



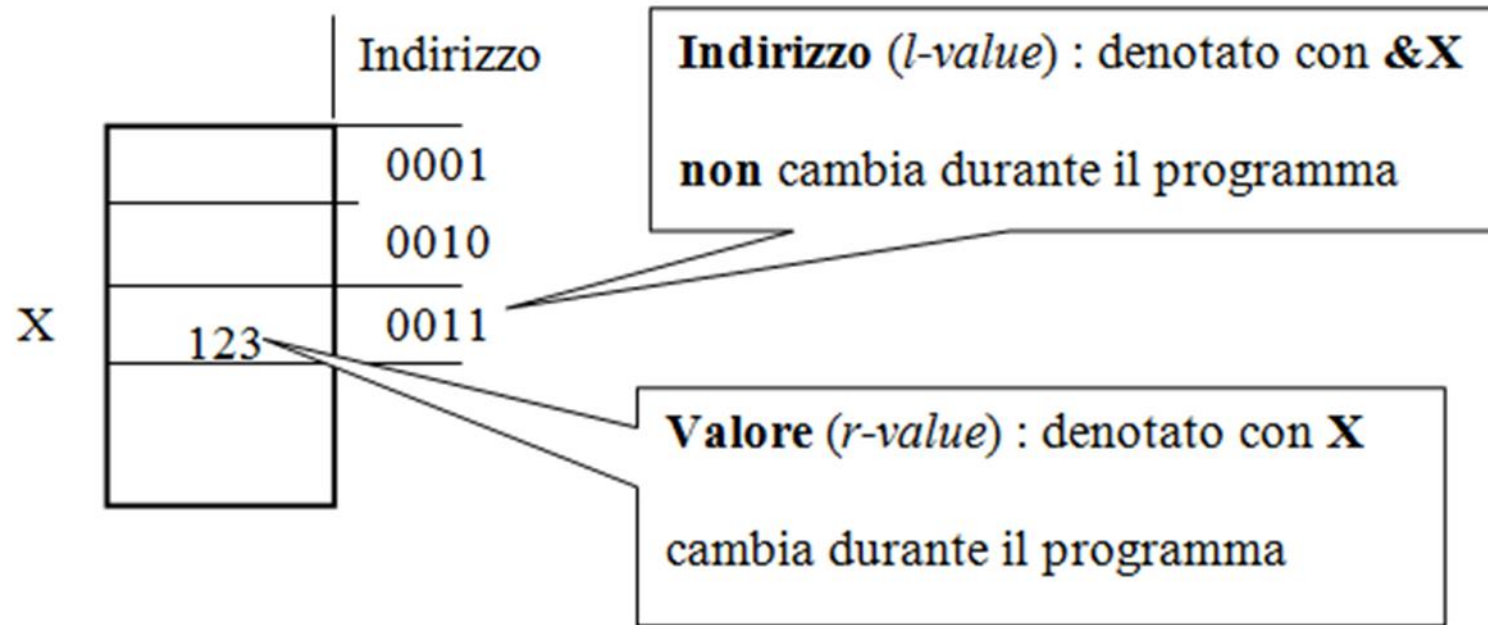
**UNIVERSITÀ
DI PARMA**

C++ variabili e tipi di dato

- si definisce variabile uno spazio identificato da un nome in cui è possibile scrivere, recuperare e manipolare dati nel corso del programma
- una variabile è caratterizzata da:
 - il suo valore ***right value*** (*rvalue*)
 - il suo indirizzo, ***left value*** (*lvalue*)
 - lo spazio occupato
- la ***dichiarazione*** di una variabile associa un identificatore a un tipo e determina l'allocazione di un'area di memoria (***non assegna un valore alla variabile!***)
- è possibile inizializzare una variabile durante la dichiarazione
- per convenzione l'***identificatore*** (nome variabile) è scritto in minuscolo e dovrebbe essere mnemonico (***non può essere una parola riservata!***)

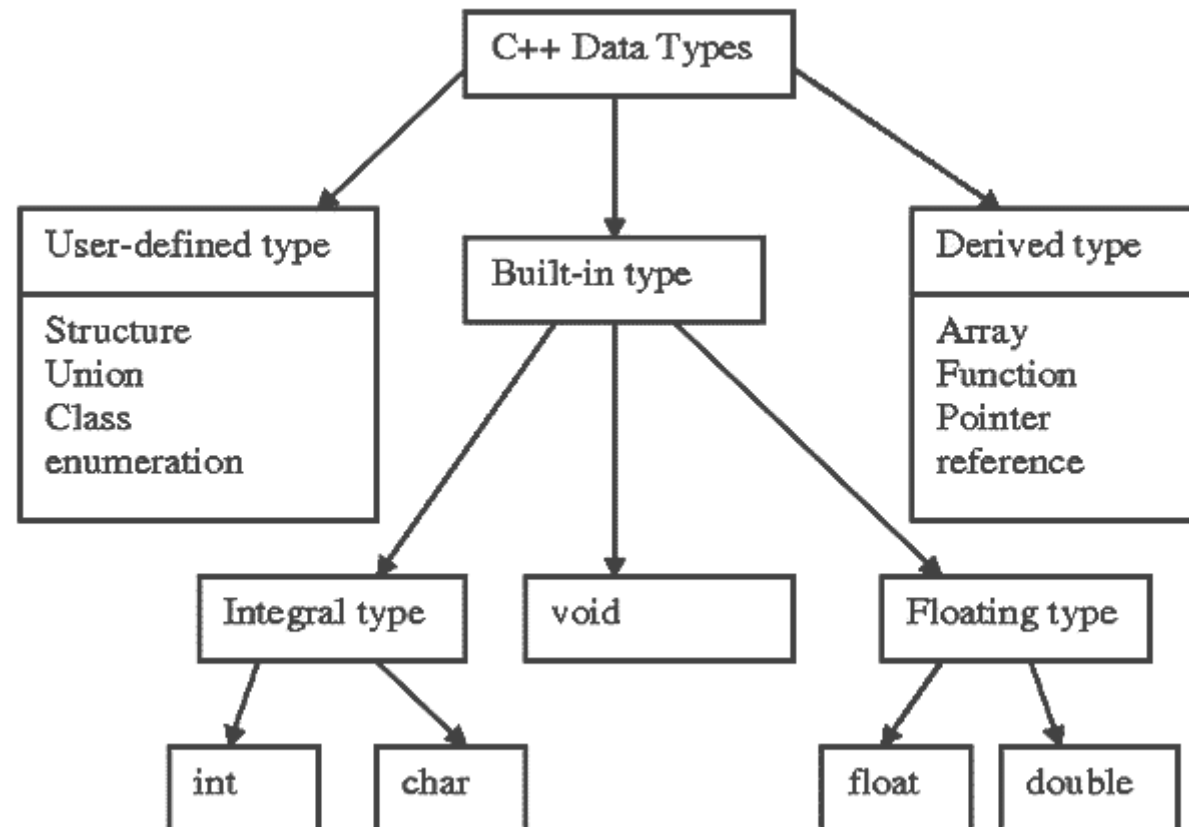
- una variabile può essere utilizzata (cioè è visibile) solo dopo la sua definizione
- le variabili definite in un ambiente (blocco o funzione) sono visibili in tutti gli ambienti in esso contenuti
- all'interno di uno stesso ambiente non è possibile definire due variabili con lo stesso nome
- in ambiente distinti è possibile definire variabili con lo stesso nome, anche di tipo diverso
- se in un ambiente sono visibili più variabili con lo stesso nome, il nome si riferisce a quella la cui dichiarazione è «più vicina» al punto di utilizzo

- variabili globali (*permanenti*)
 - create all'inizio del programma sono *visibili a tutte le funzioni* e il loro ciclo di vita termina con la terminazione del *programma*
- variabili locali (*temporanee*)
 - vengono *istanziate* al momento in cui vengono eseguite le prime istruzioni del *blocco* in cui sono contenute
 - le variabili vengono *distrutte* e la memoria ad esse associata viene rilasciata al *termine dell'esecuzione del blocco* in cui sono dichiarate



- è necessario che associare un tipo
 - ad ogni variabile
 - ad ogni parametro di funzione
 - ad ogni valore restituito da una funzione
- i tipi di dato si dividono in 3 categorie
 - tipi di dato **scalari** (int, char, float, double, boolean ...)
 - tipi di dato **strutturati** definiti nelle librerie (string ...) o dall'utente (utilizzando struct o class)
 - **puntatori**
- il tipo specifica
 - la quantità di memoria che verrà allocata per la variabile (o risultato dell'espressione)
 - l'insieme dei valori che è possibile memorizzare in una variabile
 - la modalità di interpretazione di questi valori (schemi di bit - rappresentazione)
 - le operazioni che è possibile eseguire sui valori

- *direttiva* al preprocessore
 - `#define PI_GRECO 3.141592`
 - il preprocessore sostituisce ogni occorrenza di `PI_GRECO` con `3.141592`
- *const*
 - `const int PI_GRECO 3.141592;`
 - il valore non può essere modificato nel corso del programma
 - `PI_GRECO = 2; // error: assignment of read-only variable 'PI_GRECO'`



```
#include <iostream>
int main() {
    int i; double d;
    float f; char c;
    bool b; int v[10];
    int *pi; double *pd;
    std::cout << "size of int:      " << sizeof i << " bytes " << std::endl;
    std::cout << "size of double:    " << sizeof d << " bytes " << std::endl;
    std::cout << "size of float:      " << sizeof f << " bytes " << std::endl;
    std::cout << "size of char:       " << sizeof c << " bytes " << std::endl;
    std::cout << "size of bool:       " << sizeof b << " bytes " << std::endl;
    std::cout << "size of int[10]:    " << sizeof v << " bytes " << std::endl;
    std::cout << "size of int*:       " << sizeof pi << " bytes " << std::endl;
    std::cout << "size of double*:    " << sizeof pd << " bytes " << std::endl;
    return 0;
}
```

```
size of int:      4 bytes
size of double:   8 bytes
size of float:    4 bytes
size of char:     1 bytes
size of bool:     1 bytes
size of int[10]:  40 bytes
size of int*:     4 bytes
size of double*:  4 bytes
```

	http://en.cppreference.com/w/cpp/types/numeric_limits		
	<code>min()</code>	<code>lowest()</code> (C++11)	<code>max()</code>
<code>numeric_limits< char ></code>	<u>CHAR_MIN</u>	<u>CHAR_MIN</u>	<u>CHAR_MAX</u>
<code>numeric_limits< short ></code>	<u>SHRT_MIN</u>	<u>SHRT_MIN</u>	<u>SHRT_MAX</u>
<code>numeric_limits< signed short ></code>			
<code>numeric_limits< int ></code>	<u>INT_MIN</u>	<u>INT_MIN</u>	<u>INT_MAX</u>
<code>numeric_limits< signed int ></code>			
<code>numeric_limits< long ></code>	<u>LONG_MIN</u>	<u>LONG_MIN</u>	<u>LONG_MAX</u>
<code>numeric_limits< signed long ></code>			
<code>numeric_limits< long long ></code>	<u>LLONG_MIN</u>	<u>LLONG_MIN</u>	<u>LLONG_MAX</u>
<code>numeric_limits< signed long long ></code>			
<code>numeric_limits< float ></code>	<u>FLT_MIN</u>	<u>-FLT_MAX</u>	<u>FLT_MAX</u>
<code>numeric_limits< double ></code>	<u>DBL_MIN</u>	<u>-DBL_MAX</u>	<u>DBL_MAX</u>
<code>numeric_limits< long double ></code>	<u>LDBL_MIN</u>	<u>-LDBL_MAX</u>	<u>LDBL_MAX</u>

```
#include <limits>
#include <iostream>
using namespace std;
int main()
{
    cout << "type \t lowest \t highest \n";
    cout << "int \t" << numeric_limits<int>::lowest() << '\t'
        << numeric_limits<int>::max() << '\n';
    cout << "int \t" << INT_MIN << '\t' << INT_MAX << '\n';
    cout << "char\t" << static_cast<int>(numeric_limits<char>::lowest()) << '\t'
        << static_cast<int>(numeric_limits<char>::max()) << '\n';
    cout << "char\t" << CHAR_MIN << '\t' << CHAR_MAX << '\n';
    cout << "float\t" << numeric_limits<float>::lowest() << '\t'
        << numeric_limits<float>::max() << '\n';
    cout << "double\t" << numeric_limits<double>::lowest() << '\t'
        << numeric_limits<double>::max() << '\n';
}
```

type	lowest	highest
int	-2147483648	2147483647
int	-2147483648	2147483647
char	-128	127
char	-128	127
float	-3.40282e+038	3.40282e+038
double	-1.79769e+308	1.79769e+308

- operazioni su numeri:
 - +, -, *, /, %, *attenzione sono assegnamenti*
 - attenzione: la divisione tra interi dà risultato intero (trunc)
 - assegnamento: =, +=, -= ...
- confronti: >, >=, <, <=, !=, ==
 - attenzione: i confronti non si possono concatenare
- operazioni booleane (and, or, not): &&, ||, !
 - `cout << (3 < 5) << endl;` // output -> 1
 - `cout << (3 < 5 < 4) << endl;` // output -> 1 (!!!)
 - `cout << (3 < 5 && 5 < 4) << endl;` // output -> 0

- nel caso di operazioni con operandi misti il risultato dell'espressione viene automaticamente convertito nel tipo con precisione maggiore
 - esempio: $(15.2 / 2)$ il risultato sarà di tipo *float*
 - esempio:

```
double d;  
int i = 5;  
d = i;                // il valore di i viene convertito in double
```
 - in questo caso si tratta di *casting implicito*
- in C++ è possibile effettuare una conversione esplicita (*casting esplicito*) mediante l'operatore cast

```
static_cast<tipo>(espressione)
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {
```

```
    cout << "static_cast<int>(4.9) = " << static_cast<int>(4.9) << endl;
```

```
    cout << "static_cast<int>(4.2) = " << static_cast<int>(4.2) << endl;
```

```
    cout << "static_cast<double>(10) / 3 = " << static_cast<double>(10) / 3 << endl;
```

```
    cout << "static_cast<double>(10 / 3) = " << static_cast<double>(10 / 3) << endl;
```

```
    cout << "(double)15 / 2 = " << (double)15 / 2 << endl;
```

```
    return 0;
```

```
}
```

```
static_cast<int>(4.9) = 4
static_cast<int>(4.2) = 4
static_cast<double>(10) / 3 = 3.33333
static_cast<double>(10 / 3) = 3
(double)15 / 2 = 7.5
```

```
#include <cstdlib>
#include <iostream>
#include <ctime>

int main() {
    std::srand(std::time(nullptr)); // use current time as seed for random generator
    int random_variable = std::rand();
    std::cout << "Random value on [0 " << RAND_MAX << "]: " << random_variable << '\n';
    int range;
    std::cout << "Insert range ";
    std::cin >> range;
    std::cout << "Random value on [0 " << range << "]: " << random_variable % range <<
    '\n';
}
```


Dati strutturati

DEFINIZIONE DI NUOVI TIPI DI DATO

- è possibile specificare un *alias* per tipi di dato già definiti

- *typedef*

```
typedef bool logico;  
logico trovato;  
trovato = true;
```

- *using*

```
using intero = int;  
intero n;
```

- la dichiarazione ***typedef*** permette di creare un alias per la definizione di un tipo di dato

- esempio tipo semplice

```
typedef unsigned long ulong;    // ulong nuovo tipo
ulong l;                        // l variabile di tipo unsigned long
```

- esempio struttura

```
typedef struct {
    double x;
    double y;
} punto;                        // punto nuovo tipo di dato
punto p1;                       // p1 variabile di tipo punto
p1.y = 4.5;
```

- per dati con significati speciali, è possibile definire insiemi di valori come sequenze di identificatori tramite il costruttore di tipo enum.

```
enum {Lu, Ma, Me, Gi, Ve, Sa, Do} giorno;  
giorno = Me;  
// oppure  
typedef Mese {Gen, Feb, Mar, Apr, Mag, Giu, Lug, Ago, Set, Ott, Nov, Dic};  
Mese meseCorrente;  
meseCorrente = Mar;
```

- il primo identificatore ha valore 0, il successivo ha valore 1, e così via. È comunque possibile assegnare agli identificatori valori espliciti:

```
enum Color { red, green, blue };  
Color r = red;  
switch(r) {  
    case red : std::cout << "red\n"; break;  
    case green: std::cout << "green\n"; break;  
    case blue : std::cout << "blue\n"; break;  
}
```