h>
#include <stdlib.h>
…
int
main (int argc, char *argv[])
{
    int *lista = (int *) malloc ((argc - 2) * sizeof (int));
…
```

Esiste anche una soluzione ricorsiva che viene mostrata nella subroutine seguente:

Listato u3.18. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/SC2AheV2* , *http://ideone.com/sk64m* .

```
int
ricercaseq (int lista[], int x, int a, int z)
{
    if (a > z)
      {
        // La corrispondenza non è stata trovata.

        return -1;
      }
    else if (x == lista[a])
      {
        return a;
      }
    else
      {
        return ricercaseq (lista, x, a+1, z);
      }
}
```

Ricerca binaria

«

Una copia di questo file dovrebbe essere disponibile presso *allegati/ricercabin.c* .

Listato u3.19. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/vU9RwE9m* , *http://ideone.com/jEyYk* .

```
#include <stdio.h>

int
ricercabin (int lista[], int x, int a, int z)
```

```
{
    int m;

    // Determina l'elemento centrale.

    m = (a + z) / 2;

    if (m < a)
      {
        // Non restano elementi da controllare: l'elemento
        // cercato non c'è.

        return -1;
      }
    else if (x < lista[m])
      {
        // Si ripete la ricerca nella parte inferiore.

        return ricercabin (lista, x, a, m-1);
      }
    else if (x > lista[m])
      {
        // Si ripete la ricerca nella parte superiore.

        return ricercabin (lista, x, m+1, z);
      }
    else
      {
        // La variabile m rappresenta l'indice dell'elemento
        // cercato.

        return m;
      }
}
```

```
int main (int argc, char *argv[])
{
    int lista[argc - 2];
    int x;
    int i;

    // Acquisisce il primo argomento come valore da cercare.

    sscanf (argv[1], "%i", &x);

    // Considera gli argomenti successivi come gli elementi
    // dell'array da scandire.

    for (i = 2; i < argc; i++)
      {
        sscanf (argv[i], "%i", &lista[i-2]);
      }

    // Esegue la ricerca.

    i = ricercabin (lista, x, 0, argc-2);

    // Emette il risultato.

    printf ("%i si trova nella posizione %i\n", x, i);

    return 0;
}
```

Per questo esempio vale la stessa considerazione fatta nella sezione precedente, a proposito dell'uso di ***malloc()*** al posto di un array con una quantità di elementi definita dinamicamente durante il

funzionamento del programma.

Algoritmi tradizionali

«

Bubblesort

«

Viene mostrata prima una soluzione iterativa e successivamente la funzione '**bsort**' in versione ricorsiva.

Una copia di questo file dovrebbe essere disponibile presso *allegati/bsort.c* .

Listato u3.20. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/10ot10rUOE* , *http://ideone.com/hJIie* .

```
#include <stdio.h>

void
bsort (int lista[], int a, int z)
{
    int scambio;
    int j;
    int k;

    // Inizia il ciclo di scansione dell'array.

    for (j = a; j < z; j++)
      {
        // Scansione interna dell'array per collocare nella
        // posizione j l'elemento giusto.

        for (k = j+1; k <= z; k++)
          {
            if (lista[k] < lista[j])
              {
```

```
                    // Scambia i valori.

                    scambio = lista[k];
                    lista[k] = lista[j];
                    lista[j] = scambio;
                  }
              }
        }
}

int
main (int argc, char *argv[])
{
    int lista[argc-1];
    int i;

    // Considera gli argomenti come gli elementi
    // dell'array da ordinare.

    for (i = 1; i < argc; i++)
      {
        sscanf (argv[i], "%i", &lista[i-1]);
      }

    // Esegue il riordino.

    bsort (lista, 0, argc-2);

    // Emette il risultato.

    for (i = 0; i < (argc-1); i++)
      {
        printf ("%i ", lista[i]);
      }
```

```
    printf ("\n");

    return 0;
}
```

Segue la funzione '**bsort**' in versione ricorsiva.

Listato u3.21. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/izHITJ7y* , *http://ideone.com/5mSDT* .

```
void
bsort (int lista[], int a, int z)
{
    int scambio;
    int k;

    if (a < z)
      {
        // Scansione interna dell'array per collocare nella
        // posizione a l'elemento giusto.

        for (k = a+1; k <= z; k++)
          {
            if (lista[k] < lista[a])
              {
                // Scambia i valori.

                scambio = lista[k];
                lista[k] = lista[a];
                lista[a] = scambio;
              }
          }

        bsort (lista, a+1, z);
```

```
        }
}
```

Al posto di dichiarare l'array ***lista*** con una quantità di elementi definita in fase di funzionamento, si può usare la funzione ***malloc()***, avendo cura di incorporare il file '`stdlib.h`':

Listato u3.22. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/M76CQ76gHg* , *http://ideone.com/j7hB5* .

```
#include <stdio.h>
#include <stdlib.h>
…
int
main (int argc, char *argv[])
{
    int *lista = (int *) malloc ((argc - 1) * sizeof (int));
…
```

Torre di Hanoi

«

Una copia di questo file dovrebbe essere disponibile presso *allegati/hanoi.c* .

Listato u3.23. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/FvvVQmru* , *http://ideone.com/kr6DC* .

```
#include <stdio.h>

void hanoi (int n, int p1, int p2)
{
    if (n > 0)
```

```
        {
          hanoi (n-1, p1, 6-p1-p2);
          printf ("Muovi l'anello %i dal piolo %i "
                  "al piolo %i\n",
                  n, p1, p2);
          hanoi (n-1, 6-p1-p2, p2);
        }
}

int main (int argc, char *argv[])
{
    int n;
    int p1;
    int p2;

    sscanf (argv[1], "%i", &n);
    sscanf (argv[2], "%i", &p1);
    sscanf (argv[3], "%i", &p2);

    hanoi (n, p1, p2);

    return 0;
}
```

Quicksort

«

Una copia di questo file dovrebbe essere disponibile presso *allegati/qsort.c* .

Listato u3.24. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/QbDq6aMz* , *http://ideone.com/NJwFO* .

```
#include <stdio.h>

int
part (int lista[], int a, int z)
{
    // Viene preparata una variabile per lo scambio di
    // valori.

    int scambio = 0;

    // Si assume che «a» sia inferiore a «z».

    int i = a + 1;
    int cf = z;

    // Inizia il ciclo di scansione dell'array.

    while (1)
      {
        while (1)
          {
            // Sposta «i» a destra.

            if ((lista[i] > lista[a]) || (i >= cf))
              {
                break;
              }
            else
              {
                i += 1;
              }
```

```
        }
      while (1)
        {
          // Sposta «cf» a sinistra.

          if (lista[cf] <= lista[a])
            {
              break;
            }
          else
           {
              cf -= 1;
           }
        }
      if (cf <= i)
        {
          // È avvenuto l'incontro tra «i» e «cf».

          break;
        }
      else
        {
          // Vengono scambiati i valori.

          scambio = lista[cf];
          lista[cf] = lista[i];
          lista[i] = scambio;

          i += 1;
          cf -= 1;
        }
    }

  // A questo punto lista[a..z] è stata ripartita e «cf» è
```

```
    // la collocazione di «lista[a]».

    scambio = lista[cf];
    lista[cf] = lista[a];
    lista[a] = scambio;

    // A questo punto, lista[cf] è un elemento (un valore)
    // nella giusta posizione.

    return cf;
}

void
quicksort (int lista[], int a, int z)
{
    // Viene preparata la variabile «cf».

    int (cf) = 0;

    if (z > a)
      {
        cf = part (lista, a, z);
        quicksort (lista, a, cf-1);
        quicksort (lista, cf+1, z);
      }
}

int
main (int argc, char *argv[])
{
    int lista[argc - 1];
    int i;

    // Considera gli argomenti come gli elementi
```

```
    // dell'array da ordinare.

    for (i = 1; i < argc; i++)
      {
        sscanf (argv[i], "%i", &lista[i-1]);
      }

    // Esegue il riordino.

    quicksort (lista, 0, argc-2);

    // Emette il risultato.

    for (i = 0; i < (argc-1); i++)
      {
        printf ("%i ", lista[i]);
      }
    printf ("\n");

    return 0;
}
```

Per questo esempio vale la stessa considerazione già fatta a proposito dell'uso di ***malloc()*** al posto di un array con una quantità di elementi definita dinamicamente durante il funzionamento del programma.

Permutazioni

«

Una copia di questo file dovrebbe essere disponibile presso *allegati/permuta.c* .

Listato u3.25. Per provare il codice attraverso un servizio *pastebin*: *http://codepad.org/ca66C9da* , *http://ideone.com/I5bJV* .

```
#include <stdio.h>

void visualizza (int lista[], int dimensione)
{
    int i;

    for (i = 0; i < dimensione; i++)
      {
        printf ("%i ", lista[i]);
      }
    printf ("\n");
}

void permuta (int lista[], int a, int z, int dimensione)
{
    int scambio;
    int k;

    // Se il segmento di array contiene almeno due elementi,
    // si procede.

    if ((z - a) >= 1)
      {
        // Inizia un ciclo di scambi tra l'ultimo elemento e
        // uno degli altri contenuti nel segmento di array.

        for (k = z; k >= a; k--)
          {
            // Scambia i valori.

            scambio = lista[k];
```

```
            lista[k] = lista[z];
            lista[z] = scambio;

            // Esegue una chiamata ricorsiva per permutare
            // un segmento più piccolo dell'array.

            permuta (lista, a, z - 1, dimensione);

            // Scambia i valori.

            scambio = lista[k];
            lista[k] = lista[z];
            lista[z] = scambio;
          }
      }
    else
      {
        // Visualizza l'array.

        visualizza (lista, dimensione);
      }
}

int
main (int argc, char *argv[])
{
    int lista[argc - 1];
    int i;

    // Considera gli argomenti come gli elementi
    // dell'array da permutare.

    for (i = 1; i < argc; i++)
      {
```

```
        sscanf (argv[i], "%i", &lista[i-1]);
      }

    // Esegue le permutazioni.

    permuta (lista, 0, argc - 2, argc - 1);

    return 0;
}
```

Per questo esempio vale la stessa considerazione già fatta a proposito dell'uso di ***malloc()*** al posto di un array con una quantità di elementi definita dinamicamente durante il funzionamento del programma.