UNIVERSITA' DEGLI STUDI ROMA TRE

DIPARTIMENTO DI FISICA "E. AMALDI"

Laboratorio di Programmazione e Calcolo

6 crediti

a cura di

Severino Bussino

Anno Accademico 2021-22

0) Struttura del Corso

1) Trattamento dell'informazione
Elementi di Architettura di un Computer

Verra' trattata in una delle prossime lezioni

2) Sistemi operativi

3) Introduzione alla
Programmazione ad oggetti (OO)

4) Simulazione del Sistema Solare

5) Introduzione al linguaggio C/C++

6) Elementi di linguaggio C/C++

A 1 - istruzioni e operatori booleani

2 - iterazioni (`for`, `while`, `do … while`)

B - istruzioni di selezione (`if`, `switch`, `else`)

C - funzioni predefinite. La classe `math`.

A-1) Istruzioni e operatori booleani

Variabili di tipo booleano (1)

- Una variabile booleana può assumere solo due valori: vero o falso (true o false).

```
#include <iostream>
using namespace std;

int main() {
        bool a, b ;

        a = true  ;
        b = false ;

        cout << " valore della variabile a: " << a << endl
             << " valore della variabile b: " << b << endl ;

        return 1;
}
```

Variabili di tipo booleano (2)

- In C++ ogni espressione numerica può essere utilizzata come espressione booleana in quanto il C++ interpreta
  - **zero** come **false**
  - ogni altro valore numerico come **true**
- Il valore restituito e' (vedi programma precedente):

```
valore della variabile a: 1
valore della variabile b: 0
```

Operatori booleani

<	minore di
<=	minore o uguale di
>	maggiore di
>=	maggiore o uguale di
==	uguale a (da non confondere con =)
!=	diverso da

Hanno precedenza su

&&	and logico
||	or logico
!	Not logico

Esempi

- `4 < 5` vale 1 (true)
- `2 >= 3` vale 0 (false)
- `4 == 4` vale 1 (true)

```
double x = 1.5,  y = -1.8;
```

- `x > y` vale 1 (true)
- `x > y + 5` vale 0 (false)
- `(4<5) && 7<6` vale 0 (false)

Esercizi

Valutare le seguenti espressioni booleane

!(4<5)

3 <= 4 && 5 < 7

2 < 1 || 6 < 8

!(5 ==5) && 3<7

Soluzioni

- !(4<5) vale 0 - falso
- 3 <= 4 && 5 < 7 vale 1 - vero
- 2 < 1 || 6 < 8 vale 1 - vero
- !(5 ==5) && 3<7 vale 0 - falso

Meglio controllare!

```
#include <iostream>
using namespace std;

int main() {


    cout << " !(4<5)              vale " <<  !(4<5)  << endl
         << " 3 <= 4 && 5 < 7     vale " << (3 <= 4 && 5 < 7) << endl
         << " 2 < 1 || 6 < 8      vale " << (2 < 1 || 6 < 8 ) << endl
         << " !(5==5) && 3<7      vale " << (!(5==5) && 3<7 ) << endl;


       return 1;
}
```

```
nbseve(~/LabCalA_2)>./provabool2
 !(4<5)              vale 0
 3 <= 4 && 5 < 7     vale 1
 2 < 1 || 6 < 8      vale 1
 !(5==5) && 3<7      vale 0
```

A-2) Iterazioni

(for, while, do … while)

Iterazioni

Tre istruzioni che consentono di eseguire un *loop* (ciclo):

1. for
2. while
3. do...while

con alcune differenze non solo sintattiche…

Problema (1)

Supponiamo di voler calcolare la somma dei primi N numeri pari. Ad esempio, se N=10:

```
S = 2 + 4 + 6 + … + 20
```

Se pero' N e' grande…

```
S = 2 + 4 + 6 + ..... + 2*N
```

(in questo caso esiste una formula analitica, ma supponiamo di non conoscerla)

Problema (2)

Abbiamo necessita' di

- un contatore
  (che conti i numeri pari e ci dica quando fermarci nell'operazione di addizione
- una istruzione che calcoli il numero pari
- una variabile su cui memorizzare le somme parziali ed aggiornare la somma totale

Proviamo!!!

```
#include <iostream>
using namespace std;

int main() {

    int i =  0 ;
    int n = 10 ;
    int p, s   ;

    s =  0     ;
    for(i ; i<n ; i++) {

        p = 2 * (i+1) ;

        s = s +   p   ;

     }

     cout << endl << " La somma dei primi " << n
         << " numeri pari vale " << s << endl << endl ;


     return 1;

}
```

Sintassi di for

```
for (espr1; espr2; espr3) istruzione
```

espr1 espr2 e espr3 sono espressioni C++

istruzione può essere

- una singola istruzione o
- una sequenza di istruzioni racchiusa tra **parentesi graffe**

```
{
    .....
    .....
    .....
}
```

Nell'esecuzione di un'istruzione for viene

1. Valutata l'espressione espr1
   (di solito inizializzazione del contatore: esempio i=0)

2. Valutata l'espressione espr2
   (di solito un test del contatore: esempio i<10)
   - Se non è nulla (VERA) si esegue l'istruzione
   - Se è nulla (FALSA) si passa alle istruzioni successive al ciclo for

3. Valutata l'espressione espr3
   (di solito incremento o decremento del contatore, es. i++)

.....

Problema (1)

Supponiamo di voler calcolare il piu' piccolo numero N tale che la somma dei primi N numeri interi sia maggiore di S.

$$1 + 2 + 3 + \ldots + N > S$$

Cerchiamo **il piu' piccolo valore di N** tale che valga la relazione precedente

Non possiamo usare un ciclo for perche' non conosciamo il numero di iterazioni da effettuare

(anche in questo caso esiste una formula analitica, ma supponiamo di non conoscerla o di utilizzarla per il debug)

Problema (2)

Abbiamo necessita' di

- un controllo che ci dica quando la somma supera il valore S e ci dica quando fermarci nell'operazione di addizione
- una istruzione che calcoli il numero (in questo caso un contatore che conti il numero di cicli)
- una variabile su cui memorizzare le somme parziali ed aggiornare la somma totale

Proviamo!!!

```
#include <iostream>
using namespace std;

int main() {

    int  i    =  0  ;
    int  s    =  0  ;
    int stot = 20. ;

    while(s<=stot) {

        i = i + 1  ;

        s = s + i  ;

     }

     cout << endl << " Per raggiungere un valore maggiore di "
         <<  stot << " e necessario sommare almeno i primi "
         <<   i   << " numeri interi " << endl << endl;


     return 1;

}
```

Sintassi di while

```
while (espressione) istruzione
```

espressione è una qualsiasi espressione C++

istruzione può essere

- una singola istruzione
- una sequenza di istruzioni racchiusa tra **parentesi graffe**

```
{
    .....
    .....
    .....

}
```

Nell'esecuzione di una istruzione while viene

1. Valutata l'espressione espressione
   - Se non è nulla (VERA) si esegue l'istruzione
   - Se è nulla (FALSA) si passa alle istruzioni successive al ciclo while

.....

Confronto tra while e for

```
for (espr1; espr2; espr3) istruzione
```

Si può riscrivere anche come

```
espr1;
while (espr2)  {
   istruzione;
   espr3;
 }
```

Sintassi di do...while

```
do istruzione while (espressione)
```

espressione è una qualsiasi espressione C++

istruzione può essere

- una singola istruzione
- una sequenza di istruzioni racchiusa tra **parentesi graffe**

```
{
    .....
    .....
    .....

}
```

Nell'esecuzione di una istruzione do … while viene

1. Eseguita l'istruzione

2. Valutata l'espressione espressione
   - Se non è nulla (VERA) si torna al punto 1
   - Se è nulla (FALSA) si passa alle istruzioni successive al do … while

……

Esempio di uso di do ... while

```
do {
        cout << "Inserisci  i tuoi anni " ;
        cin >> age;
} while(age <= 0) ;
```

Confronto tra while e do ... while (1)

while

prima valuta la condizione **poi** (se la condizione e' soddisfatta) esegue il blocco istruzioni

do ... while

prima esegue il blocco istruzioni e **poi** valuta la condizione

La condizione potrebbe essere anche calcolata nel blocco istruzioni (e quindi non valutabile in precedenza)

Confronto tra while e do ... while (2)

do ... while esegue sempre
l'istruzione almeno una volta

6) Elementi di linguaggio C/C++

A 1 - istruzioni e operatori booleani

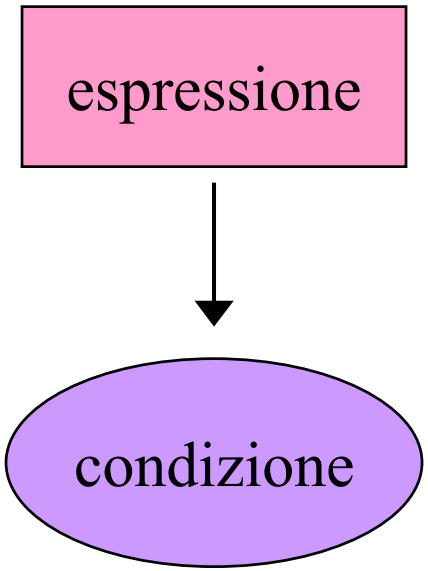
2 - iterazioni (`for`, `while`, `do … while`)

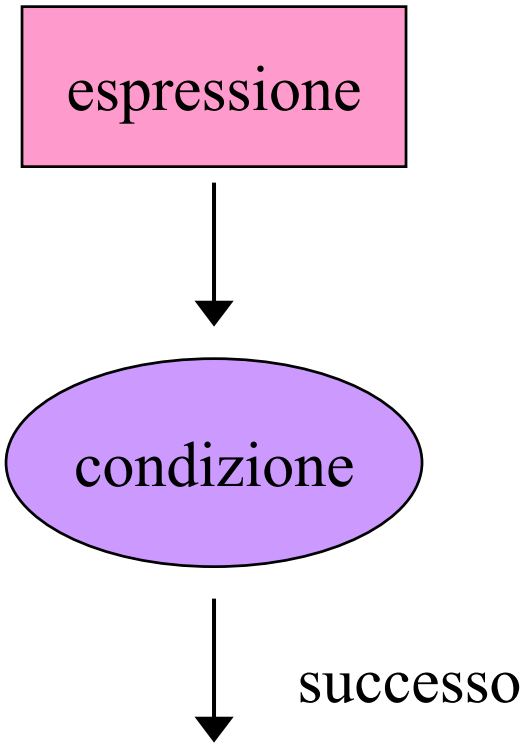
B - istruzioni di selezione (`if`, `switch`, `else`)

C - funzioni predefinite. La classe `math`.

B) Istruzioni di selezione (if, else, elseif)


espressione


espressione
condizione


espressione
condizione
successo

Sintassi di if

if (espressione) istruzione

espressione è una qualsiasi espressione C++

istruzione può essere

- una singola istruzione
- una sequenza di istruzioni racchiusa tra **parentesi graffe**

```
{
.....
.....
.....
}
```

```
if (espressione) istruzione
```

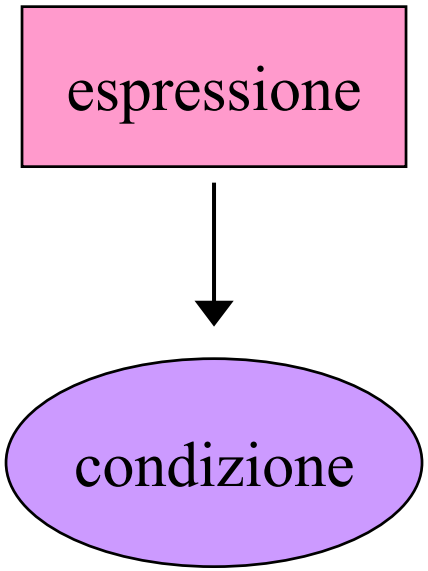
- Nell'esecuzione di un'istruzione if viene valutata l'espressione tra parentesi
  - Se il suo valore è non nullo (VERA), viene **eseguita** l'istruzione
  - Se il suo valore è nullo (FALSA), l'istruzione viene **ignorata**.
- In entrambi i casi l'esecuzione procede con l'istruzione successiva all'if

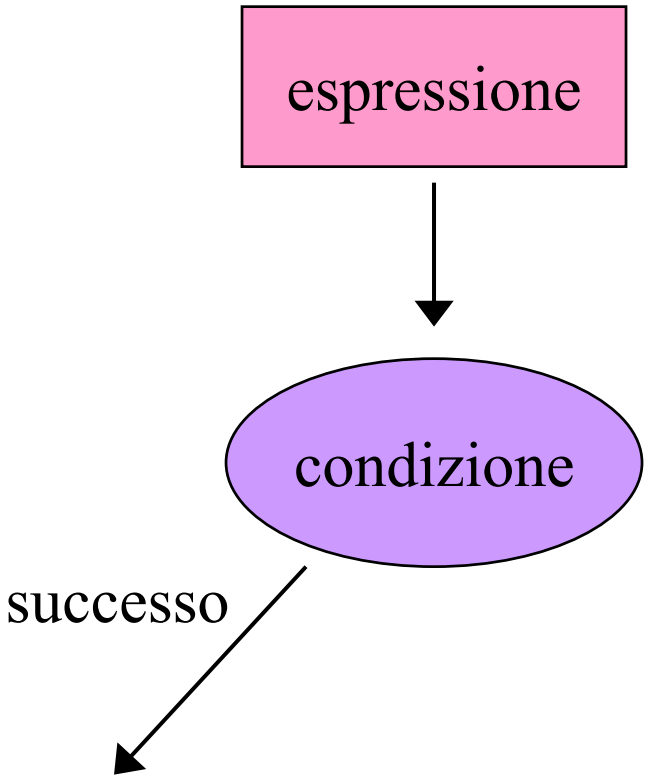
Esempio di uso di if (1)

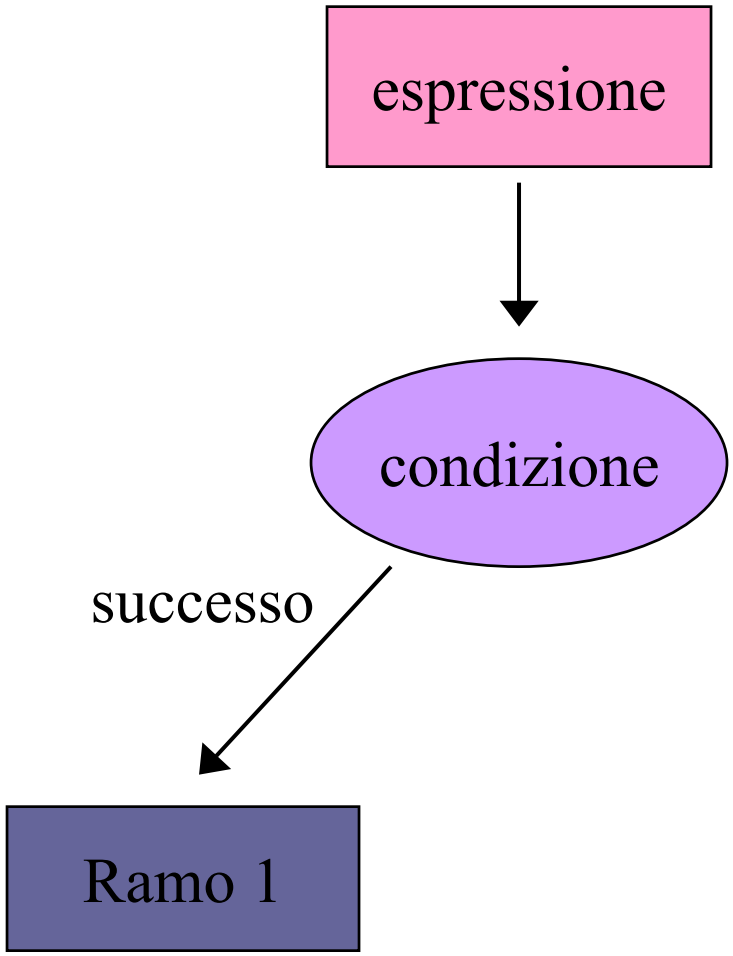
```
do {

        cout << "Inserisci  i tuoi anni " ;
        cin >> age;
        if(age <= 0) {
           cout << "Deve essere un numero positivo! " << endl ;
        }

} while(age <= 0);
```

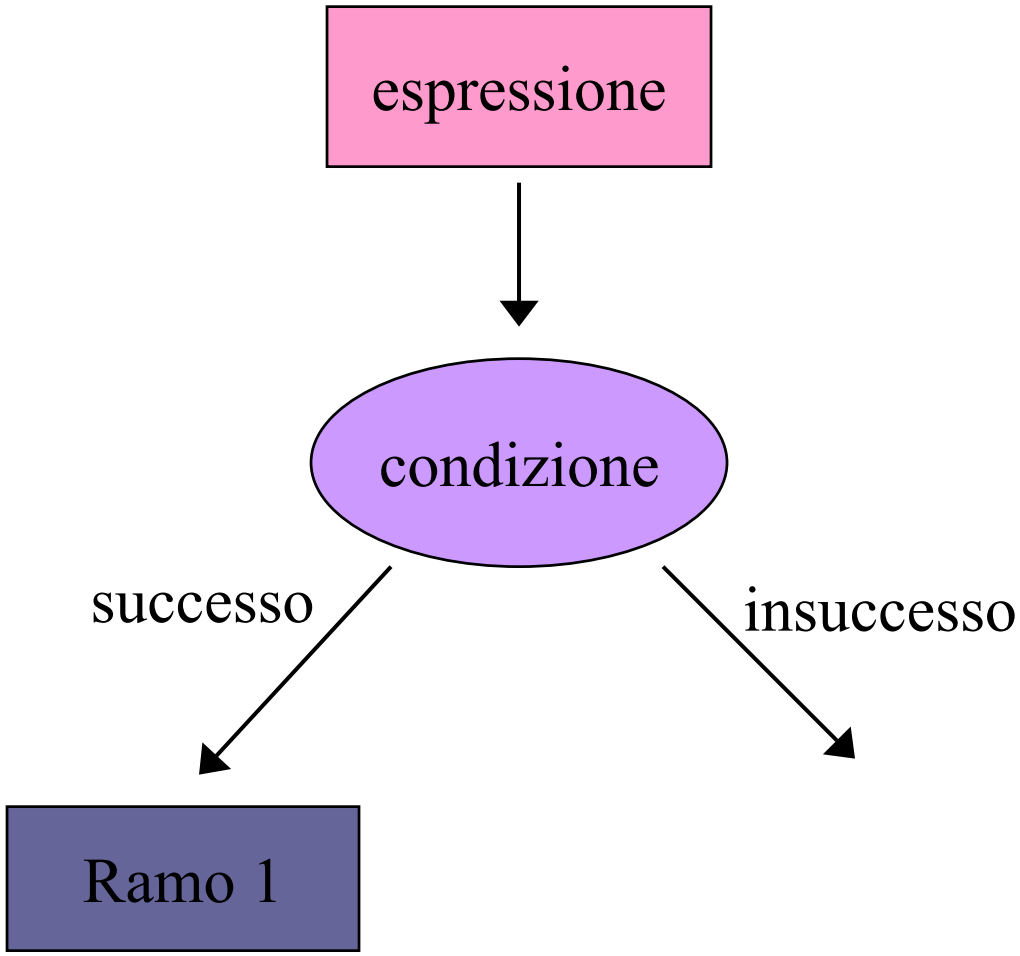
Esempio di uso di if (2)

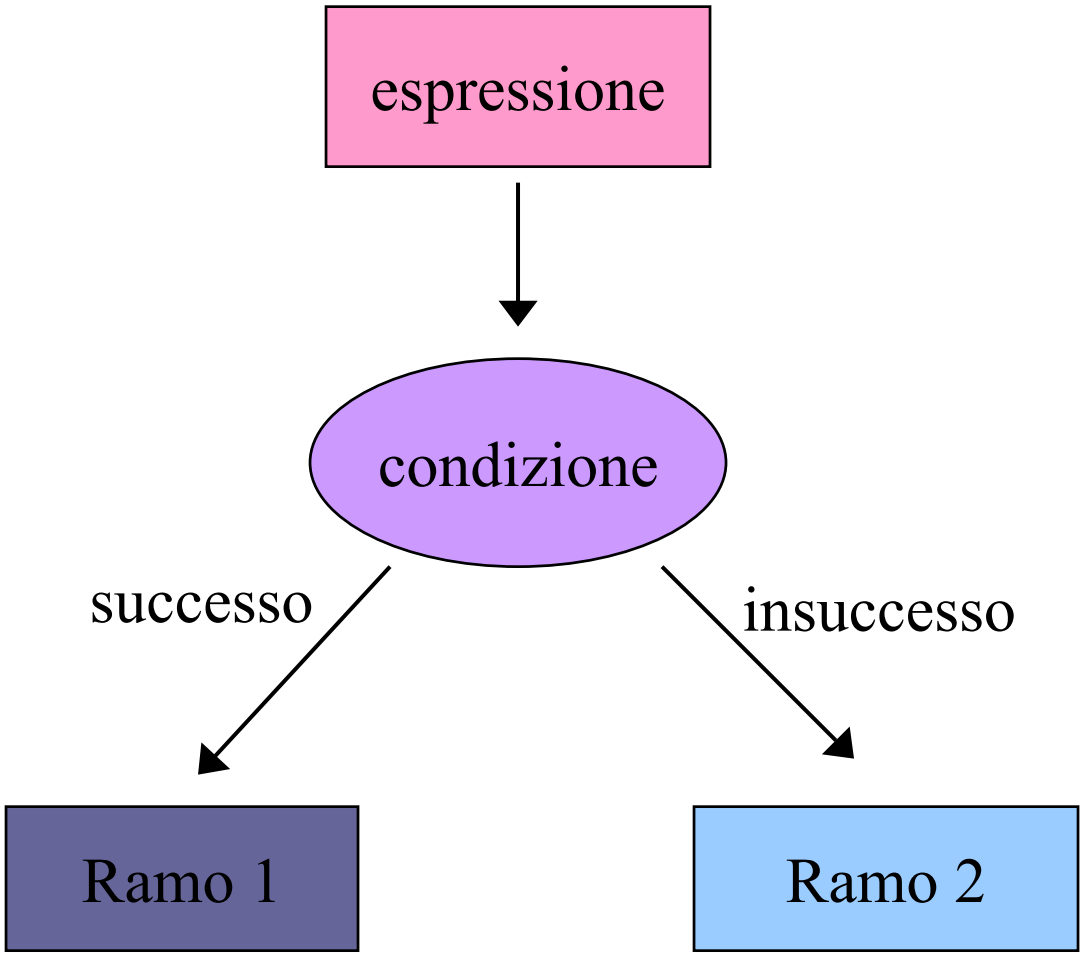
```
#include <iostream>
using namespace std;

int main() {

        double a, b ;

        cout << " Inserire i due numeri : " ;
        cin  >> a >> b ;


        if(a > b) { cout << " Il primo e piu grande."    << endl ; }
        if(a == b){ cout << " I due numeri sono uguali." << endl ; }
        if(a < b) { cout << " Il secondo e piu grande. " << endl ; }

        return 1;

}
```


espressione
condizione


espressione
condizione
successo


espressione
condizione
successo
Ramo 1


espressione
condizione
successo
insuccesso
Ramo 1


espressione
condizione
successo
insuccesso
Ramo 1
Ramo 2

Sintassi di if… else

```
if (espressione) istruzione1 else istruzione2
```

espressione è una qualsiasi espressione C++

istruzione1 e istruzione2

- singole istruzioni
- una sequenza di istruzioni racchiusa tra **parentesi graffe**

```
{
    ..............................
    ..............................
    ..............................

}
```

- Viene valutata l'espressione tra parentesi:
  - Se il valore è non nullo viene eseguita l'istruzione1 e non viene eseguita l'istruzione2
  - Se invece il valore è nullo l'istruzione1 non viene eseguita e viene eseguita l'istruzione2.

Esempi di uso di if … else (1)

```
.....

if ( x != 0 )   {
        y = y / x ;
        x = x - 1 ;
} else   {
        x = 25 ;
        y = y / x ;
}
.....
```

L'istruzione if … else deve essere utilizzata in maniera **ordinata** in modo che il programma sia **leggibile**

Esempi di uso di if … else (2)

```
#include <iostream>
using namespace std;

int main() {

        int x, y, small, large ;

        cout << " Inserire due numeri interi: " ;
        cin  >> x >> y ;

        if(x > y) {
             large = x ;
             small = y ;
        } else {
             large = y ;
             small = x ;
        }

        cout << endl  << " In ordine crescente: "
             << small << " " << large << endl << endl ;

        return 1;

}
```

if... else annidati

- Nei costrutti più complessi, in cui si vogliano porre delle condizioni all'interno di un if…else, si pone il problema di associare correttamente l'else ad un if.
- La regola è che ogni else è associato al più vicino if che lo preceda, sempre che questo non sia già stato associato ad un altro else.
- Le associazioni sono **evidenti** se si scrive il codice in maniera **ordinata**

Esempio

Quanto vale y dopo l'esecuzione di questo blocco di istruzioni, dato x = 4.5?

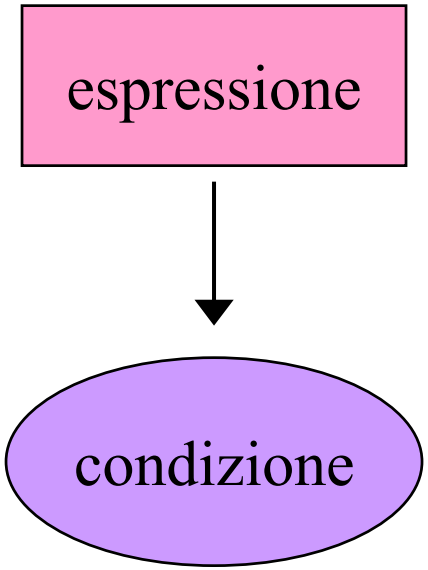
```
y = 3;
if (x > 4)
if (x > 5)
if (x > 6) y = 3;
else y = 4;
else y = 5;
```

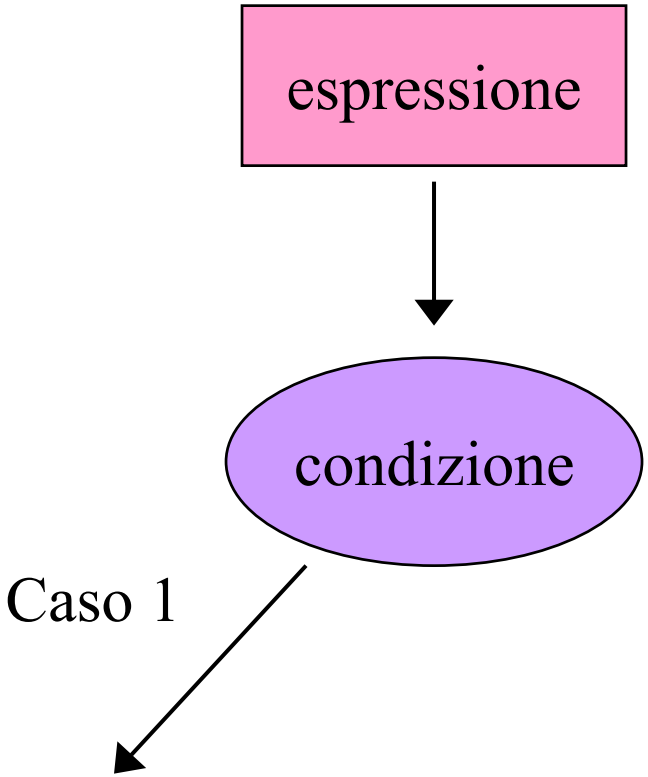
INCOMPRENSIBILE !

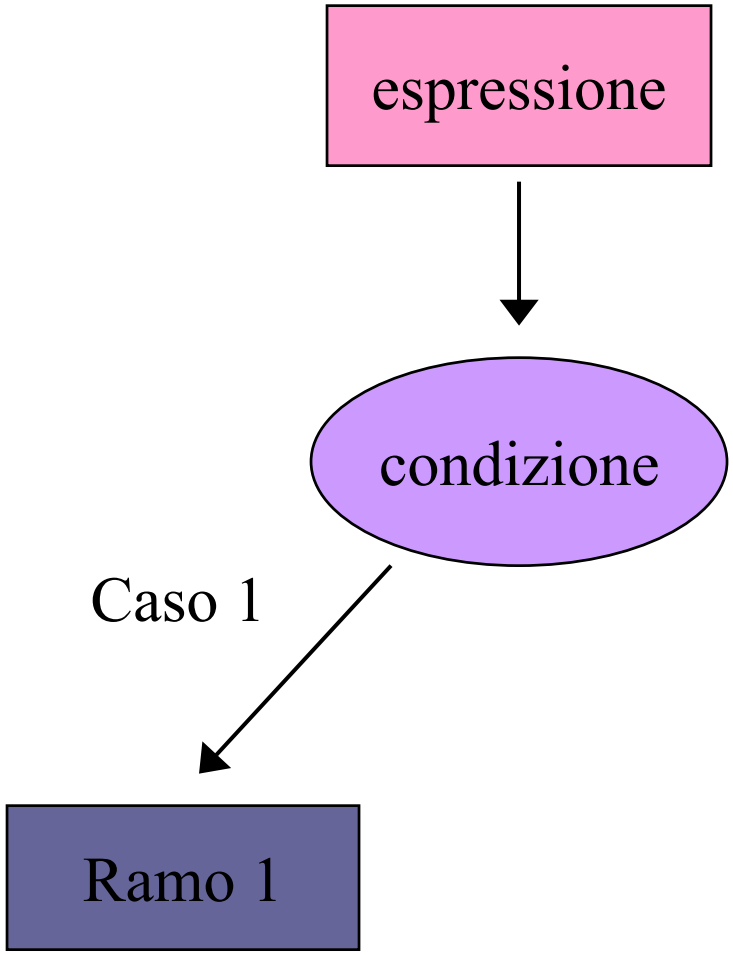
Utilizziamo l'**incolonnamento** per visualizzare i diversi blocchi logici:

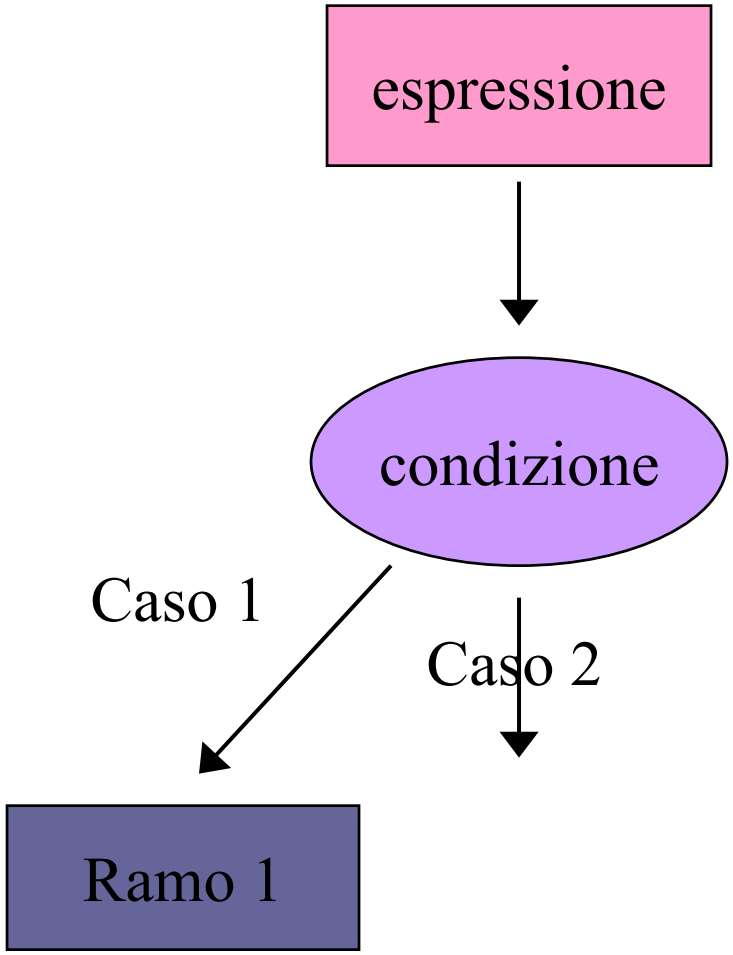
```
y=3;

if (x > 4) {                    // vero per x=4.5
    if (x > 5) {                // falso per x=4.5
        if (x > 6) {
            y = 3;
        } else {
            y = 4;
        }
    } else {                    // eseguito se x <= 5
       y = 5;                   // per x=4.5 y=5 !
    }
}
```

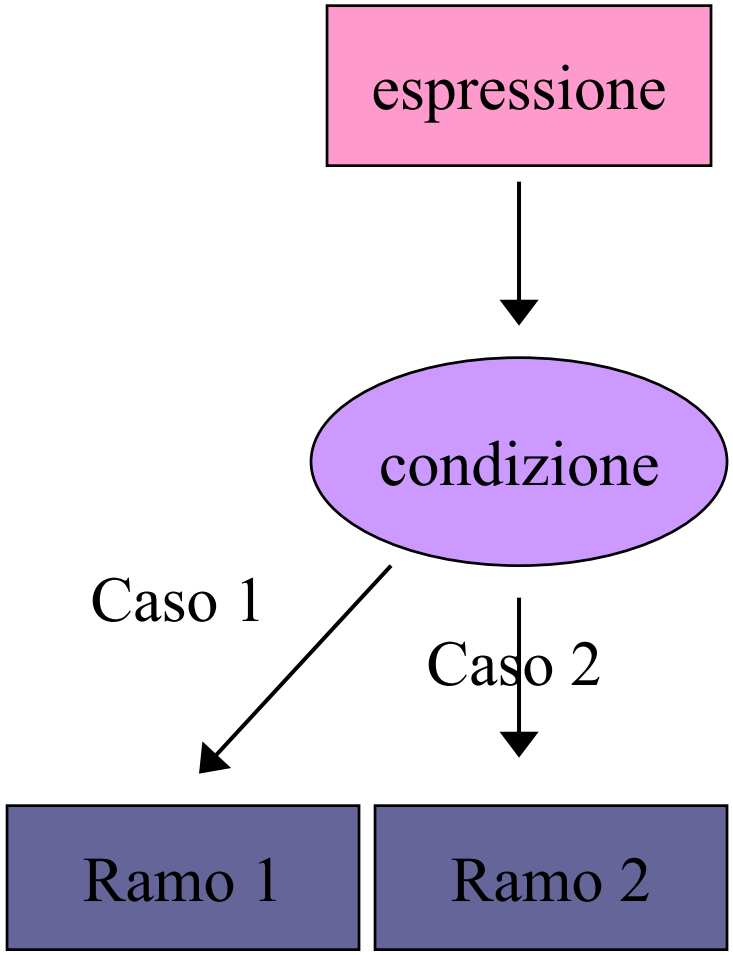
sono tutte istruzioni singole e quindi le parentesi graffe non sono necessarie, ma aiutano!

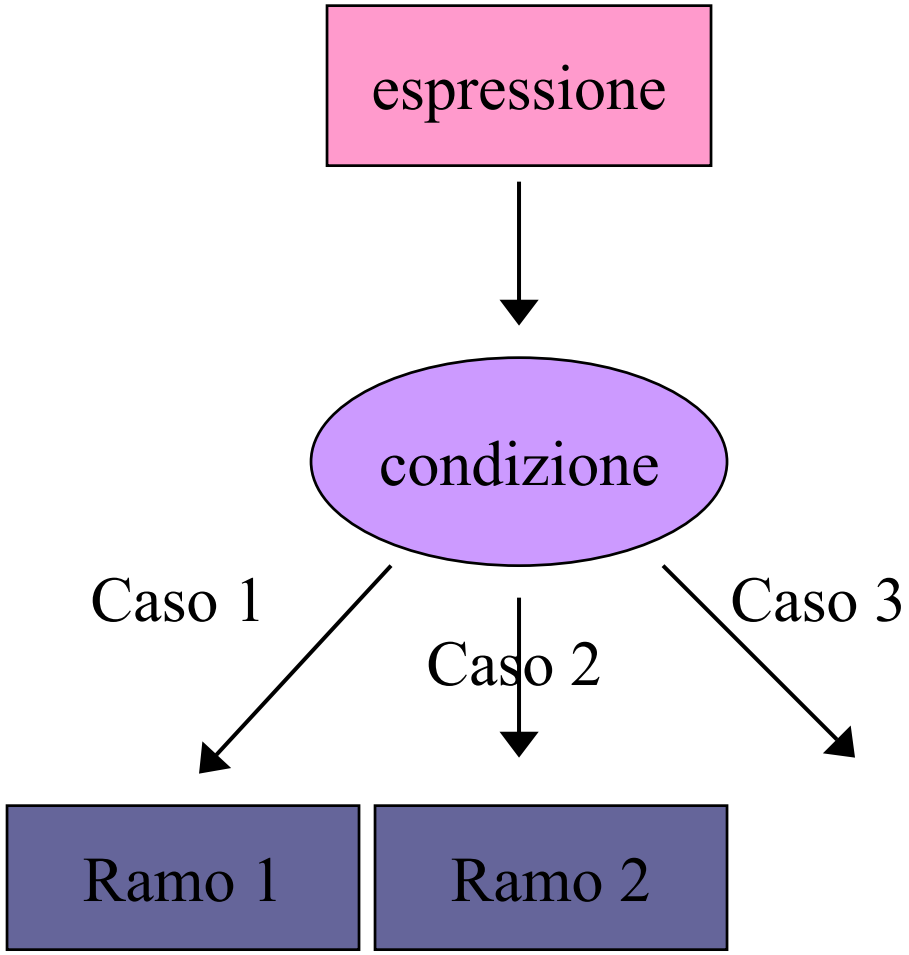

espressione
condizione

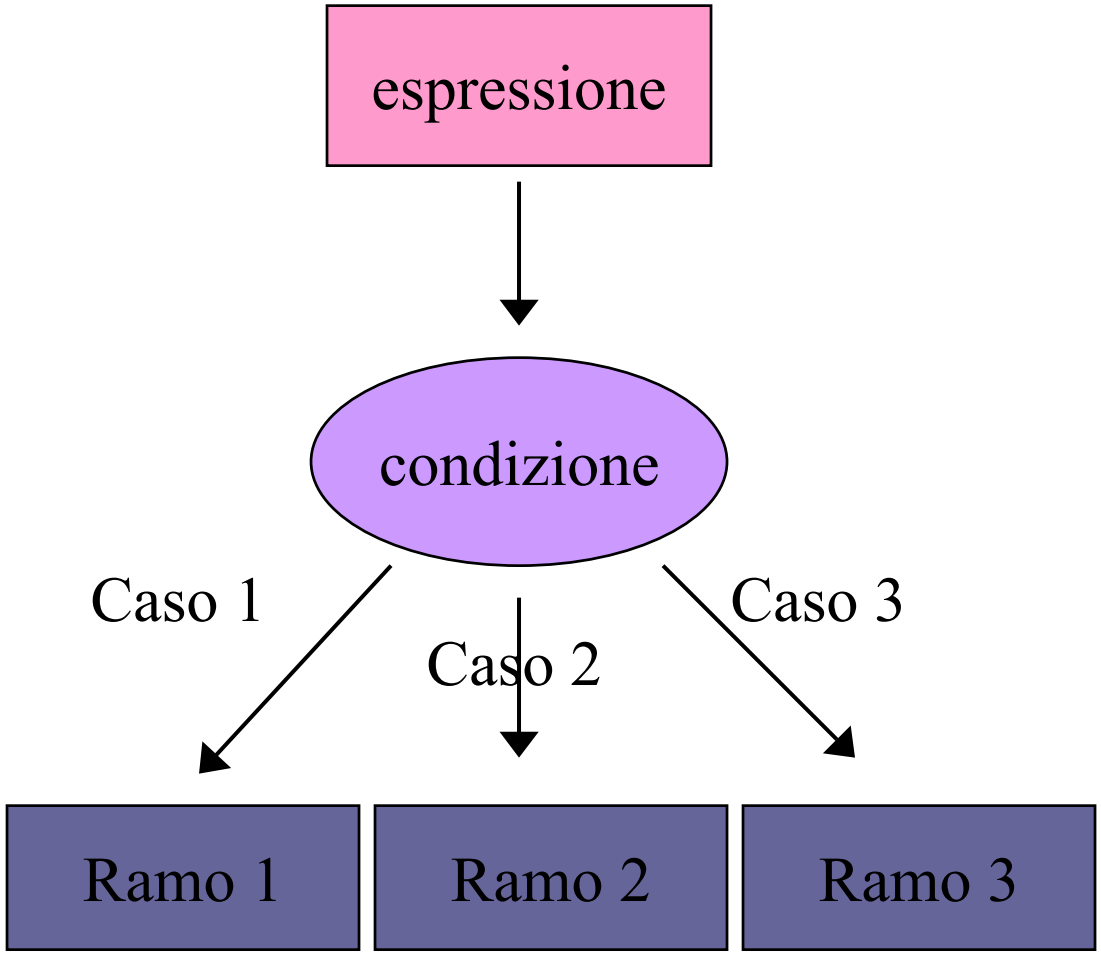

espressione
condizione
Caso 1


espressione
condizione
Caso 1
Ramo 1


espressione
condizione
Caso 1
Caso 2
Ramo 1


espressione
condizione
Caso 1
Caso 2
Ramo 1
Ramo 2


espressione
condizione
Caso 1
Caso 2
Caso 3
Ramo 1
Ramo 2


espressione
condizione
Caso 1
Caso 2
Caso 3
Ramo 1
Ramo 2
Ramo 3

Sintassi di if… else if…else

```
if(espressione1) {
        istruzione1

} else if (espressione2)  {
        istruzione2

} else if (espressione3)  {
        istruzione3


……….

} else if (espressioneN)  {
        istruzioneN

} else {
        istruzione
}
```

Nell'esecuzione dell'istruzione if… else if…else

1. Viene valutata l'espressione corrispondente a ciascun if

2. Se la condizione espressa in ciascun if e' soddisfatta, viene eseguito il blocco istruzioni ad esso associato.

3. Dopo la prima condizione soddisfatta (**se esiste**), si passa alla prima istruzione successiva al blocco if… else if … else (cioe' viene eseguito **al massimo 1** blocco istruzioni)

4. Se nessuna condizione e' soddisfatta, si esegue il blocco istruzioni corrispondente ad else
   (in questo caso **1** blocco istruzioni e' **sempre** eseguito)

5. Il blocco corrispondente a else puo' anche essere omesso (in questo caso **potrebbe** non essere eseguito **nessun** blocco istruzioni)

Esempio di uso di if… else if…else

```
#include <iostream>
using namespace std;

int main() {

        int dato ;

        cout << " Inserire il valore del dato: " ;
        cin  >> dato  ;

        if(dato == 5) {
             cout << endl  << " Dato = 5 "  << endl ;
        } else if (dato == 4) {
             cout << endl  << " Dato = 4 "  << endl ;
        }  else if (dato == 3) {
             cout << endl  << " Dato = 3 "  << endl ;
        } else {
             cout << endl  << " Dato diverso da 3, 4 o 5 " << endl ;
        }

        cout << endl  << " Ho finito. Ciao "  << endl << endl ;

        return 1;

}
```

Confronto tra if e while

while è un if insistente!

6) Elementi di linguaggio C/C++

A 1 - istruzioni e operatori booleani

2 - iterazioni (`for`, `while`, `do … while`)

B - istruzioni di selezione (`if`, `switch`, `else`)

C - funzioni predefinite. La classe `math`.

C) Funzioni predefinite.
La classe math

Uso di funzioni

- Per molti scopi possiamo utilizzare una funzione esistente, abbiamo solo bisogno di conoscere
  - Il suo *prototipo*
  - Le *precondizioni*
  - Le *postcondizioni*
- Mentre possiamo ignorare completamente
  - la sua *implementazione*
- La forma generale di una chiamata ad una funzione è un'espressione del tipo

  nome_funzione (lista_di_argomenti)

Precondizioni e Postcondizioni

- Precondizione:
  ciò che la funzione richiede

- Postcondizione:
  ciò che la funzione farà
  se è soddisfatta la precondizione

Prototipo di funzione

- Dichiarazione completa di una funzione.
  I file header contengono di solito una lista di prototipi

- File header del C header.h (es. math.h string.h)
  Corrispondenti file del C++ cheader (es. cmath) oppure
  header (es. string)

- Sintassi:

  tipo nome_funzione (lista_parametri) ;

  Dove *lista_parametri* consiste in zero
  o più parametri separati da virgole

Un parametro può essere

- tipo
- tipo identificatore

  tipo * identificatore // maggiori dettagli nella prossima lezione
- tipo & identificatore // maggiori dettagli tra 2 lezioni
- const altro_parametro

Informazioni fornite dal prototipo

- il tipo (classe) di oggetto ritornato dalla funzione
- il nome della funzione
- il numero di argomenti da usare nella chiamata
- il tipo (classe) degli argomenti

Moduli

- Un modulo (libreria) è una raccolta di cose collegate tra di loro, quali funzioni, costanti e classi.
- Ad esempio il modulo math rappresenta una raccolta di funzioni matematiche e di costanti utili, come M_PI che vale π.
- Per usare le funzioni definite in un modulo bisogna
  - Includere il file header (ad esempio cmath) del modulo per avere i prototipi delle funzioni
  - Caricare la libreria del modulo durante il link (**implicitamente** se si tratta di librerie note al compilatore, **esplicitamente** in caso di librerie private)

cplusplus.com
Information
Documentation
Reference
Articles
Forum

Reference
C Library
IOstream Library
Strings library
STL Containers
STL Algorithms
Miscellaneous

C Library
cassert (assert.h)
cctype (ctype.h)
cerrno (errno.h)
cfloat (float.h)
ciso646 (iso646.h)
climits (limits.h)
clocale (locale.h)
cmath (math.h)
csetjmp (setjmp.h)
csignal (signal.h)
cstdarg (stdarg.h)
cstddef (stddef.h)
cstdio (stdio.h)
cstdlib (stdlib.h)
cstring (string.h)
ctime (time.h)

cmath (math.h)
functions:
abs
acos
asin
atan
atan2
ceil
cos
cosh
exp
fabs
floor
fmod
frexp
ldexp
log
log10
modf
pow
sin
sinh
sqrt
tan
tanh
macro constants:
HUGE_VAL

cmath (math.h)

header

C numerics library

`cmath` declares a set of functions to compute common mathematical operations and transformations:

Trigonometric functions:

cos	Compute cosine (function)
sin	Compute sine (function)
tan	Compute tangent (function)
acos	Compute arc cosine (function)
asin	Compute arc sine (function)
atan	Compute arc tangent (function)
atan2	Compute arc tangent with two parameters (function)

Hyperbolic functions:

cosh	Compute hyperbolic cosine (function)
sinh	Compute hyperbolic sine (function)
tanh	Compute hyperbolic tangent (function)

Exponential and logarithmic functions:

exp	Compute exponential function (function)
frexp	Get significand and exponent (function)
ldexp	Generate number from significand and exponent (function)
log	Compute natural logarithm (function)
log10	Compute common logarithm (function)
modf	Break into fractional and integral parts (function)

Power functions

pow	Raise to power (function)
sqrt	Compute square root (function)

Rounding, absolute value and remainder functions:

ceil	Round up value (function)
fabs	Compute absolute value (function)
floor	Round down value (function)
fmod	Compute remainder of division (function)

Esempio di uso di math

```
#include <iostream>
#include <cmath>

using namespace std;

int main() {

        double a, b, x ;

        a = 2  ;
        b = 3  ;
        x = 30 ;

        double xr = x * M_PI / 180. ;

        cout << endl << " " << a << " elevato a " << b << " = "
             << pow(a, b) << endl ;

        double y = sin(xr)  ;
        cout << " sin(" << x << ") = " << y << " (x in gradi) "
             <<  endl << endl ;

        cout <<  "   pi greco =  "  << M_PI << endl ;
        cout <<  " e di Nepero = "  << M_E  << endl  << endl ;

        return 1;

}
```

http://www.cplusplus.com/reference/

Reference

Reference of the C++ Language Library, with detailed descriptions of its elements and examples on how to use its functions

The standard C++ library is a collection of functions, constants, classes, objects and templates that extends the C++ language providing basic functionality to perform several tasks, like classes to interact with the operating system, data containers, manipulators to operate with them and algorithms commonly needed.

The declarations of the different elements provided by the library are split in several headers that shall be included in the code in order to have access to its components:

algorithm	complex	exception	list	stack
bitset	csetjmp	fstream	locale	stdexcept
cassert	csignal	functional	map	strstream
cctype	cstdarg	iomanip	memory	streambuf
cerrno	cstddef	ios	new	string
cfloat	cstdio	iosfwd	numeric	typeinfo
ciso646	cstdlib	iostream	ostream	utility
climits	cstring	istream	queue	valarray
clocale	ctime	iterator	set	vector
cmath	deque	limits	sstream	

It can be divided into:

C Library

The elements of the C language library are also included as a subset of the C++ Standard library. These cover many aspects, from general utility functions and macros to input/output functions and dynamic memory management functions:

cassert (assert.h)	C Diagnostics Library (header)
cctype (ctype.h)	Character handling functions (header)
cerrno (errno.h)	C Errors (header)
cfloat (float.h)	Characteristics of floating-point types (header)
ciso646 (iso646.h)	ISO 646 Alternative operator spellings (header)
climits (limits.h)	Sizes of integral types (header)
clocale (locale.h)	C localization library (header)
cmath (math.h)	C numerics library (header)
csetjmp (setjmp.h)	Non local jumps (header)
csignal (signal.h)	C library to handle signals (header)
cstdarg (stdarg.h)	Variable arguments handling (header)
cstddef (stddef.h)	C Standard definitions (header)
cstdio (stdio.h)	C library to perform Input/Output operations (header)
cstdlib (stdlib.h)	C Standard General Utilities Library (header)
cstring (string.h)	C Strings (header)
ctime	C Time Library (header)

Language support library:

limits	Numeric limits (header)
new	Dynamic memory (header)
typeinfo	Type information (header)
exception	Standard exception class (class)

Diagnostics library:

stdexcept	Exception classes (header)

General utilities library:

utility	Utility components (header)
functional	Function objects (header)
memory	Memory elements (header)

Strings library:

string	C++ Strings library (library)

Localization library:

locale	Localization library (header)

C++ Standard Library: Standard Template Library (STL)

m/tech/stl/

Containers library:

bitset	Bitset (class template)
deque	Double ended queue (class template)
list	List (class template)
map	Map (class template)
multimap	Multiple-key map (class template)
multiset	Multiple-key set (class template)
priority_queue	Priority queue (class template)
queue	FIFO queue (class template)
set	Set (class template)
stack	LIFO stack (class template)
vector	Vector (class template)

Iterators library:

iterator	Iterator definitions (header)

Algorithms library:

STL Algorithms	Standard Template Library: Algorithms (library)

Numeric library:

complex	Complex numbers library (header)
valarray	Library for arrays of numeric values (header)
numeric	Generalized numeric operations (header)

C++ Standard Library: Input/Output Stream Library

Provides functionality to use an abstraction called *streams* specially designed to perform input and output operations on sequences of character, like files or strings. This functionality is provided through several related classes, as shown in the following relationship map, with the corresponding header file names on top: