**STREAM**

E’ un flusso di dati da sorgente a ricevente. Noi vedremo gli stream su file o insiemi di dati.

Il cout fa stream in output da programma a console

Il cin fa stream in input da console a programma

* Il se scrivo 2.5 e faccio cin>>intero prende solo fino a prima del punto

Funzioni:

* Cin.clear(); ripulisce lo stato dello stream, utile se si vogliono ignorare errori sul cin
* Cin.ignore(byneruc:kunuts<streanSuze>::max(), “\n”); ignora i caratteri fino a \n (ossia quando si preme invio)

Si usano in coppia.

Si può fare

While(cin>>n){} //continua finchè non si inserisce qualcosa che causi un errore (ad esempio se inserisco un tipo sbagliato)

Altro modo per leggere:

getLine(cin, stringa);

Se faccio cout<<hex<<”22”; scrive 22 in esadecimale

Cout.precision(8); dice quanti decimali deve scrivere (in questo caso 8)

**DA FILE**

OUTPUT

Ofstream ofs(“nomefile\_da\_scrivere.txt”, ios::app); //Cerca il file nella cartella presente dove sta l’exe

ofs<<”Questa roba viene scritta nel file senza cancellare quello che già c’è”; //Se ofs lo avessi aperto con ios::trunc avrebbe cancellato la roba precedente

ofs.close(); //Chiudere sempre lo stream una volta finito

SI PUO’ VEDERE SE UN FILE GIA’ ESISTE? Posso provare ad aprirlo

INPUT

Ifstream ifs(“nomefile.txt”, ios::in);

if(!ifs)

cout<<”File non aperto per problemi”;

cout<<”\t”; fa un tab FUNZIONA ANCHE SULL’OFS? Probabilmente sì

if(ifs2.eof()) //Controlla se la stream è arrivata alla fine del file

if(ifs2.fail()) //Controlla se ci sono errori

ifs2.setState(ios\_base::failbit); //Mette lo stream in fail

ifs2.close(); //chiude lo stream

STREAM DI STRINGHE

Ostringstream oss;

int n = 22;

oss<<”{temperature = ”<<scientific<<n<<”}; //Essendo uno stream posso usare le notazioni (in questo caso scientifica), il vantaggio di usare questo al posto di string è questo

cout<<oss.str();

**OVERLOAD DI OPERATORI DI INPUT E OUTPUT**

Boh, si possono fare.

Esempi carini nel file UserDifenedIOTest.cpp

Oggetto.var =1

Oggetto.var2=3;

Posso scriverlo come

Oggetto={1,3}

**ECCEZIONI**

Estendono la classe exception

Si possono creare delle eccezioni personali estenendo exception.

//Impediscono il crash del programma

COME LE LANCIO?

Try{

Throw variabile; //Lo manda nel catch

//throw “a”;

//throw;

}catch(int variabile){cout<<variabile;}

Catch(char e){}

Catch(…){cout<<”Eccezione generica”;} **NB:** posso scrivere più catch di seguito, il primo che lo prende libera gli errori

Eccezioni già fatte:

try{

throw excpetion();

}catch(exception e){

Cout<<”Eccezione di tipo “<<e.what();

}

**TEMPLATE**

Fanno “ricette” per funzioni o classi

Posso scriverci funzioni che non chiedano un tipo specifico, ma lo lavorino comunque bene

FUNZIONI

Template <typename T> //Al posto di typename posso mettere class se è per classi

T GetMax(T a, T b){

T result;

result = (a>b)? a: b;

return result;

}

**NB:** in questo caso però bisogna che questi oggetti/variabili siano compatibili con l’operatore “>”

Altrimente non funzionerebbe

Posso usare anche più tipi in una funzione:

template <class T, class U>

T GetMin(T a, U b){ //Fare attenzione, ritornerà una variabile del tipo del primo parametro passato

T result;

result = (a<b)? a: b;

return result;

}

CLASSI

Template <class T>

Class myPair{

T a, b;

public:

myPair(T first, Tsecond);

};

**NB:** non sono classi vere e proprie, ha problemi con la standard accoppiata .cpp-.h infatti vengono compilate solo quando il compilatore viene a sapere che prendono determinati tipi di variabile

//1 ora e 10, ascoltare

**LIBRERIE**

#include “”

Roba di XML lol non mi va di seguire è tardi FRA SONO LE DICIANNOVEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE DIO