Appunti di Linguaggi di Programmazione

Linguaggio – C/C++

Domande e risposte

1) Sintassi C/C++

Il linguaggio C/C++ è composto da una serie di elementi lessicali, cioè parole che identificano le più piccole unità a cui attribuire un significato. Questi elementi sono divisi in:

→ parole e caratteri riservati del linguaggio (public, for, while, =, etc.);
→ numeri;
→ stringhe, cioè sequenza di caratteri;
→ identificatori, definiti dal programmatore e utilizzati per identificare elementi del programma quali classi, variabili, funzioni etc.

Tutti gli identificatori (classi, variabili etc.) devono rispettare le seguenti regole in modo da non sollevare errori sintattici in fase di compilazione:

→ non devono iniziare con numeri;
→ non devono essere presenti spazi;
→ sono ammesse solo lettere, numeri e i soli caratteri _ e $ e nessun altro carattere è ammesso;
→ non usare parole riservate;
→ due identificatori con lo stesso scope non possono avere lo stesso nome.

2) Variabili in C/C++

Le variabili sono locazioni di memoria (una riga della ipotetica tabella) utilizzate per salvare dei dati. Ogni variabile deve essere definita con:

→ un identificatore, il suo nome, necessario per farvi riferimento;
→ un tipo, per definire la tipologia di variabile che andremo a memorizzare (intero, stringa etc.);
→ uno scope, che mi dice in che area del programma è utilizzabile.

3) Quali sono i tipi di variabile in C/C++

I tipi di variabile in C/C++ sono:

→ puntatori/riferimenti;
→ array: tipo aggregato;
→ int: da -2^{31} a $2^{31}-1$;

→ float: numeri floating;
→ enum: archivia uno dei valori del set di enumerazione definito da tale tipo;
→ struct: aggregato costituito da un insieme di campi che possono essere anche di tipo diverso.

4) Cosa sono gli array in C/C++?

→ Un array è una sequenza di variabili di tipo omogeneo (cioè dello stesso tipo) e distinguibili l'una dall'altra (indirizzabili) in base alla loro posizione all'interno della sequenza.
→ In alcuni linguaggi di programmazione (come Java) l'array è un tipo di oggetto ma è possibile considerarlo come una collezione di variabile dello stesso tipo.

12	23	34	45	56	67	78	89	96

5) Stringhe in C/C++

In C standard una stringa non è un tipo di dato predefinito ma un array di caratteri. L'array contiene come ultimo elemento il carattere terminatore '\0'.

```
char s[10];
```

G	i	a	n	l	u	c	a	\0

In C++ una stringa può essere una variabile non primitiva, ovvero un'istanza della classe string.

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    string s1 = "Ciao";
}
```

6) Strutture in C/C++

Le strutture in C/C++ sono aggregazioni di dati di tipo diverso.
Esempio:

```
struct esempio {
    int esempio1;
    char esempio2;
    float esempio3;
    int esempion;
};
```

Per accedere ad un campo è necessario utilizzare la dot notation.
Forma

```
variabile.camporecord
```

Esempio:

```
esempio e;
e.esempio1;
```

7) Puntatori in C/C++

Un puntatore è un tipo di variabile che contiene, al posto del dato, l'indirizzo della locazione di memoria in cui è memorizzato il dato stesso. I puntatori sono un tipo di dato che rappresentano gli indirizzi dei vari elementi del programma.
Una variabile di tipo puntatore è dichiarata nel modo seguente:

```
<tipo>* <puntatore>;
```

L'operatore * è chiamato operatore di de indirizzamento

Esempi di dichiarazione dei puntatori:

```
int* p;
float* p2;
double* p3;
studente* p4;        //studente è una struct
```

Altri esempi di dichiarazione dei puntatori:

```
int * p;
float * p2;
double * p3;
studente * p4;       //studente è una struct
```

Oppure:

```
int *p;
float *p2;
double *p3;
studente *p4;        //studente è una struct
```

Attenzione: Se si dichiarano più puntatori dello stesso tipo bisogna fare attenzione all'operatore *, il quale deve essere preceduto.
Con la seguente dichiarazione:

```
int *p1,p2;
```

sia ha che p1 è un puntatore ad un intero mentre p2 è una semplice variabile intera. La sintassi della dichiarazione di due puntatori dello stesso tipo è:

```
int *p1,*p2;
```

In C/C++ l'operatore che fornisce l'indirizzo di una variabile è: &
La sintassi di assegnazione è la seguente:

```
int i;
int* p1 = &i;
int* p2;
p2 = p1;
```

In C/C++ non è ammessa l'operazione di assegnamento a una variabile puntatore l'indirizzo di una variabile di tipo diverso da quello del puntatore:

```
int i;
float* p;
p = &i;            //Errore! Non è possibile convertire da int* a float* nella dichiarazione
```

Il puntatore ha la seguente forma:

0x22FF74 → Indirizzo esadecimale (base 16).

È possibile assegnare a una variabile puntatore anche un indirizzo assoluto (fisico) di memoria, ma solo previo casting:

```
int *p;
p = (int*) 0x22ff74;
```

Senza il casting il valore 0x22ff74 sarebbe interpretato come un valore numerico e quindi si avrebbe un errore di compilazione "non è possibile convertire da **int** a **int***".

8) Aritmetica dei puntatori

In C/C++ è possibile eseguire operazioni aritmetiche su puntatori. Le operazioni ammesse sono l'addizione (+), sottrazione (-), incremento (++), decremento (--) e confronto (==).

Più precisamente:

→ al valore del puntatore è possibile sommare o sottrare esclusivamente dei valori interi;

→ la somma e la differenza tra un puntatore e un valore intero viene intesa come lo spostamento logico del puntatore di un numero di byte pari al prodotto del numero intero per la dimensione dell'oggetto puntato.

Esempio:

```
int i;          //inidirizzo i = 0x1000
int* p = &i;    //p = 0x1000
p = p + 2;      //p = 0x1000 + (2 * 4) = 0x1008
```

Incremento (++) e decremento (--) del valore di un puntatore: il valore viene incrementato o decrementato del numero di byte dell'oggetto puntato.

Attenzione alle priorità (* e ++/--)

Nel caso dell'operazione *p++ viene incrementato il valore del puntatore e non dell'oggetto puntato.

L'istruzione corretta è la seguente: (*p) ++ o (*p) +1

Le operazioni di confronto dei puntatori sono:

→ == e! = che testano se i due puntatori fanno riferimento o meno alla stessa locazione di memoria;

→ (>, <,>=, <=) controllano la posizione reciproca delle locazioni di memoria.

9) Array e puntatori

L'array rappresenta un indirizzo che assume il ruolo di un puntatore costante (cioè non modificabile) al primo elemento dell'array, cioè quello di indice 0.

Esempio:

```c
#include <stdio.h>

int main() {
  int V[10];
  int i = 0;

  for (i = 0; i < 10; i++)
    V[i] = i + 1;

  printf("V = %p", V);
  printf("\n(*V) = %i", *V);
  printf("\n(V + i) = %p", V + i);
  printf("\n*(V + i) = %i", *(V + i));

  return 0;
}
```

Output del programma:

```
V = 0061FEF4
(*V) = 1
(V + i) = 0061FF1C
*(V + i) = 10
```

Il nome dell'array rappresenta anche un puntatore al primo elemento della struttura, quando si passa un array come parametro a una funzione si passa alla funzione solo un indirizzo e quindi si effettua un passaggio per indirizzo. Ciò migliora anche l'efficienza del codice, visto che sullo stack vengono allocati soltanto 4 byte.

Esempio:

```c
#include <stdio.h>

const int N = 3;

int main() {
  int V[N] = {2, 7, 9};

  printf("*V = %i", *V);
  printf("\n*V+1 = %i", *V+1);
  printf("\n*(V + 1) = %i", *(V + 1));
  printf("\nV = %p", V);
  printf("\nV + 1 = %p", V + 1);

  return 0;
}
```

Output

```
*A: 2
*A+1: 3
*(A+1): 7
A: 0x6ffe00
A+1: 0x6ffe04
Premere un tasto per continuare . . .
```

10) Funzioni in C/C++

Un programma C/C++ è costituito da un insieme di funzioni. La funzione main rappresenta il punto d'inizio. Una funzione viene definita nel seguente modo:

```
<result type> <function name> ( <arg>* ) {
    <declaration>*
    <statement>*
}
```

Ad esempio:

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

Considerando la funzione che calcola il fattoriale di un numero:

```
int fact(int n) {
    if (n == 0)
        return 1;
    else
        return n * fact(n - 1);
}

int fact(int n) {
    n == 0 ? 1 : n * fact(n - 1);
}
```

In questo caso i parametri vengono passati per valore (viene passato il contenuto della variabile nel record di attivazione). All'interno di una funzione vi possono essere anche variabili dichiarate e tali variabili vengono chiamate variabili locali. Esse possono essere inizializzate.

Esempio

```
int fact_i(int n) {
    int i, acc = 1;

    for (i = 1; i < n; i++) acc *= i;
    return acc;
}
```

Tutti i parametri (tranne array, riferimenti, oggetti, ecc.) vengono passati ad una funzione per valore.

Esempio 1

```
int x = 3;
int y = 42;

swap(int* a, int* b) {
    int* tmp = *a;
    *a = *b;
    *b = *tmp;
}
```

La chiamata

```
swap(&x, &y);        //scambia i valori di x e y
```

Esempio 2

```
int x = 3;
int y = 42;

swap(int& a, int& b) {
    int tmp = a;
    a = b;
    b = tmp;
}

swap(x, y);      //scambia i valori di x e y
```

11) Passaggio parametri di array

In C/C++ gli array vengono passati ad una funzione per riferimento. Un parametro di tipo array converte un'espressione di tipo T [] (array) ad un puntatore (indirizzo di memoria) al primo elemento.

Esempio:

```
int i[3] = {1, 0, 0};
int v[3] = {1, 2, 3};

int vsmult(int a[], int b[]) {
    int result = 0;

    for (int i = 0; i < 3; i++)
        result += a[i] * b[i];

    return result;
}
```

La chiamata:

```
int res = vsmult(i,v);                    //deposita 1 in res, senza copiare in a e b gli interi arrays
```

12) Passaggio parametro di una struttura

Il passaggio dei parametri per riferimento (puntatore o indirizzo di memoria) è utile anche per le strutture. Una struttura ha una dimensione di almeno 164 ottetti (bytes di 8 bits). Risulta necessario l'utilizzo dei puntatori per poter manipolare il valore di una variabile di tipo struttura.

Esempio

```
struct persona {
    char nome[80];
    int eta;
    char citta_nascita[80];
}
p = {"Gianluca Rota", 20, "Bergamo"};
```

Sia il metodo

```
void stampaPersona(struct persona* p) {
    printf("<%s, %i, %s>", (*p).nome, p->eta, p->citta_nascita);
}
```

La chiamata

```
stampa_persona(&p);
```

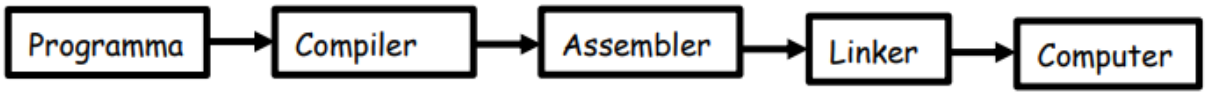
produce in output:

```
<Gianluca Rota, 20, Bergamo>
```

13) Definizione e caratteristiche di compilatore

Il compilatore è un programma software che traduce le istruzioni di un linguaggio di programmazione ad alto livello in linguaggio macchina. Esso salva le nuove istruzioni in memoria (crea file.exe). Il compilatore traduce prima il codice poi lo esegue; per tale motivo offre una traduzione più veloce ma l'eseguibile creato (esempio.exe) sarà utilizzabile solo sul calcolatore su cui il codice è stato compilato e quindi non sarà portatile.

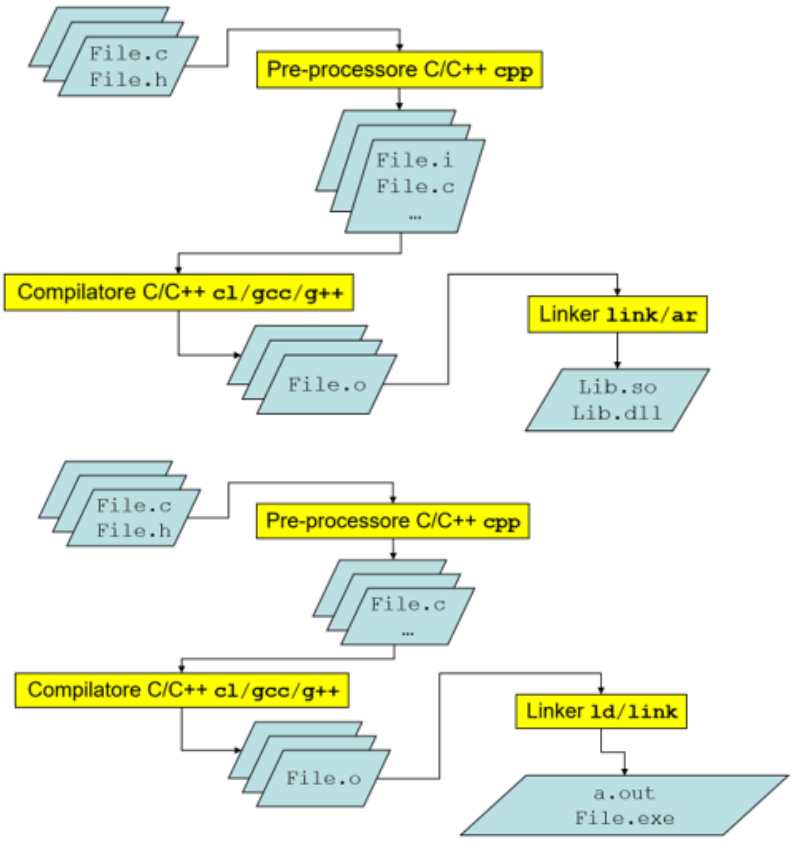
14) Catena programmativa



15) Cosa si intende per preprocessore?

Il preprocessore o precompilatore è un programma che effettua sostituzioni testuali sul codice sorgente di un programma, ovvero la precompilazione. Le direttive nel file di origine indicano al preprocessore di eseguire azioni specifiche. Ad esempio, il preprocessore può sostituire i token nel testo oppure inserire il contenuto di altri file nel file di origine.

Processo di compilazione programmi C/C++



16) Come viene effettuata la compilazione dei programmi C/C++

La compilazione dei programmi C viene effettuata mediante il seguente comando

```
C:\Users\gianl>gcc prova.c
```

Il codice sorgente è di solito distribuito su un insieme di files e directory. I programmi C/C++ si basano sulla distinzione tra:

→ header files (con estensione .h o .hpp);

→ file di implementazione (con varie estensioni: .c, .cc, .C, .cpp).

Osservazione: è possibile chiamare, oltre al compilatore, il preprocessore e il linker mediante delle estensioni.

Preprocessore: `C:\Users\gianl>gcc -E prova.c`

Linker: `C:\Users\gianl>gcc -c prova.c`

17) Quali sono le tre tipologie di direttive?

Le tre tipologie di direttive sono:

→ inclusione di testo;

→ definizione di macro;

→ condizionali.

Esempio: Direttiva di inclusione

```
#include <file.h>

//oppure

#include "file.h"
```

include tutto il file nel sorgente

Esempio: Direttiva di definizione

```
#define PI 3.14L

//oppure

#define max(x, y) ((x) < (y) ? (y) : (x))
```

Esempio: Direttiva di condizione

```
#ifdef PI
//...
#else
//...
#endif

//oppure
#ifndef PI
//...
#else
//...
#endif
```

18) Descrivere le caratteristiche della compilazione separata e "header" files

Ogni programma di una certa dimensione dovrebbe essere modularizzato in maniera appropriate e la modularizzazione di un programma corrisponde all'operazione di compilazione separata. Il risultato della compilazione è un file "oggetto" contente dei riferimenti irrisolti a codice non direttamente disponibile. Il linker ha il compito di risolvere questi riferimenti e, nel caso, segnalare degli errori qualora qualche riferimento rimanga irrisolto.

Esempio: considerando il seguente file usof.c

```
#include <stdio.h>

int f(int);

int main() {
  printf("f(42) = %i", f(42));
}
```

```
C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\2_Anno\Attivita_Didattica\PrimoSemestre\Linguaggi_Programmazione\Esempi\C_C++>gcc usof.c
C:\Users\gianl\AppData\Local\Temp\ccoKmj5m.o:usof.c:(.text+0x16): undefined reference to `f'
collect2.exe: error: ld returned 1 exit status
```

Questo errore viene segnalato dal linker mentre cerca di creare il file esegubile.

Definisco f nel file deff.c

```c
#include <stdio.h>

int f(int x) {
  return x == 42 ? 1 : 0;
}
```

Compilazione

```
C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\2_Anno\Attivita_Didattica\PrimoSemestre\Linguaggi_Programmazione\Esempi\C_C++>gcc deff.c
c:/mingw/bin/../lib/gcc/mingw32/6.3.0/../../../libmingw32.a(main.o):(.text.startup+0xa0): undefined reference to `WinMain@16'
collect2.exe: error: ld returned 1 exit status
```

Il seguente errore di compilazione può essere risolto mediante l'utilizzo di -c.

```
C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\2_Anno\Attivita_Didattica\PrimoSemestre\Linguaggi_Programmazione\Esempi\C_C++>gcc -c deff.c

C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\2_Anno\Attivita_Didattica\PrimoSemestre\Linguaggi_Programmazione\Esempi\C_C++>gcc -c usof.c

C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\2_Anno\Attivita_Didattica\PrimoSemestre\Linguaggi_Programmazione\Esempi\C_C++>
```

Compilazione ed Esecuzione

```
C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\2_Anno\Attivita_Didattica\PrimoSemestre\Linguaggi_Programmazione\Esempi\C_C++>gcc deff.o usof.o

C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\2_Anno\Attivita_Didattica\PrimoSemestre\Linguaggi_Programmazione\Esempi\C_C++>a
f(42) = 1
```

19) Quali sono le regole per definizioni e dichiarazioni distribuite su files multipli?

Le regole per definizioni e dichiarazioni distribuite su files multipli e no sono:

→ tutte le dichiarazioni di variabili e funzioni devono essere consistenti per tipo;

→ ogni oggetto può essere definito una volta sola (in caso contrario il linker segna un errore).

Un modo di separare l'interfaccia di un "modulo" (foo in questo caso) dalla sua implementazione, evitando al tempo stesso il pericolo di ridefinizioni incontrollate consiste nell'usare attentamente gli header files.

Esempio:

```c
#include <stdio.h>
#include "deff.c"

int main() {
  printf("f(42) = %i", f(42));
}
```

```c
#include <stdio.h>

extern int f(int x) {
  return x == 42 ? 1 : 0;
}
```

Compilazione ed Esecuzione

```
C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\2_Anno\Attivita_Didattica\PrimoSemestre\Linguaggi_Programmazione\Esempi\C_C++>gcc usof.c

C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\2_Anno\Attivita_Didattica\PrimoSemestre\Linguaggi_Programmazione\Esempi\C_C++>gcc usof.c -o f.out

C:\Users\gianl\OneDrive\Documenti\Uni_Mi_Bicocca\CDL\2_Anno\Attivita_Didattica\PrimoSemestre\Linguaggi_Programmazione\Esempi\C_C++>f.out
f(42) = 1
```

20) Cosa si intende per libreria?

È un file in un particolare formato che può essere manipolato dal linker, che può essere statico o dinamico. La si può definire anche come una collezione di file oggetto con un indice associato che permette al linker (od al programma in esecuzione) di andare a caricare il codice corrispondente ad un dato "entry point".

Esempio estensione: .so in Unix/Linux, .dll in Windows.

21) Memoria dinamica in C/C++

I linguaggi C/C++ obbligano a gestire esplicitamente la memoria dinamica (free store o heap in C/C++). In C la memoria dinamica viene allocata e de-allocata usando la coppia di funzioni malloc e free (e derivati).

Esempio codice C di gestione della memoria:

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

int main() {
  int* p = (int*) malloc(10 * sizeof(int));
  int i;

  *p = 1;
  *(p + 1) = 42;
  *(p + 2) = 0;

  for (i = 0; i < 10; i++)
    printf("%d\n", *(p + i));

  free(p);
}
```

Output

```
1
42
0
1919381362
1766223201
678651244
691419256
1547322173
1735357008
544039282
```

In altri linguaggi di programmazione di più alto livello con garbage collection, tipo Java, l'istruzione free sarebbe una no – op. Le caratteristiche della funzione malloc sono:
→ restituisce un puntatore di tipo void* ad una zona di memoria nel free store;
→ necessità di casting esplicito per evitare problemi di compilazione;
→ le dimensioni del blocco di memoria ritornato dipendono dal parametro passato alla funzione;
→ se non vi è memoria disponibile, malloc restituisce un puntatore nullo e quindi bisogna sempre controllare il risultato di un'allocazione nel free store.
In C++ vengono utilizzati gli operatori new e delete per manipolare il free store.
La versione in C++dell'esempio precedente è

```
#include <iostream>

using namespace std;

int main() {
  int *p = (int*) new int[10];

  *p = 1;
  *(p + 1) = 42;
  *(p + 2) = 0;

  for (int i = 0; i < 10; i++)
    cout<< *(p + i) << endl;

  delete[] p;
}
```

Output

```
1
42
0
0
0
0
0
0
0
0
```

22) Quale significato hanno i due modificatori di dichiarazione in C/C++?
I due modificatori di dichiarazione in C/C++ hanno il seguente significato:
→ extern: la dichiarazione seguente ha una definizione non locale, ovvero la definizione dell'oggetto dichiarato si trova più in là nel file od in un altro file e viene essenzialmente utilizzata per dichiarazioni da mettere in un file di

interfaccia;
→ static: la dichiarazione o definizione seguente viene "fissata" nello spazio di memoria globale e non è visibile al di fuori del file in cui essa appare. Viene utilizzata soprattutto per definizioni globali in un file.

23) Quali sono le caratteristiche del modificatore static?

Il modificatore static di fatto fissa la dichiarazione o definizione seguente nello "scope" che la racchiude. Le variabili dichiarate static mantengono il loro valore tra una chiamata di funzione e la successiva.

Esempio:

```
int foo() {
    static int x = 42;
    return x++;
}
```

Output

```
42, 43, 44, 45, ..
```

24) Costanti: caratteristiche

Le costanti in C/C++ hanno le seguenti caratteristiche:
→ devono essere inizializzate (a meno che non siano dichiarate extern);
→ non sono modificabili;
→ quando si usa un puntatore vi sono due oggetti da considerare: il puntatore e l'oggetto puntato. La dichiarazione const deve poter distinguere tra i due;
→ possono essere utilizzate per non permettere la modifica degli array passati come parametri alle funzioni;
→ risultano utili in alternativa a #define.

Esempi:

```
char* p;
char  s[] = "Ford Prefect";

const char* pc = s;
pc[3] = 'X';              // Errore.
pc = p;

char *const cp = s;
cp[3] = 'Y';
cp = p;                   // Errore.

const char *const cpc = s;
cpc[3] = 'Z';             // Errore.
cpc = p;                  // Errore.
```

```
const int qd = 42;
const char s[10] = "qwerty";
const double pi  = 3.14L;
```

Un importante utilizzo di const è il seguente

```
char* strcpy(char* p, const char* q);
// Gli elementi di q non si possono modificare.
```

25) Streams, I/O su file

Le librerie di I/O in C/C++ sono molto legate alla piattaforma sottostante, ed in particolare al "file system". In C tutte le operazioni di I/O coinvolgono streams (di solito associati a files). Nella libreria C (in stdio.h) esistono tre streams fondamentali

→ stdin (standard input);

→ stdout (standard output);

→ stderr (standard error).

La gestione dell'input in C viene effettuata mediante tre funzioni principali:

→ `int fgetc(FILE* istream)`: legge un carattere da istream e lo restituisce;

→ `char* fgets(char* s, int n, FILE* istream)` : legge al più n caratteri da istream nella stringa s e la restituisce; se c'è un newline o se istream è alla fine (end of file) l'operazione di lettura si ferma; un puntatore nullo viene restituito in caso di errore;

→
```
int fscanf(FILE* istream,
           const char* format,
           …)
```

legge dati formattati da un flusso e restituisce il numero di campi convertiti e assegnati correttamente e se il valore restituito è pari a 0 indica che nessun campo è stato assegnato.

La funzione scanf è simmetrica a printf.

```
scanf("%d", &x) === fscanf(stdin, "%d", &x)
```

Le funzioni di output in C più semplici sono le seguenti:

→ `int fputc(int c, FILE* ostream)` : scrive il carattere c (notare la conversione) su ostream

→ `int fputs(const char* s, FILE* ostream)` : scrive la stringa s su ostream

→ `int fprintf(FILE* ostream, const char* format, …)` :scrive la stringa format su ostream dopo averla "interpretata" sulla base degli argomenti (in numero variabile) forniti.

Nella libreria C++ (in iostream) esistono tre streams fondamentali

→ std :: cin standard input;
→ std :: cout standard output;
→ std :: cerr standard error.

Per quanto riguarda l'input in C++, viene utilizzato >> per leggere un valore dallo stream, mentre per quanto riguarda l'output in C++ viene utilizzato << per scrivere un valore sullo stream. In C, per associare uno stream ad un file si usa la funzione fopen.

```
FILE* fopen(const char* filename, const char* mode);
```

Il parametro filename è il nome (completo) del file da "aprire" mentre mode indica la modalità di apertura di un file che può essere:

→ "r": apertura file di testo;
→ "w": apre azzerando, o crea un file di scrittura;
→ "a": apre un file in modalità append, apre o crea un file di testo in scrittura (alla fine del file). Esso ritorna un puntatore ad uno stream FILE od il puntatore nullo se viene segnalato un qualche errore.

In C, per spezzare l'associazione uno stream ad un file si usa la funzione fclose.

```
int fclose(FILE* stream);
```

Segnala semplicemente al file system che il file associato a stream non verrà più utilizzato dal programma e ritorna 0 se va a buon termine, altrimenti EOF.

In C++, l'input e l'output si basa sulla costruzione delle istanze delle classi ifstream e ofstream (definite in <iostream.h>). In quanto file sono streams, le operazioni di lettura e scrittura vengono effettuate mediante gli operatori >> (input) e << (output).

26) Significato della keyword typedef

La keyword typedef rappresenta una direttiva che ha lo scopo di assegnare dei nomi alternativi a dei tipi di dato esistenti per rendere il codice riutilizzabile più facilmente tra un'implementazione e un'altra.

Esempi:

```
typedef char buffer[1024];
buffer x;    /* x e` dichiarata di fatto come char x[1024] */

struct _person { char* name; int age; char coll[80]; };
typedef struct _person* Person;
Person p = (Person) malloc(sizeof struct _person);

p->age = 42;/* p e` di fatto un puntatore a struct _person */
```

27) Implementazione code con priorità in C/C++

Una coda con priorità può essere implementata mediante un array ordinato o meno, gravando sulla complessità delle operazioni elementari insert ed extract. Gli heap vengono implementati in tempo logaritmico.

```
typedef int (* pq_compare) (void*, void*);
typedef struct _pq {
     void** pq;
     int size;
     int count;
     pq_compare compare;
} * pq_priority_queue;

pq_priority_queue
pq_new_priority_queue(pq_compare cf, int size);

void  pq_insert(pq_priority_queue, void*);
void* pq_extract(pq_priority_queue);
int   pq_count(pq_priority_queue);
bool  pq_is_empty(pq_priority_queue);
```

```
pq_priority_queue
pq_new_priority_queue(pq_compare cf, int size) {

     pq_priority_queue pq
        = (pq_priority_queue) malloc(sizeof(struct _pq));

     /* Error checking here… */

     pq->pq = malloc(size * sizeof(void *));

     /* Error checking here… */

     pq->compare = cf;
     pq->count   = 0;
     pq->size    = size;
     return pq;
}
```

```
void* pq_extract(pq_priority_queue pq) {
     int e = 0;
     int i;
     for (i = 0; i < pq->count; i++)
            if (pq->compare(pq->pq[i], pq->pq[e]))
                  e = i;
     exchange(pq->pq, e, pq->count - 1);
     return pq->pq[--(pq->count)];
}
```

28) Differenza tra const e define

La differenza tra const e define è che la prima la usi esclusivamente per definire un dato, la seconda puoi usarla per definire qualunque cosa, pure del codice sotto forma di macro.