Laurea in Informatica
A.A. 2021-2022

Corso "Base di Dati"

Accesso a PostgreSQL da software

Prof. Massimiliano de Leoni

Università degli Studi di Padova

UNIVERSITÀ DEGLI STUDI DI PADOVA

1° Esempio: Stampa il numero di tuple di una tabella T

```
#include <iostream.h>
#include <libpq-fe.h>

void do_exit(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int main()
{
    PGconn *conn = PQconnectdb("dbname=testdb");                    //Connessione al database

    if (PQstatus(conn) == CONNECTION_BAD)                           //Se non è possibile connettersi
    {
        cerr << "Connection to database failed:" << PQerrorMessage(conn);
        do_exit(conn);
    }

    PGresult *res = PQexec(conn, "SELECT COUNT(*) FROM T");         //Esegui una query sulla connessione

    if (PQresultStatus(res) != PGRES_TUPLES_OK)                     //Se ci sono stati problemi
    {
        cerr << "Non è stato restituito un risultato" << PQerrorMessage(conn);
        PQclear(res);
        do_exit(conn);
    }

    cout << PQgetvalue(res, 0, 0);                                  //Estra il risultato e stampa su stdout

    PQclear(res);                                                   //Chiudi la connessione
    PQfinish(conn);

    return 0;
}
```

UNIVERSITÀ DEGLI STUDI DI PADOVA

Apertura della connessione:

`PGconn *PQconnectdb(char *connInfo);`

- La **funzione `PQconnectdb`** apre la connessione con il database a partire da un stringa di parametri
- Stringa in due formati:
  - `"host=… port=… dbname=… user=… password=…"`
    - Questi (ed altri) pararametri sono opzionali: se non vengono specificati, su usano i valori di default.
    - La lista completa è nella Sezione 33.1.2 della documentazione
  - `"postgresql://user:password@host:port/dbname`
    - Questi (ed altri) pararametri sono opzionali: se non specificati, si usano valori di default.
    - Esempi:
      - `postgresql://localhost:5432`
      - `postgresql://localhost/testdb`
      - `postgresql://user@localhost`
      - `postgresql://user:secret@localhost/testdb`

UNIVERSITÀ DEGLI STUDI DI PADOVA

1° Esempio: Stampa il numero di tuple di una tabella T

```
#include <iostream.h>
#include <libpq-fe.h>

void do_exit(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int main()
{
    PGconn *conn = PQconnectdb("dbname=testdb");                    //Connessione al database

    if (PQstatus(conn) == CONNECTION_BAD)                    //Se non è possibile connettersi
    {
        cerr << "Connection to database failed:" << PQerrorMessage(conn);
        do_exit(conn);
    }

    […]
}
```

`PQstatus(conn)` restituisce lo stato della connessione tramite due costanti: `CONNECTION_OK` e `CONNECTION_BAD`

`PQerrorMessage(conn)` restituisce l'ultimo messaggio di errore generato

Effettuazione della query:

`PGresult *PQexec (PGconn *conn, const char *command);`

- La **funzione `PQexec`** invia un commando `command` al database tramite la connessione `conn`
- Esempio:

```
PGresult *res = PQexec(conn, "SELECT COUNT(*) FROM T");
```

- Il puntatore di ritorno:
  - è nullo in caso di problem molto seri: mancanza di memoria, di connessione al server, ecc…
  - non è nullo in generale e la **funzione `PQresultStatus`** permette di capire l'esito:
    - il valore `PGRES_TUPLES_OK` indica che sono state restituite tuple (per es. in caso di query)
    - il valore `PGRES_COMMAND_OK` indica che NON sono state restituite tuple (per es. in caso di inserimenti, cancellazioni, ecc.) ma il comando ha funzionato
    - Altri valori indicano un qualche bug, spesso a livello client (per esempio, errore di sintassi della query).

UNIVERSITÀ DEGLI STUDI DI PADOVA

1° Esempio: Stampa il numero di tuple di una tabella T

```
#include <iostream.h>
#include <libpq-fe.h>

void do_exit(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int main()
{
    PGconn *conn = PQconnectdb("dbname=testdb");                    //Connessione al database

    […]

    PGresult *res = PQexec(conn, "SELECT COUNT(*) FROM T");         //Esegui una query sulla connessione

    if (PQresultStatus(res) != PGRES_TUPLES_OK)                     //Se ci sono stati problemi
    {
        cerr << "Non è stato restituito un risultato" << PQerrorMessage(conn);
        PQclear(res);
        do_exit(conn);
    }

    cout << PQgetvalue(res, 0, 0);                                  //Estra il risultato e stampa su stdout

    PQclear(res);                                                   //Chiudi la connessione
    PQfinish(conn);

    return 0;
}
```

UNIVERSITÀ DEGLI STUDI DI PADOVA

Analisi del risultato:
`char *PQgetvalue(const PGresult *res, int row_number, int column_number);`

- La **funzione `PQgetvalue(res,i,j)`** restituisce il valore dello j-esimo attributo per la i-esima tupla.
- La prima tupla è quella con i=0 e il primo attributo è j=0
- Esempio:

```
PGresult *res = PQexec(conn, "SELECT COUNT(*) FROM T");
char *value=PQgetvalue(res, 0, 0);
```

- Il puntatore di ritorno:
  - è sempre una stringa anche se l'attributo è numerico: in tal caso, viene rappresentato come stringa,
  - è una stringa vuota, se il risultato è una stringa vuota ma anche se è il valore è NULL
    - La funzione `int PQgetisnull(PGresult *res, int row_number,int column_number)` permette di distinguere: 1 se null, 0 se non null
    - Per esempio se **`PQgetisnull`**`(res,0,0)`=1 allora il primo attributo della prima tupla ha un valore nullo

UNIVERSITÀ DEGLI STUDI DI PADOVA

1° Esempio: Stampa il numero di tuple di una tabella T

```
#include <iostream.h>
#include <libpq-fe.h>

void do_exit(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int main()
{
    PGconn *conn = PQconnectdb("dbname=testdb");                    //Connessione al database

    […]

    PGresult *res = PQexec(conn, "SELECT COUNT(*) FROM T");         //Esegui una query sulla connessione

    if (PQresultStatus(res) != PGRES_TUPLES_OK)                     //Se ci sono stati problemi
    {
        cerr << "Non è stato restituito un risultato" << PQerrorMessage(conn);
        PQclear(res);
        do_exit(conn);
    }

    cout << PQgetvalue(res, 0, 0);                                  //Estra il risultato e stampa su stdout

    PQclear(res);                                                   //Chiudi la connessione
    PQfinish(conn);

    return 0;
}
```

`PQclear(res)` elimina il risultato dalla memoria

`PQfinish(conn)` chiude la connessione col DB

UNIVERSITÀ DEGLI STUDI DI PADOVA

Come iterare su un insieme di righe

- Il metodo **`PQntuples`**`(res)` permette di ottenere il numero di tuple del risultato
- Ciclo con **`PQgetvalue`**`(res, i, j)` per estrarre i valori

```
int numTuple=PQntuples(res);

   for(int i=0;i<numTuple;i++)
   {
        //Stampo i valori dei primi due attributi di ogni riga
       cout << PQgetvalue(res, i, 0) << "\t" << PQgetvalue(res, i, 1)
       cout << "\n";
   }
```

Esempio: Stampa matricola e data di nascita degli studenti

Matricola	Cognome	Nome	Data di nascita
6554	Rossi	Mario	05/12/1978
…	…	…	…

studenti

```
#include <iostream.h>
#include <libpq-fe.h>

void do_exit(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int main()
{
    PGconn *conn = PQconnectdb("dbname=testdb");

    if (PQstatus(conn) == CONNECTION_BAD)
    {
        cerr << "Connection to database failed:" << PQerrorMessage(conn);
        do_exit(conn);
    }

    PGresult *res = PQexec(conn, "SELECT MATRICOLA, DATA_DI_NASCITA FROM STUDENTI");
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        cerr << "Non è stato restituito un risultato" << PQerrorMessage(conn);
        PQclear(res);
        do_exit(conn);
    }

    int numTuple=PQntuples(res);

    cout << PQfname(res, 0) << "\t" << PQfname(res, 1);
    cout << endl;

    for(int i=0;i<numTuple;i++)
    {
        cout << PQgetvalue(res, i, 0) << "\t" << PQgetvalue(res, i, 1)
        cout << endl;
    }

    PQclear(res);
    PQfinish(conn);

    return 0;
}
```

`PQfname(res,i)`
restituisce il nome dell'attributo relativo alla i-esima colonna

UNIVERSITÀ DEGLI STUDI DI PADOVA

Esempio: Codice che stampa il risultato di una query generica

```
#include <iostream.h>
#include <libpq-fe.h>

int main()
{
    PGconn *conn = PQconnectdb("dbname=testdb");

    […]

    char query[40];

    cout >> "Inserire la query:"
    cin << query;

    PGresult *res = PQexec(conn, query);

    […]

    int numTuple=PQntuples(res);
    int numAttributi=PQnfields(res);

    for(int i=0;i<numAttributi)
        cout << PQfname(res, i) << "\t";
    cout << endl;

    for(int i=0;i<numTuple;i++)
    {
        for (int j=0;i<numAttributi;j++)
            cout << PQgetvalue(res, i, j) << "\t";
        cout << endl;
    }

    PQclear(res);
    PQfinish(conn);

    return 0;
}
```

`PQnfields(res)` restituisce il numero degli attributi (colonne) della relazione

Stampo i nomi di tutti gli attributi, separandolo dagli altri con una tabulazione

Per ogni riga, stampo i valori assegnato ad ogni attributo, separandolo dagli altri con una tabulazione

Esempio: Stampa tutte le informazioni degli studenti

Matricola	Cognome	Nome	Data di nascita
6554	Rossi	Mario	05/12/1978
…	…	…	…

studenti

```
#include <iostream.h>
#include <libpq-fe.h>

void do_exit(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int main()
{
    PGconn *conn = PQconnectdb("dbname=testdb");

    if (PQstatus(conn) == CONNECTION_BAD)
    {
        cerr << "Connection to database failed:" << PQerrorMessage(conn);
        do_exit(conn);
    }

    PGresult *res = PQexec(conn, "SELECT * FROM STUDENTI");
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        cerr << "Non è stato restituito un risultato" << PQerrorMessage(conn);
        PQclear(res);
        do_exit(conn);
    }

    int numTuple=PQntuples(res);

    cout << "Matricola" << "\t" << "Cognome" << "Nome" << "\t" << "Data di nascita";
    cout << endl;

    for(int i=0;i<numTuple;i++)
    {
        cout << PQgetvalue(res, i, 0) << "\t" << PQgetvalue(res, i, 1) << PQgetvalue(res, i, 2) << "\t" << PQgetvalue(res, i, 3)
        cout << endl;
    }

    PQclear(res);
    PQfinish(conn);

    return 0;
}
```

Possibile Problema: Soluzione dipendente dall'ordine con cui gli attributi sono stati definiti o, peggio, il database potrebbe non garantire il rispetto dell'ordine

Soluzione: Usare i nome delle colonne invece delle posizioni

- La funzione int **`PQfnumber`**`(res, column_name)` restituisce la posizione della colonna con nome `column_name` nel risultato `res`.

```
int main()
{
    PGconn *conn = PQconnectdb("dbname=testdb");
    […]
    PGresult *res = PQexec(conn, "SELECT * FROM STUDENTI");
    […]

    int numTuple=PQntuples(res);

    cout << "Matricola" << "\t" << "Cognome" << "Nome" << "\t" << "Data di nascita" << endl;

    for(int i=0;i<numTuple;i++)
    {
        cout << PQgetvalue(res, i, Pqfnumber(res,"Matricola")) << "\t";
        cout << PQgetvalue(res, i, Pqfnumber(res,"Cognome")) << "\t";
        cout << PQgetvalue(res, i, Pqfnumber(res,"Nome")) << "\t";
        cout << PQgetvalue(res, i, Pqfnumber(res,"Data_di_Nascita"));
        cout << endl;
    }

    […]
}
```

Funzione `sprintf`

- La funzione `sprintf(str, format, ...)` costruisce una stringa che viene memorizzata nell'array di caratteri `str`
- Format contiene la stringa che rappresenta il template e contiene i tag di form formattazione, tra cui:
  - `%d` per gli interi,
  - `%f` per i float,
  - `%s` per le stringhe.
- Esempio: `sprintf(query, "Il cognome è %s e la matricola è %d",cognome,numMatr);`
  1. Il valore di cognome (var. stringa) viene messo al posto del primo tag %s
  2. Il valore di matricola (var. intera) viene messo al posto del secondo tag %d
  3. Il risultato viene assegnato a query (puntatore a char).

Esempio: Stampa le informaz. degli studenti con un dato cognome e matricola maggiore di un valore

Matricola	Cognome	Nome	Data di nascita
…	…	…	…

```
#include <iostream.h>
#include <libpq-fe.h>


int main()
{
    PGconn *conn = PQconnectdb("dbname=testdb");

    […]
    char query[80];
    char cognome[30];
    int numMatr;

    cout >> "Inserire il cognome: ";
    cin >> cognome;
    cout >> "Inserire il minimo numero di matricola: ";
    cin >> numMatr;

    sprintf(query, "SELECT * FROM STUDENTI WHERE COGNOME=\'%s\' AND MATRICOLA>%d",cognome,numMatr);

    PGresult *res = PQexec(conn, query);
    if (PQresultStatus(res) != PGRES_TUPLES_OK)
    {
        cerr << "Non è stato restituito un risultato" << PQerrorMessage(conn);
        PQclear(res);
        do_exit(conn);
    }

    int numTuple=PQntuples(res);

    cout << "Matricola" << "\t" << "Cognome" << "Nome" << "\t" << "Data di nascita« << endl;

    for(int i=0;i<numTuple;i++)
    {
        cout << PQgetvalue(res, i, Pqfnumber(res,"Matricola")) << "\t";
        cout << PQgetvalue(res, i, Pqfnumber(res,"Cognome")) << "\t";
        cout << PQgetvalue(res, i, Pqfnumber(res,"Nome")) << "\t";
        cout << PQgetvalue(res, i, Pqfnumber(res,"Data_di_Nascita"));
        cout << endl;
    }

    […]
}
```

`sprintf` permette di costruire una stringa a partire da un template + variabili

Parametri presi da console

Argomenti da linea di comando / 1

- È possibile passare dei parametri da linea di comando

```
C:> nomeprog pippo 12 pluto 25
```

Nome Programma Argomenti

- Possibile aggiungere due parametri al "main":

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

invece di

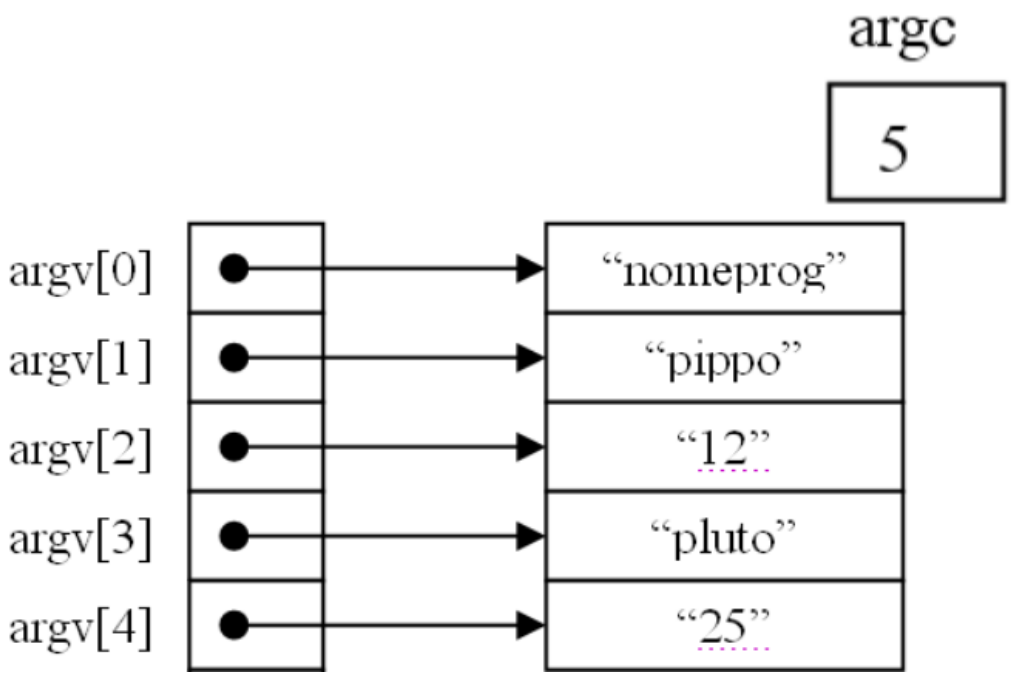
```
int main ()
```

Argomenti da linea di comando / 2

I parametri della funzione main sono 2:

- **int argc:** contiene il numero di stringhe inserite dall'utente a linea di comando (cardinalità del 2° argomento).
- **char* argv[]:** l'array che contiene le stringhe inserite dall'utente a linea di comando (ogni elemento dell'array è un puntatore a carattere.

```
C:> nomeprog pippo 12 pluto 25
```



Esempio: Stampa le informaz. degli studenti con un dato cognome e matricola maggiore di un valore

Parametri presi da console. Es:

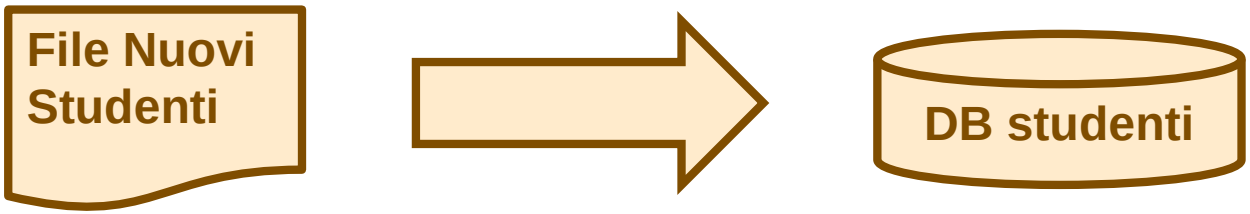
```
C:> nomeprog leoni 1000
```

```
#include <iostream.h>
#include <libpq-fe.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    if (argc<3)
    {
        cerr << "Il numero di argomenti da linea di comando non e' corretto" << endl;
        return(1);
    }

    PGconn *conn = PQconnectdb("dbname=testdb");
    […]

    char query[80];
    char* cognome=argv[1];
    int numMatr=atoi(argv[2]);

    sprintf(query, "SELECT * FROM STUDENTI WHERE COGNOME=\'%s\' AND MATRICOLA>%d",cognome,numMatr);

    PGresult *res = PQexec(conn, query);

    […]

    int numTuple=PQntuples(res);

    cout << "Matricola" << "\t" << "Cognome" << "Nome" << "\t" << "Data di nascita« << endl;

    for(int i=0;i<numTuple;i++)
    {
        cout << PQgetvalue(res, i, Pqfnumber(res,"Matricola")) << "\t";
        cout << PQgetvalue(res, i, Pqfnumber(res,"Cognome")) << "\t";
        cout << PQgetvalue(res, i, Pqfnumber(res,"Nome")) << "\t";
        cout << PQgetvalue(res, i, Pqfnumber(res,"Data_di_Nascita")) << endl;
    }

    […]
}
```

`atoi(stringa)`

converte una stringa ad intero

Esempio: Leggere da file la lista degli student da aggiungere




```
311 Cogn1 Nome1 13/3/2000
556 Cogn2 Nome2 17/6/2000
1056 Cogn3 Nome3 22/12/1999
.....
```

Matricola	Cognome	Nome	Data di nascita
...	...	...	...

UNIVERSITÀ DEGLI STUDI DI PADOVA

Lettura da e scrittura su file in C++ / 1

- Operazioni nella libreria standard fstream
- Per aprire un file `n` in lettura: `ifstream infile(n);`
- Per aprire un file `n` in scrittura: `ofstream outfile(n);`

- Si puo' usare ifstream come cin: `infile >> variabile;`
- Si puo' usare ofstream come cout: `outfile << variabile;`

- In lettura, se `infile >> variabile` restituisce false, si e' raggiunto la fine del file.

- Alla fine dell'uso, i file vanno chiusi con `infile.close();`

Lettura da e scrittura su file in C++ / 2

- L'operazione `infile >> variabile` legge una porzione di linea di testo fino al prossimo spazio.
- Per leggere una intera riga:

  `getStream (infile,variabile)`
- Esempio, legge il contenuto di un file e lo scrive su console (assumendo nessuno spazio nelle righe):

```
{
    […]
    char buffer[100]
    ifstream file("mioFile");

    while(file >> buffer)
          cout << buffer;

    file.close();

    […]
}
```

UNIVERSITÀ DEGLI STUDI DI PADOVA

Esempio: Aggiungere studenti da file (assumendo che i cognomi non hanno lo spazio)

```
#include <iostream.h>
#include <libpq-fe.h>
#include <fstream.h>

ExecStatusType aggiungiStudente(Pgconn *conn, char *matricola, char *cognomenome, char *nome, char *dataNascita
{
    char query[60];
    sprintf(query,
        "INSERT INTO STUDENTI(MATRICOLA,COGNOME,NOME,DATANASCITA) VALUES (%d, \'%s\', \'%s\', \'%s\')",
        matricola, cognome, nome, dataNascita);
    PGresult *res = PQexec(conn, query);
    ExecStatusType risultato=PQexec(conn, query);
    PQclear(res);
    return(risultato);
}

int main(int argc, char* argv[])
{
    char matricola[10], char cognome[50], char nome[50], char dataNascita[10];
    PGconn *conn = PQconnectdb("dbname=testdb");
    […]

    ifstream infile(argv[1]);

    while (infile >> matricola >> cognome >> nome >> dataNascita)
    {
        if (aggiungiStudente(conn, matricola, cognome, nome, dataNascita)!=PGRES_COMMAND_OK)
            cerr << "Impossibile aggiungere la matricola " << matricola << ": " << PQerrorMessage(conn) << endl;
    }
    PQfinish(conn);
    infile.close();
}
```

Prepared Statements / 1

- Per ogni statement (=instruzione) SQL, occorre:
  1. Fare il parsing dello statement
  2. Fare l'ottimizzazione del "query plan" (nel caso di query)
  3. Eseguire lo statement
- Se lo stesso statement deve essere ripetuto, non è necessario ripetere i passi 1 e 2.

- I "prepared statement" permettono di evitare i passi 1 e 2
  - Utili quando la stessa query deve essere ripetuta
  - In un primo passo, lo statement viene "preparato"
  - In un secondo passo, lo statement preparato viene eseguito

Prepared Statements / 2

- `PGresult *PQprepare(PGconn *conn, const char *stmtName, const char *query, int nParams, const Oid *paramTypes)`
  - Si usa per preparare uno statement di nome `stmtName`, dove `query` è lo statement e `nParam` è il numero di parametri
  - La query contiene $1, …, $i che sono i "placeholders" per i valori dei parametri.
  - Ok a lasciare `paramType` sempre uguale `NULL`

Esempio:

```
PQprepare(conn , "aggStudenti",
        "INSERT INTO STUDENTI(MATRICOLA,COGNOME,NOME,DATANASCITA)
VALUES ($1,$2,$3,$4)", 4, NULL);
```

UNIVERSITÀ DEGLI STUDI DI PADOVA

Prepared Statements / 3

- `PGresult *PQexecPrepared(PGconn *conn, const char *stmtName, int nParams, const char * const *paramValues, const int *paramLengths, const int *paramFormats, int resultFormat)`

  Si usa per eseguire un prepared statement di nome `stmtName`, dove `paramValues` contiene la lista dei parametri e `nParams` è il numero di parametri. OK avere sempre:

  - `paramFormat` come un array di zeri, di lunghezza quanto il numero di parametri;
  - `paramLenght = NULL;`
  - `resultForm = 0.`

Esempio:

```
PQexecPrepared(conn , "aggStudenti", 4, parametri , NULL ,
{0,0,0,0}, 0);
```

Esempio PreparedStatement

```
#include <iostream.h>
#include <libpq-fe.h>
#include <fstream.h>

ExecStatusType aggiungiStudente(Pgconn *conn, char *matricola, char *cognomenome, char *nome, char *dataNascita
{
    char* parametri={matricola, cognome, nome, dataNascita};
    ExecStatusType risultato=PQexecPrepared(conn , "aggStudenti", 4, parametri , NULL , {0,0}, 0);
    PQclear(res);
    return(risultato);
}

int main(int argc, char* argv[])
{
    char matricola[10], char cognome[50], char nome[50], char dataNascita[10];
    PGconn *conn = PQconnectdb("dbname=testdb");

    PGresult *stmt = PQprepare(conn , "aggStudenti",
        "INSERT INTO STUDENTI(MATRICOLA,COGNOME,NOME,DATANASCITA) VALUES ($1,$2,$3,$4)", 4, NULL);
    […]

    ifstream infile(argv[1]);

    while (infile >> matricola >> cognome >> nome >> dataNascita)
    {
        if (aggiungiStudente(conn, matricola, cognome, nome, dataNascita)!=PGRES_COMMAND_OK)
            cerr << "Impossibile aggiungere la matricola " << matricola << ": " << PQerrorMessage(conn) << endl;
    }

    infile.close();
}
```

Riferimenti

- Manuale PostgreSQL al Capitolo 33: https://www.postgresql.org/docs/13/libpq.html.