Tipi di Dato: Array

Informatica A AA 2023 / 2024

Giacomo Boracchi

2 Ottobre 2023

giacomo.boracchi@polimi.it

Installate Code::Blocks

Queste istruzioni dovrebbero andare bene

Win 10 e Mac OS:
https://boracchi.faculty.polimi.it/teaching/InfoA/2021_InfoA_installazione_IDE.pdf

(su mac potrebbero esserci problem)

Win 11:

https://boracchi.faculty.polimi.it/teaching/InfoA/Install_code_blocks%20_win11.pdf

Assicuratevi di scaricare la versione aggiornata di Code::Blocks, che trovate sempre sul sito: http://www.codeblocks.org/

Installate Code::Blocks

Not secure | codeblocks.org/downloads/26

Main

- Home
- Features
- Screenshots
- Downloads
  - Binaries
  - Source
  - SVN
- Plugins
- User manual
- Licensing
- Donations

Quick links

- FAQ
- Wiki
- Forums
- Forums (mobile)
- Nightlies
- Ticket System
- Browse SVN
- Browse SVN log

Please select a setup package depending on your platform:

- **Windows XP / Vista / 7 / 8.x / 10**
- **Linux 32 and 64-bit**
- **Mac OS X**

NOTE: For older OS'es use older releases. There are releases for many OS version and platforms on the **Sourceforge.net** page.

NOTE: There are also more recent *nightly builds* available in the **forums** or (for Ubuntu users) in the **Ubuntu PPA repository**. Please note that we consider nightly builds to be *stable*, usually.

NOTE: We have a **Changelog for 20.03**, that gives you an overview over the enhancements and fixes we have put in the new release.

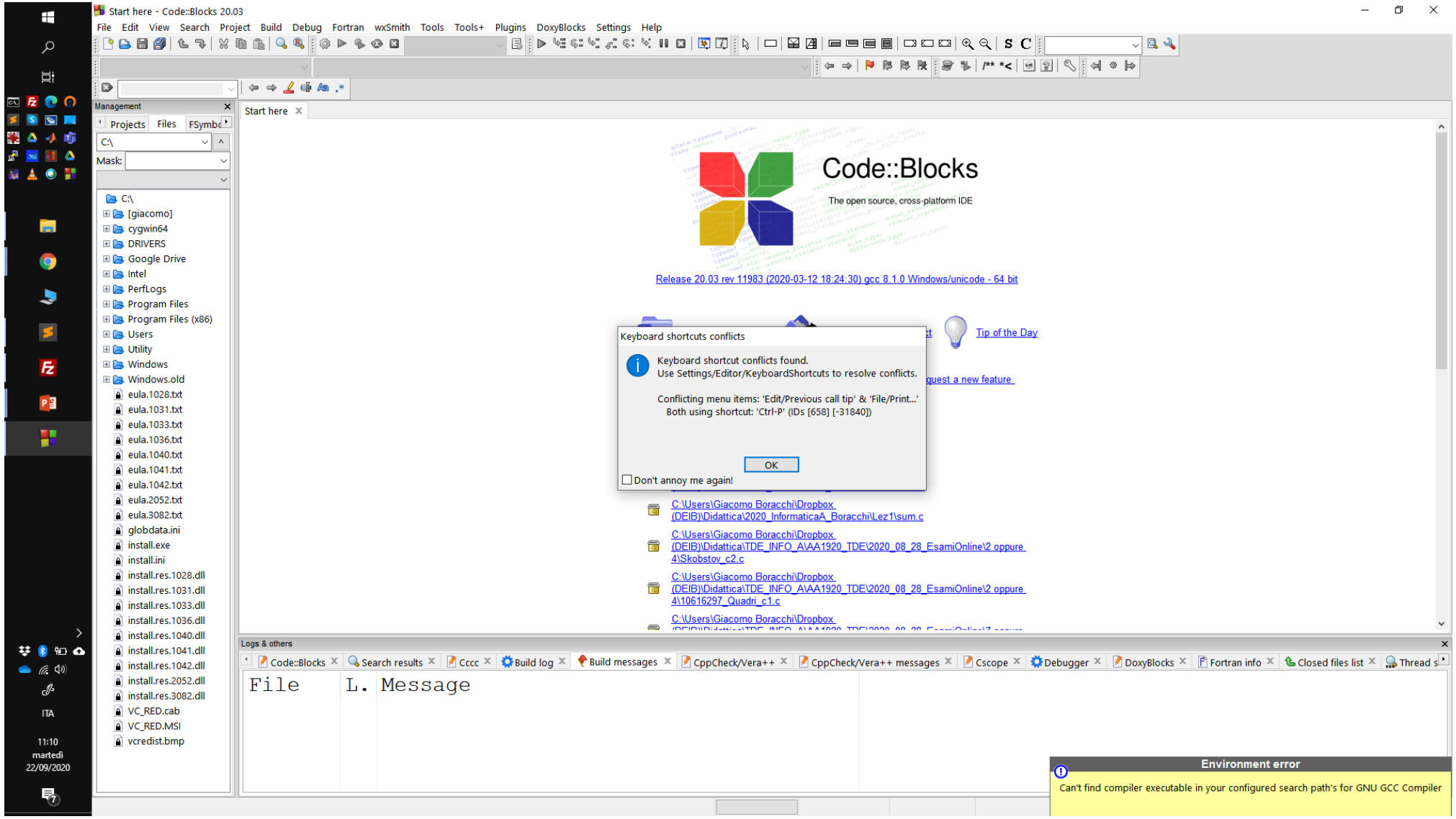
NOTE: The default builds are 64 bit (starting with release 20.03). We also provide 32bit builds for convenience.

Windows XP / Vista / 7 / 8.x / 10:

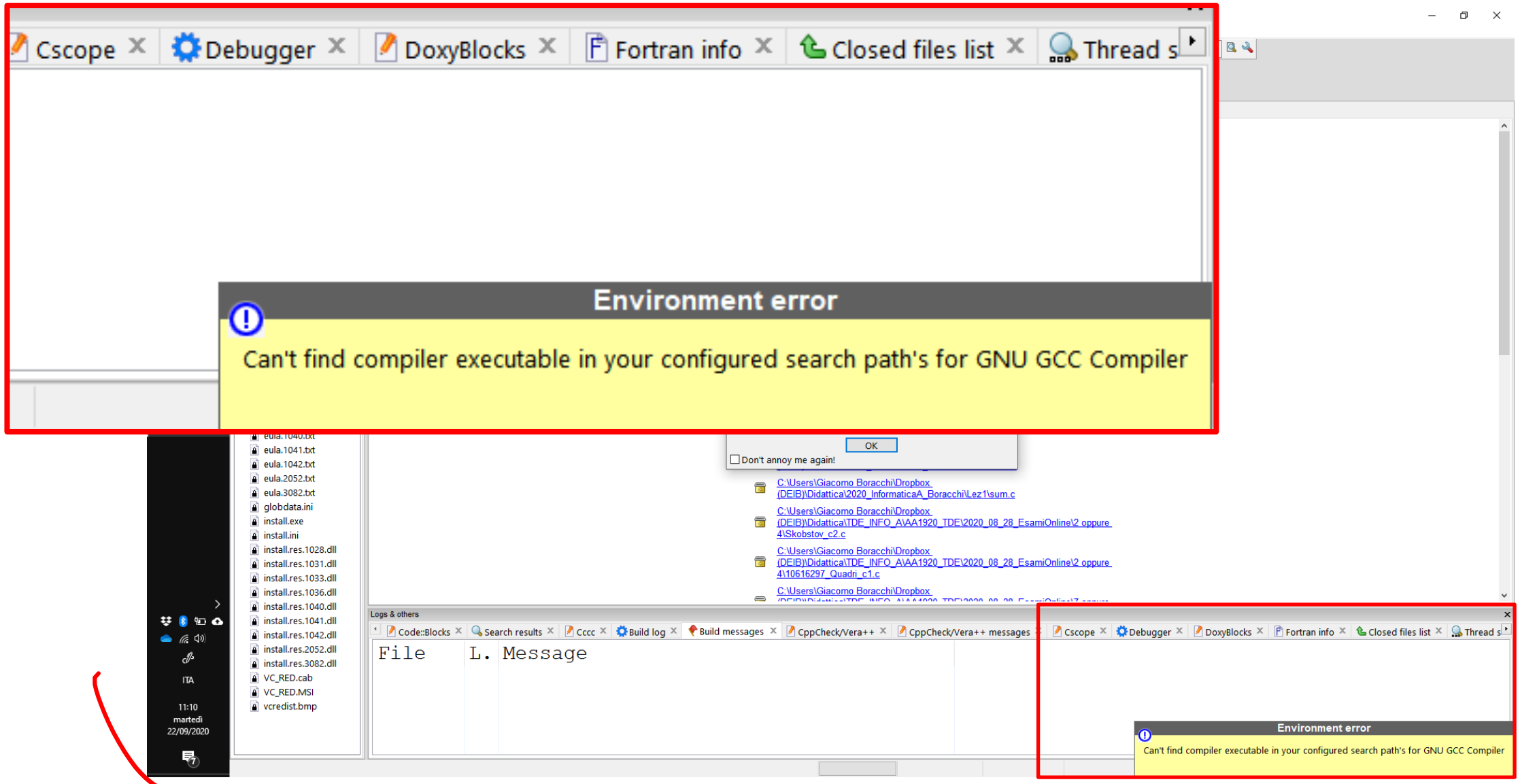
File	Date	Download from
codeblocks-20.03-setup.exe	29 Mar 2020	**FossHUB** or **Sourceforge.net**
codeblocks-20.03-setup-nonadmin.exe	29 Mar 2020	**FossHUB** or **Sourceforge.net**
codeblocks-20.03-nosetup.zip	29 Mar 2020	**FossHUB** or **Sourceforge.net**
codeblocks-20.03mingw-setup.exe	29 Mar 2020	**FossHUB** or **Sourceforge.net**
codeblocks-20.03mingw-nosetup.zip	29 Mar 2020	**FossHUB** or **Sourceforge.net**
codeblocks-20.03-32bit-setup.exe	02 Apr 2020	**FossHUB** or **Sourceforge.net**
codeblocks-20.03-32bit-setup-nonadmin.exe	02 Apr 2020	**FossHUB** or **Sourceforge.net**
codeblocks-20.03-32bit-nosetup.zip	02 Apr 2020	**FossHUB** or **Sourceforge.net**
codeblocks-20.03mingw-32bit-setup.exe	02 Apr 2020	**FossHUB** or **Sourceforge.net**
codeblocks-20.03mingw-32bit-nosetup.zip	02 Apr 2020	**FossHUB** or **Sourceforge.net**

NOTE: The codeblocks-20.03-setup.exe file includes Code::Blocks with all plugins. The codeblocks-20.03-setup-nonadmin.exe file is provided for convenience to users that do not have administrator rights on their machine(s).

Troubleshooting: compiler not found

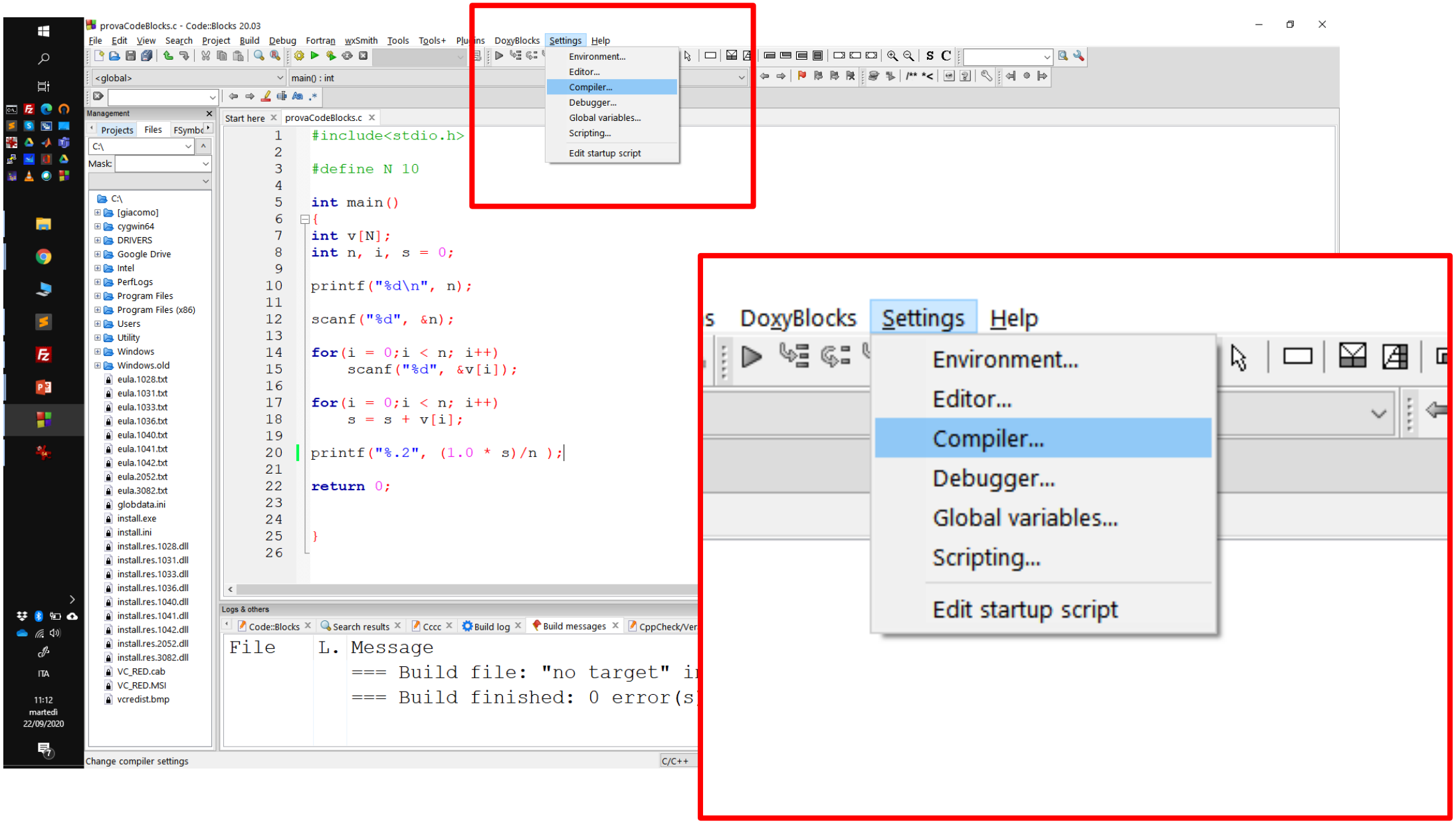


Troubleshooting: compiler not found

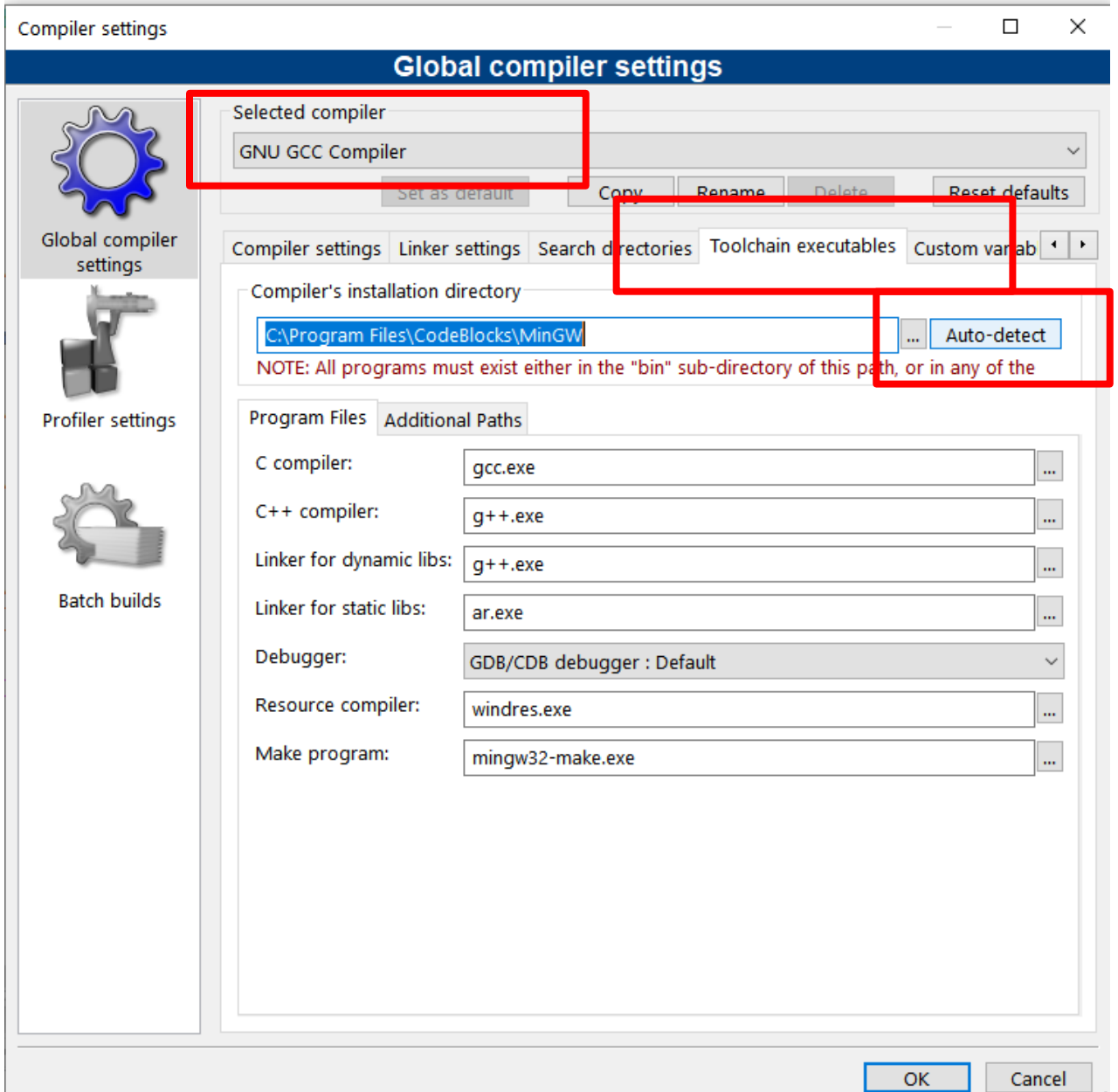


https://media.lanecc.edu/users/birdb/CS133G/CodeBlocks%20Troubleshooting.html

Troubleshooting: compiler not found



Troubleshooting: compiler not found



Selezionare Auto-detect nell'opzione toolchain executables del compilatore
Nella maggior parte dei casi sarà GNU GCC Compiler
quello che selezionerete

Warm up

Scrivere un programma per conteggiare quanto la vostra aula ha speso in totale per il pranzo Mercoledì scorso.

Calcolare la spesa media per il pranzo

Soluzione

```c
#include<stdio.h>
int main()
{
    int n, i = 0;
    float tot = 0, x = 0, max = -1;
    do{
        printf("\nquanti eravate? ");
        scanf("%d", &n);
    }while(n <= 0);

    while(i < n)
    {   printf("\nquanto hai speso? ");
        scanf("%f", &x);
        tot += x;
        i++;
        if (max < x)
            max = x; }

    printf("\n spesa media %.3f", (1.0 * tot) / n);
    printf("\n spesa massima %.3f", max);
    return 0; }
```

Altri quesiti

Altre domande:

- chi ha speso di più di tutti
- se qualcuno ha speso più di tutti gli altri messi assieme
- Si supponga di «fare alla romana» e che quindi tutti devono pagare il prezzo medio. Dire a chi ha pagato di quanto deve ricevere e a chi ha pagato quanto deve versare.

Per rispondere all'ultima domanda servirebbe **tener traccia** di quanto viene «versato» da ciascuno (i.e. i valori assegnati alla variabile `x`).

Riprendendo il paragone variabili-foglietti su cui scrivere, servirebbe, al posto i un foglietto `x`, una **sequenze di foglietti**, ed ciascun foglietto tiene traccia dei valori inseriti

Quindi sequenze di variabili: gli array

Tipi di Dato Strutturati: Gli array

Sequenze di variabili

I Tipi Strutturati in C

Permettono di immagazzinare informazione aggregata

- Vettori e matrici in matematica
- Testi (sequenza di caratteri)
- Immagini
- Rubriche
- Archivi,.. etc.

Le variabili strutturate memorizzano diversi elementi informativi:

- omogenei
- eterogenei

Oggi vedremo gli **array**

Il Costruttore Array

Gli array sono **sequenze** di **variabili omogenee**

- **sequenza:** hanno un ordinamento (sono indicizzabili)
- **omogenee:** tutte le variabili della sequenza sono dello stesso tipo

- Ogni **elemento** della sequenza è **identificato da** un **indice**
- L'array si colloca in **celle consecutive** della memoria

Il Costruttore Array

Sintassi dichiarazione di una variabile mediante costruttore array

```
tipo nomeArray[Dimensione];
```

- `tipo` la keyword di un tipo (built in o user-defined)
- **`nomeArray`** è il nome della variabile
- **`Dimensione`** è **un numero** che stabilisce il numero di elementi della sequenza.

NB: **`Dimensione`** è un **numero fisso**, noto a compile-time:

- non può essere una variabile (il suo valore sarebbe definito solo a run-time)
- non è possibile modificare le dimensioni durante l'esecuzione (e.g. allungare o accorciare l'array)

Il Costruttore Array, esempi

Esempi

- `int vet[8];`
- `char stringa[5];`
- `float resti[8];`

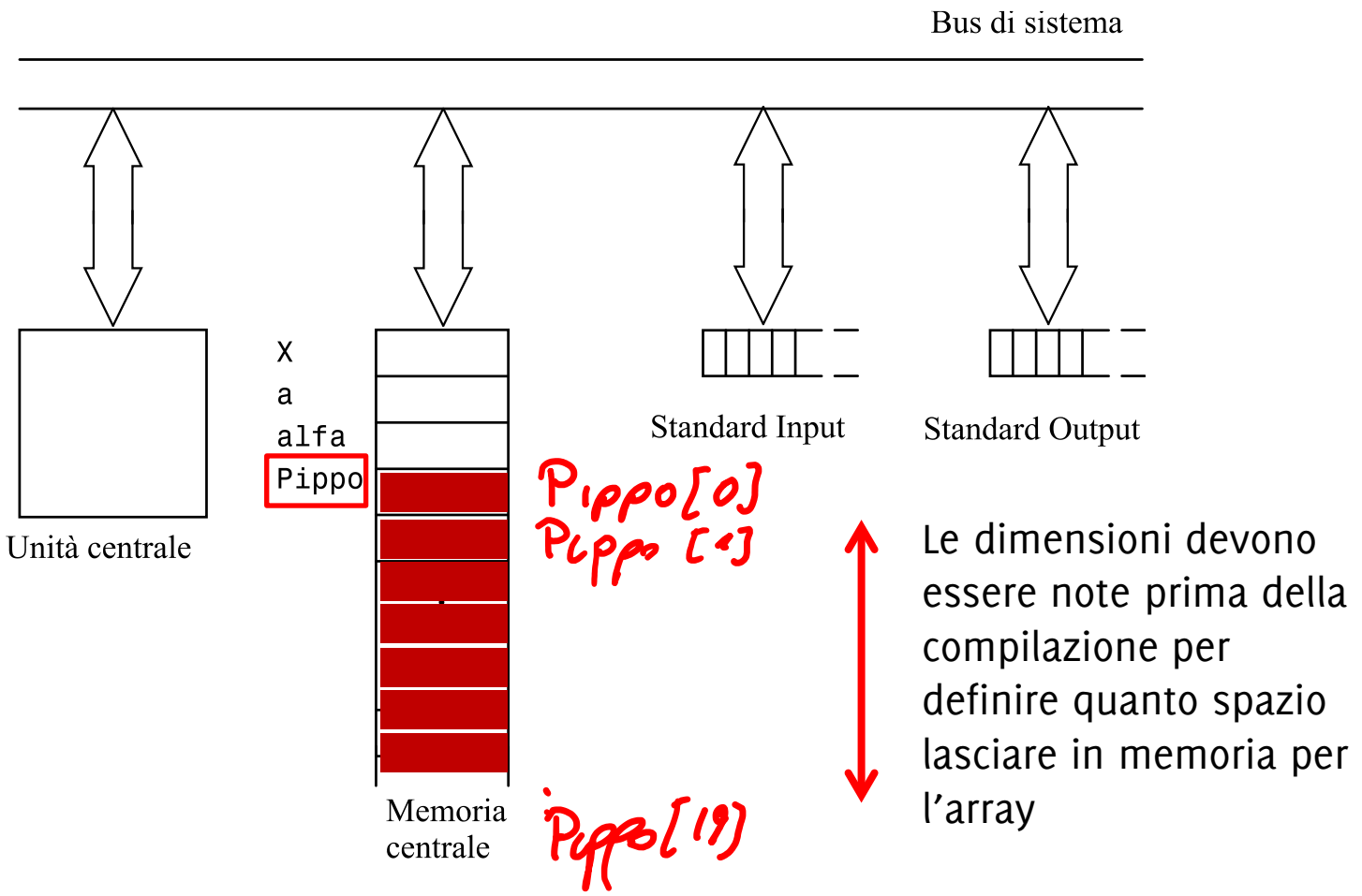
vet
134
34
123
43215
2365
-145
523
45

stringa
‘a’
‘K’
‘\n’
‘3’
‘\t’

resti
2.45
3.24
4.23
1245.2
236.5
-5.0
43.53
0

Lo spazio allocato per gli array

```
int Pippo[20];
```



Le dimensioni devono essere note prima della compilazione per definire quanto spazio lasciare in memoria per l'array

Accedere agli elementi dell'array

È possibile accedere agli elementi dell'array **specificando un indice tra quadre `[]`**

```
int vet[20];
```

`vet[0]` è il primo elemento della sequenza

`vet[19]` è l'ultimo elemento della sequenza

Fondamentale:

Ogni elemento dell'array è una variabile del tipo dell'array!

`vet[7]` conterrà un valore intero

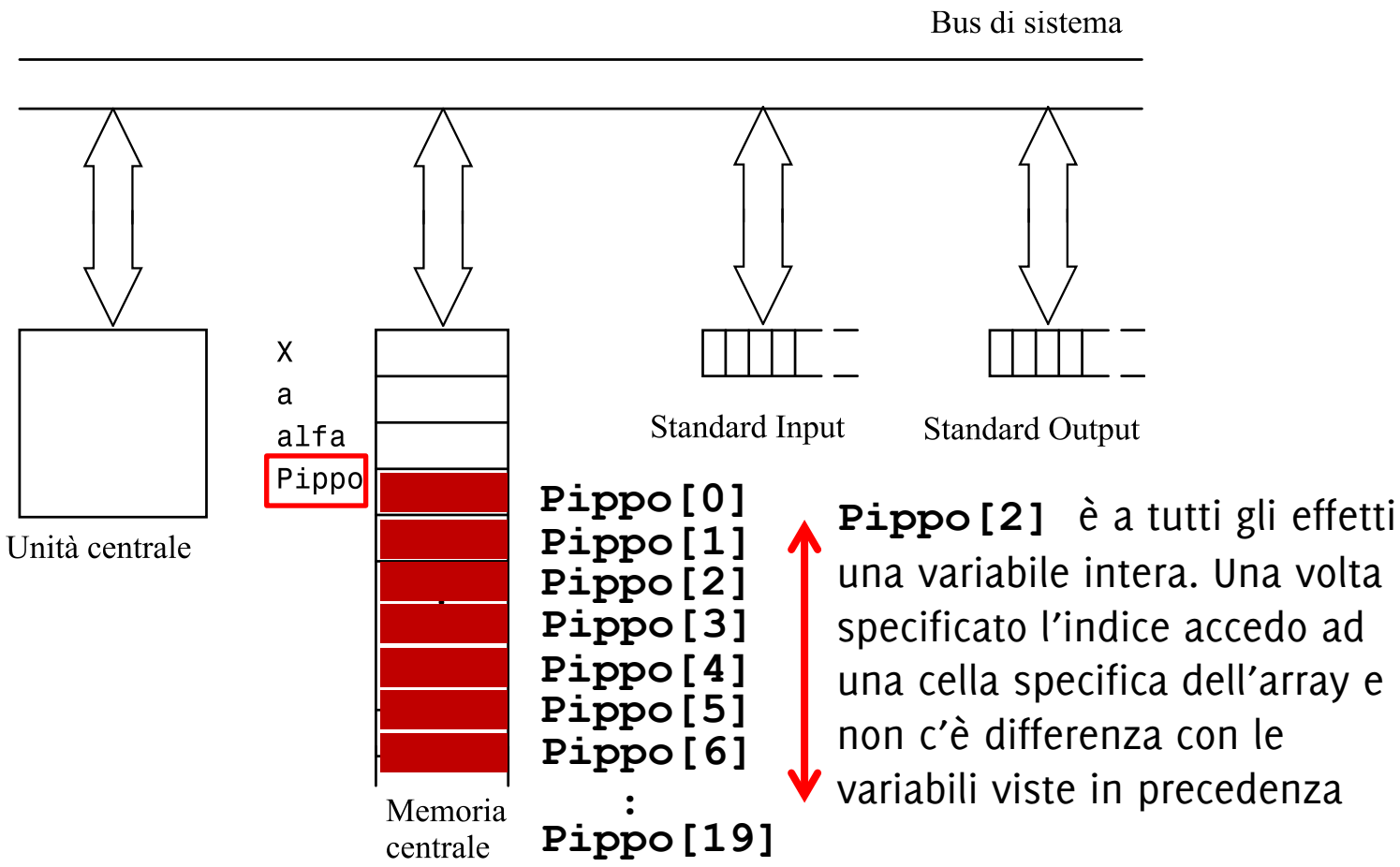
Una volta **fissato l'indice**, non c'è differenza tra un elemento dell'array ed una qualsiasi **variabile** dello stesso tipo

```
int a; a = vet[0]; vet[0] = a; vet[0] += a;
```

NB in C gli array sono indicizzati a partire da o

Lo spazio allocato per gli array

```
int Pippo[20];
```



Pippo[2] è a tutti gli effetti una variabile intera. Una volta specificato l'indice accedo ad una cella specifica dell'array e non c'è differenza con le variabili viste in precedenza

Accedere agli elementi dell'array

Il valore **dell'indice** è di tipo **integral** (`char` , `int`)

È quindi possibile utilizzare una **variabile per definire l'indice** all'interno dell'array.

Dato:

```
int vet[20]; int i = 0;
```

L'espressione: `vet[i]` va interpretata nel seguente modo:

1. Leggi il valore di `i` (o valuta l'espressione tra quadre)
2. Accedi all'elemento di `vet` alla posizione di indice `i`
3. Leggi il valore che trovi in quella cella di memoria (`vet[i]`)

con lo stesso criterio posso interpretare `vet[i + 1];`

Esempi di Operazioni su Array

Una volta **fissato l'indice** in un array si ha una **variabile** del **tipo dell'array** che può essere usata per

- Assegnamenti

```
vet[2] = 7; vet[4] = 8 % 3;
i = 0; vet[i] = vet[i+1];
```

- operazioni logiche

```
vet[0] == vet[9]; vet[1] < vet[4];
```

- operazioni aritmetiche

```
vet[0] == vet[9] / vet[2] + vet[1] / 6;
```

- operazioni di I/O

```
scanf("%d", &vet[9]);
printf("valore pos %d = %d", i, vet[i]);
```

Assegnamento tra Array

Es: Scrivere un frammento di codice per dichiarare un array di dimensione 3 e per scrivere in ogni variabile un numero (da 1 a 3) corrispondente alla posizione della cella.

Assegnamento tra Array

Es: Scrivere un frammento di codice per dichiarare un array di dimensione 3 e per scrivere in ogni variabile un numero (da 1 a 3) corrispondente alla posizione della cella.

```
int vet[3];
vet[0] = 1;
vet[1] = 2;
vet[2] = 3;
```

.. e senza Array

```
int a,b,c;
a = 1;
b = 2;
c = 3;
```

vs

```
int vet[3];
vet[0] = 1;
vet[1] = 2;
vet[2] = 3;
```

Come faccio a richiamare "il secondo valore inserito"?

- Con le variabili devo salvare da qualche parte che **a** contiene il primo valore, **b** il secondo… perché le variabili non hanno un ordinamento
- Con il vettore mi basta accedere a **vet[1]** perché gli elementi di un vettore seguono un ordinamento

.. e senza Array

```
int a,b,c;
a = 1;
b = 2;
c = 3;
```

vs

```
int vet[3];
vet[0] = 1;
vet[1] = 2;
vet[2] = 3;
```

La soluzione diventa decisamente impraticabile quando si richiedono molte variabili: occorre usare array

- perché sono indicizzati
- perché posso popolarli/elaborarli con un ciclo

Con i vettori tipicamente il **`for`** risulta più comodo del **`while`** perché la variabile del ciclo viene usata per indicizzare gli elementi dell'array

Esempio

Scrivere un programma che dichiara un array di dimensione 300 e scrive in ogni cella un numero da 1 a 300.

Esempio

Scrivere un programma che dichiara un array di dimensione 300 e scrive in ogni cella un numero da 1 a 300.

```
#include<stdio.h>
int main()
{
    int vet[300]; int i;
    for (i = 0; i < 300 ; i++)
      vet[i] = i + 1 ;
    return 0;
}
```

Tipico uso del for per scorrere un array

Il valore dell'array

Abbiamo visto che gli elementi dell'array contengono valori del tipo dell'array.

Quando scrivo `int vet[300];`

-> So che in `vet[0]` troverò un intero.

Il valore dell'array

Abbiamo visto che gli elementi dell'array contengono valori del tipo dell'array.

Quando scrivo `int vet[300];`

-> So che in `vet[0]` troverò un intero.

Cosa c'è invece in `vet`?

-> L'indirizzo del primo elemento in memoria, i.e.

```
vet == &vet[0];
```

È possibile stampare un indirizzo usando `"%p"`

```
printf("\n -> vet      %p", vet);
```

Il valore dell'array

Abbiamo visto che gli elementi dell'array contengono valori del tipo dell'array.

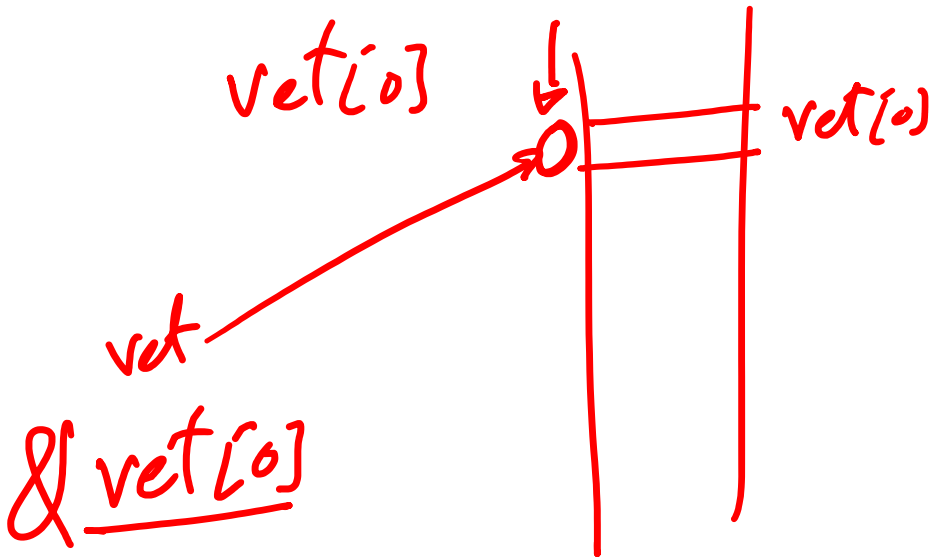
Quando scrivo `int vet[300];`

-> So che in **vet[0]** troverò un intero.

Cosa c'è invece in **vet**?

-> L'indirizzo del primo elemento in memoria, i.e.

vet == &vet[0];



È possibile stampare un indirizzo usando **"%p"**

```
printf("\n -> vet        %p", vet);
```

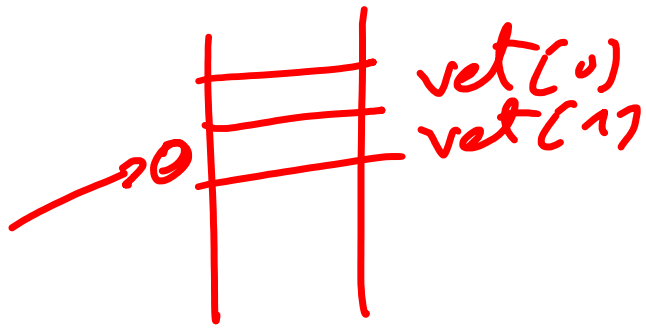
Indirizzi nei vettori

```c
#include<stdio.h>
#define N 300
int main()
{
int vet[N], i;
printf("\nvet contiene l'indirizzo della prima cella");
printf("\n -> vet     %p", vet); // %p place-holder per stampare indirizzi
printf("\n -> &vet[0] %p", &vet[0]);
printf("\ngli elementi dell'array sono in posizioni consecutive");
printf("\n -> &vet[1] %p", &vet[1]);

return 0;

}
```

```
"C:\Users\Giacomo\Dropbox (DEIB)\Didattica\2021_Informatica_A_Boracchi\Lez7\indirizziLive.exe"

vet contiene l'indirizzo della prima cella
 -> vet     000000000061F960
 -> &vet[0] 000000000061F960
gli elementi dell'array sono in posizioni consecutive
 -> &vet[1] 000000000061F964
Process returned 0 (0x0)   execution time : 0.093 s
Press any key to continue.
```

Indirizzi nei vettori

```c
#include<stdio.h>
#define N 300
int main()
{
int vet[N], i;
printf("\nvet contiene l'indirizzo della prima cella");
printf("\n -> vet     %p", vet); // %p place-holder per stampare indirizzi
printf("\n -> &vet[0] %p", &vet[0]);
printf("\ngli elementi dell'array sono in posizioni consecutive");
printf("\n -> &vet[1] %p", &vet[1]);

return 0;

}
```




```
"C:\Users\Giacomo\Dropbox (DEIB)\Didattica\2021_Informatica_A_Boracchi\Lez7\indirizziLive.exe"

vet contiene l'indirizzo della prima cella
 -> vet     000000000061F960
 -> &vet[0] 000000000061F960
gli elementi dell'array sono in posizioni consecutive
 -> &vet[1] 000000000061F964
Process returned 0 (0x0)   execution time : 0.093 s
Press any key to continue.
```

Numero in esadecimale

Posizioni consecutive dell'array sono consecutive anche in memoria

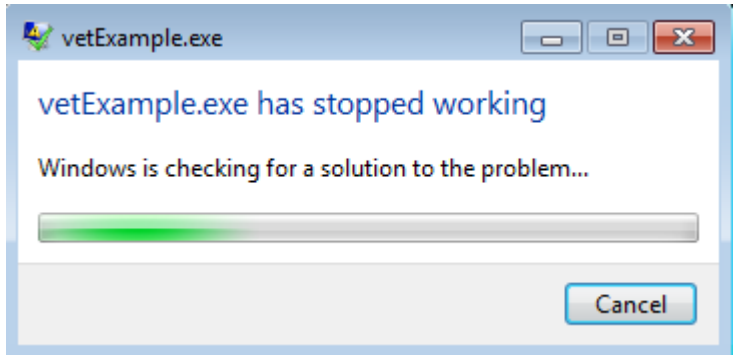
Le Dimensioni degli Array

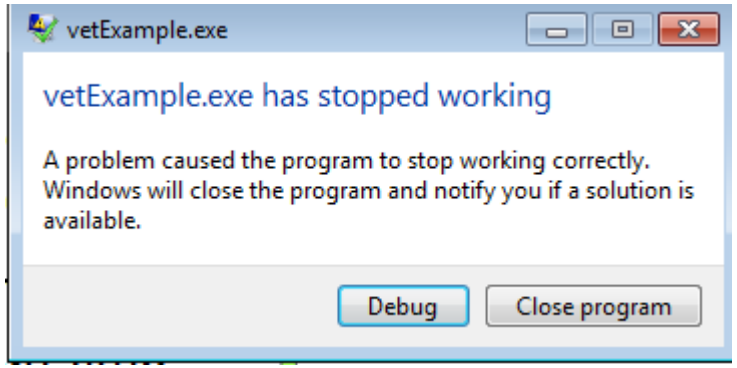
Non è possibile accedere ad un elemento dell'array ad una posizione superiore alla dimensione:

```
int vet[20];
```

scrivere **`vet[40]`** (o anche solo **`vet[20]`** visto che le 20 celle vanno da **`vet[0]`** a **`vet[19]`**).

In tal caso si ha **segmentation fault**, che nella migliore delle ipotesi si manifesta **solamente** a run-time (come quando si dimentica **`&`** in **`scanf(…)`**).





Errore! (togliamo punti)

```
int dim; /* il valore a dim è associato solo durante
l'esecuzione */

scanf("%d", &dim);

float resti[dim]; /* le dimensioni dell'array devono
essere note a priori */
```

Il Costruttore Array, COSE DA NON FARE!

Errore

```
int dim; /* il valore a dim è associato solo durante
l'esecuzione */

scanf("%d", &dim);

float resti[dim]; /* quindi il compilatore non sa quanto
spazio riservare in memoria per resti */
```

Anche se il gcc lo permette, questo NON è da fare perché espone il codice a vulnerabilità

Errore! (togliamo punti)

```
int dim; /* il valore a dim è associato solo durante
l'esecuzione */

scanf("%d", &dim);

float resti[dim]; /* le dimensioni dell'array devono
essere note a priori */
```

Anche se alcune versioni del C lo permettono, i variable-length array non sono sicuri o efficienti e seguono un meccanismo di gestione della memoria che non vedremo.

Potremo fare queste operazioni grazie alla memoria dinamica.

Per chi volesse saperne di più:

https://en.wikipedia.org/wiki/Variable-length_array

#define

Spesso si ricorre alla direttiva di precompilazione **define** per dichiarare un array

```
#define NOME_DEFINE valoreNumerico
```

Prima della compilazione, ogni istanza di **NOME_DEFINE** (riferibile all'uso di variabile) verrà sostituita da **valoreNumerico**

Se dichiaro `int vet[NOME_DEFINE];` le dimensioni di **vet** sono note prima di iniziare la compilazione

L'utilizzo di **define** rende il codice più leggibile, e facilmente modificabile quando occorre cambiare la dimensione dell'array (richiede comunque la ricompilazione del codice sorgente)

NB non occorre il ; dopo **valoreNumerico**

Inizializzazione di un array nella dichiarazione

Sintassi compatta:

```
int n[5] = {0}; // tutti a zero
int n[5] = {1, 2, 3, 4, 5}; // Specifica tutti i valori
```

Inizializzazione parziale: gli elementi del vettore che seguono quelli inseriti sono **0**

```
int n[5] = {13};  //i rimanenti 4 elementi sono posti a 0
```

Inizializzare con troppi elementi tra le graffe è un errore di sintassi

Se la lunghezza dell'array è omessa, gli inizializzatori la determinano:

```
int n[] = {5, 47, –2, 0, 24};
```

è equivalente a:

```
int n[5] = {5, 47, – 2, 0, 24};
```

In tal caso la dimensione è inferita automaticamente, e si avranno 5 elementi nell'array (con indici che variano tra 0 e 4)

Inizializzazione di un array nella dichiarazione

Sintassi compatta:

```
int n[5] = {0}; // tutti a zero
int n[5] = {1, 2, 3, 4, 5}; // Specifica tutti i valori
```

Inizializzazione parziale: gli elementi del vettore che seguono quelli inseriti sono **0**

Ini

Se

è e

In

(con indici che variano tra 0 e 4)

NOTA BENE: VALE SOLO PER L'INIZIALIZZAZIONE, QUINDI CONTESTUALMENTE ALLA DICHIARAZIONE

Nelle altre istruzioni non ha nemmeno senso scrivere `vet[N]` se `N` rappresenta le dimensioni di `vet`

```c
#include<stdio.h>
#define N 5

int main()
{
    int i, v1[N], v2[N] = {1,2,3,4,5}, v3[N] = {1,2};

    printf("nessuna inizializzazione:\nv1 = [");
    for(i = 0; i < N; i++)
        printf("%d, ", v1[i]);
    printf("]\n\n");

    printf("inizializzazione totale:\nv2 = [");
    for(i = 0; i < N; i++)
        printf("%d, ", v2[i]);
    printf("]\n\n");

    printf("inizializzazione parziale:\nv3 = [");
    for(i = 0; i < N; i++)
        printf("%d, ", v3[i]);
    printf("]");

    printf("\n\nv1[N] e' fuori dal vettore: v1[N] = %d", v1[N]);
    printf("\nv2[N] e' fuori dal vettore: v2[N] = %d", v2[N]);
    printf("\nv3[N] e' fuori dal vettore: v3[N] = %d", v3[N]);

    return 0;
    }
```

```c
#include<stdio.h>
#define N 5

int main()
{
    int i, v1[N], v2[N] = {1,2,3,4,5}, v3[N] = {1,2};

    printf("nessuna inizializzazione:\nv1 = [");
    for(i = 0; i < N; i++)
        printf("%d, ", v1[i]);
    printf("]\n\n");

    printf("inizializzazione totale:\nv2 = [");
    for(i = 0; i < N; i++)
        printf("%d, ", v2[i]);
    printf("]\n\n");

    printf("inizializzazione parziale:\nv3 = [");
    for(i = 0; i < N; i++)
        printf("%d, ", v3[i]);
    printf("]");

    printf("\n\nv1[N] e' fuori dal vettore: v1[N] = %d", v1[N]);
    printf("\nv2[N] e' fuori dal vettore: v2[N] = %d", v2[N]);
    printf("\nv3[N] e' fuori dal vettore: v3[N] = %d", v3[N]);

    return 0;
    }
```

```
"C:\Users\Giacomo\Dropbox (DEIB)\Didattica\2021_Informatica_A_Boracchi\Lez7\iniz.exe"
nessuna inizializzazione:
v1 = [8, 0, 84, 0, 12260320, ]

inizializzazione totale:
v2 = [1, 2, 3, 4, 5, ]

inizializzazione parziale:
v3 = [1, 2, 0, 0, 0, ]

v1[N] e' fuori dal vettore: v1[N] = 0
v2[N] e' fuori dal vettore: v2[N] = 0
v3[N] e' fuori dal vettore: v3[N] = 0
Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

Una precisazione

Si possono usare **espressioni** come indici:

se **`x = 3, y=7 e z=4`**

`vett[5–2]` è uguale a **`vett[3]`**

ed è uguale a **`vett[x]`**

ed è uguale a **`vett[y-z]`**

Semplici algoritmi per gestire array

Acquisizione,

Stampa,

Assegnamento,

Confronto,

Copia

Copia «senza lasciare buchi»

Acquisire un Array

Esempio: Acquisizione di un array

Non esistono funzioni/comandi per acquisire un array di numeri (i.e., l'omologo di **`scanf("%d"..)`**

Esempio: Acquisizione di un array

Non esistono funzioni/comandi per acquisire un array di numeri (i.e., l'omologo di **scanf("%d"..)**

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
      int  v1 [MAX_LEN];
      int i;
      for(i = 0; i < MAX_LEN; i++)
      {
       printf("Inserire elemento posizione %d" , i+1);
       scanf("%d" , &v1[i]);
      }
}
```

Esempio

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
        int   v1 [MAX_LEN];
        int i;
        for(i = 0; i < MAX_LEN; i++)
        {
         printf("Inserire elemento posizione %d" , i+1);
         scanf("%d" , &v1[i]);
        }
}
```

Uso **MAX_LEN** come una costante nel codice

Esempio

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
        int  v1 [MAX_LEN];
        int i;
        for(i = 0; i < MAX_LEN; i++)
        {
         printf("Inserire elemento posizione %d" , i+1);
         scanf("%d" , &v1[i]);
        }
}
```

L'acquisizione con **scanf** avviene come per una qualsiasi variabile intera

Uso **MAX_LEN** come una costante nel codice

Esempio

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
        int  v1 [MAX_LEN];
        int i;
        for(i = 0; i < MAX_LEN; i++)
        {
         printf("Inserire elemento posizione %d" , i+1);
         scanf("%d" , &v1[i]);
        }
}
```

Dettaglio per evitare di stampare «Inserire elemento posizione 0»

Uso **MAX_LEN** come una costante nel codice

L'acquisizione con **scanf** avviene come per una qualsiasi variabile intera

Le Dimensioni degli Array: Effettive *vs* Reali

Distinguere tra dimensioni **reali** e dimensioni **effettive**

Le dimensioni **reali** sono quelle con cui viene **dichiarato un array**. Sono fissate prima della compilazione, non modificabili. Si fissano «grandi a sufficienza»

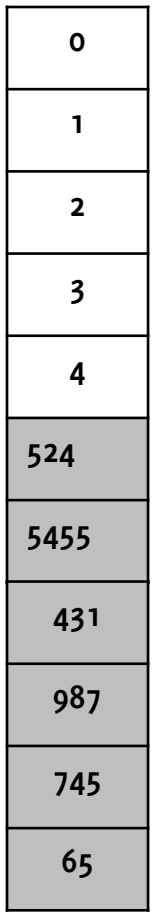
Le dimensioni **effettive** delimitano la **parte dell'array che si utilizzerà** durante l'esecuzione.

- Possono essere specificate dall'utente in una variabile (previo controllo di compatibilità con quelle reali e.g., con `do while`)

Esempio: modificare il programma precedente richiedendo prima all'utente quanti elementi inserire nell'array

Le Dimensioni degli Array: Effettive *vs* Reali

Esempio: `int v1[11];` con dimensioni effettive `n = 5;`



Dimensioni effettive dell'array: le celle che vanno da o a n (specificato dall'utente in una variabile)

Dimensioni reali dell'array: definite da **`MAX_LEN`**

Esempio

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
      int  v1 [MAX_LEN];
      int i, n; // n contiene le dimensioni effettive
      do
      {
       printf("quanti numeri vuoi inserire?");
       scanf("%d" , &n);
      }while(n < 0 || n > MAX_LEN);

      for(i = 0; i < n; i++)
      {
       printf("Inserire elemento posizione %d" , i+1);
       scanf("%d" , &v1[i]);
      }
}
```

Esempio

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
      int  v1 [MAX_LEN];
      int i, n; // n contiene le dimensioni effettive
      do
      {
       printf("quanti numeri vuoi inserire?");
       scanf("%d" , &n);
      }while(n < 0 || n > MAX_LEN);

      for(i = 0; i < n; i++)
      {
       printf("Inserire elemento posizione %d" , i+1);
       scanf("%d" , &v1[i]);
      }
}
```

Sono certo che **n** è compatibile con le dimensioni reali di **v1**

Esempio

```
#define MAX_LEN 30
#include <stdio.h>
void main()
{
      int  v1 [MAX_LEN];
      int i, n; // n contiene le dimensioni effettive
      do
      {
       printf("quanti numeri vuoi inserire?");
       scanf("%d" , &n);
      }while(n < 0 || n > MAX_LEN);

      for(i = 0; i < n; i++)
      {
       printf("Inserire elemento posizione %d" , i+1);
       scanf("%d" , &v1[i]);
      }
}
```

Sono certo che **n** è compatibile con le dimensioni reali di **v1**

Da qui in poi opero solo sulle prime **n** celle, quelle che vanno dall'indice **0** ad **n-1**

Stampa di Array

Stampa dei valori dell'array

In generale non esiste un fattore di conversione per stampare gli array. Quindi occorre procedere iterando

Stampa dei valori dell'array

In generale non esiste un fattore di conversione per stampare gli array. Quindi occorre procedere iterando

Assumiamo che l'array **v1** abbia dimensioni effettive **n**

```
printf("\nHai inserito: [");
    for(i = 0 ; i <n ; i++)
        printf(" %d ", v1[i]);
     printf("]");
```

Per le stringhe questo non è necessario (ma è comunque possibile seguire questa strada)

Esercizio TODO

Scrivere un programma che richiede all'utente di inserire una sequenza di numeri, specificando anticipatamente il numero di elementi che si intende inserire (controllare che sia compatibile con le dimensioni massime, 100)

Il programma

- Calcola il massimo della sequenza
- Stampa tutti gli elementi inseriti in ordine contrario.
- Conta le occorrenze del massimo
- Stampa tutti i numeri dall'ultima occorrenza del massimo fino all'ultimo elemento dell'array

Semplici Ricerche su Array

Esempi sugli array

Dichiarazione del vettore:

```
int a[20], i;
```

Inizializzazione del vettore:

```
for (i = 0; i < 20; i++) /* come int a[20] = {0}*/
     a[i] = 0;
```

Oppure (lettura da terminale):

```
for (i = 0; i < 20; i++) {
     printf ("Scrivi un intero: ");
     scanf ("%d", &a[i]);
}
```

Esempi sugli array

Ricerca del massimo:

```
max = a[0];
for (i = 1; i <= 19; i++)
     if (a[i] > max)
          max = a[i];
```

Calcolo della media:

```
media = 0;
for (i = 0; i <= 19; i++)
     media = media + a[i];
media = media / 20;
```

Esempi sugli array

Calcolo di massimo, minimo e media di un vettore

È sufficiente una sola scansione del vettore (un solo ciclo)

```
int a[20], max, min, i;
float media;
…
media = a[0];
max = a[0];
min = a[0];
 for ( i = 1; i <= 19; i++ ) {
        media = media + a[i];
        if ( a[i] > max )
                max = a[i];
           if ( a[i] < min )
                min = a[i];
        }
        media = media / 20;
```

Back to warm up…

Scrivere un programma per conteggiare quanto la vostra aula ha speso in totale per il pranzo Mercoledì scorso.

Calcolare la spesa media per il pranzo

Si supponga di «fare alla romana» e che quindi tutti devono pagare il prezzo medio. Dire a chi ha pagato di quanto deve ricevere e a chi ha pagato quanto deve versare.

```c
#include<stdio.h>
#include<limits.h>
int main()
{   int x[10], i, N, max, min, tot = 0;
    float media;
    min = INT_MAX; max = INT_MIN;

    do{ /*Acquisizione "sicura" della dimensione effettiva*/
        printf("inserire N: ");
        scanf("%d", &N);
    }while(N <= 0 || N > 10);

    for(i = 0; i < N; i++)
    { /*acquisizione dell'intero array*/
        printf("inserire x: "); scanf("%d", &x[i]);

        if(x[i] < min)
            min = x[i];
        if (x[i] > max)
            max = x[i];
        tot = tot + x[i];
    }
    media = 1.0 * tot / N;

    printf("\nspesa massima: %d, minima: %d, media:%.2f", max, min, media);
    for(i = 0; i < N; i++)     /*scorro nuovamente l'array per fare il conto "alla romana"*/
        printf("\n studente %d ha speso %d e riceve %.2f", i, x[i], x[i]-media);
    return 0;}
```

Un nuovo problema

Invertire una sequenza di 100 interi introdotta dall'utente (stdin)

- Con un array?
- Senza array?

```c
/* Programma InvertiSequenza */
#include <stdio.h>
int main() {
        int i;
        int a[100];
        i = 0;
        while (i < 100) {
                printf("fornisci un valore intero");
                scanf("%d", &a[i]);
                i++;
        }
        i--;
        while (i >= 0) {
                printf("%d\n", a[i]);
                i--;
        }
        return 0;
}
```

esaminare criticamente i cicli!

Ricerca di un elemento in un array

```
/* Frammento di programma.
    Verifica la presenza di un elemento in un vettore.
    Se è presente, stampa l'indice della prima occorrenza;
    se non è presente, stampa -1*/
```

Ricerca di un elemento in un array

```
/* Frammento di programma.
    Verifica la presenza di un elemento in un vettore.
    Se è presente, stampa l'indice della prima occorrenza;
    se non è presente, stampa -1*/

int dato; int risultato = -1,n;
int array[size];
……..
for ( n=0; n < size; n++ )
     if (array[n] == dato) {
          risultato = n;
          break;
     }
printf("risultato = %d\n", risultato);
```

Ricerca sequenziale

…

Biblioteca…

Ricerca di un elemento in un array

```
/* Frammento di programma.
    Verifica la presenza di un elemento in un vettore.
    Se è presente, stampa l'indice della prima occorrenza;
    se non è presente, stampa -1*/

int dato; int risultato = -1,n;
int array[size];
……..
for ( n=0; n < size && risultato == -1; n++ )
     if (array[n] == dato)
          risultato = n;

printf("risultato = %d\n", risultato);
```

Ricerca sequenziale

…

Biblioteca…

Con variabile di flag

Assegnamento Tra Array

Assegnamento tra array

Non c'è un modo per direttamente assegnare **tutti** i valori in un primo array ad un secondo array

```
#include <stdio.h>
void main()
{     int v1[300], v2[300];
      int i;
      for(i = 0 ; i < 300 ; i++)
            v1[i] = i+1;
      v2 = v1;
}
```

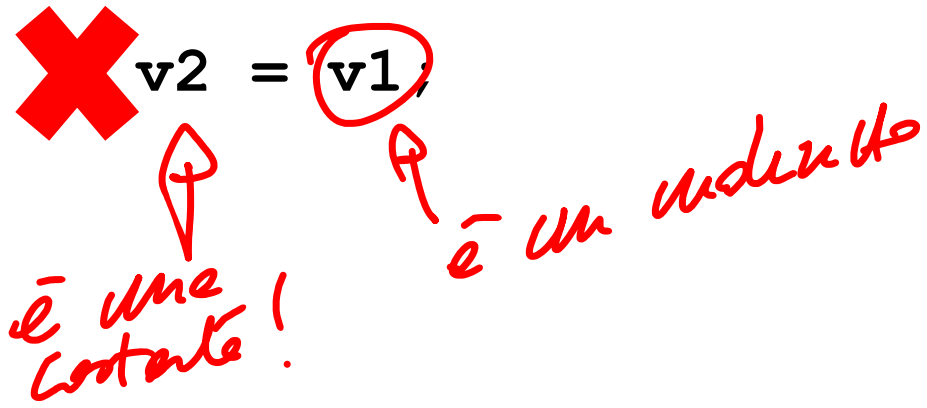
Assegnamento tra array

Non c'è un modo per direttamente assegnare **tutti** i valori in un primo array ad un secondo array

```
#include <stdio.h>

void main()
{     int v1[300], v2[300];
      int i;
      for(i = 0 ; i < 300 ; i++)
            v1[i] = i+1;
      v2 = v1;
}
```

```
assignment to expression with
array type|
```

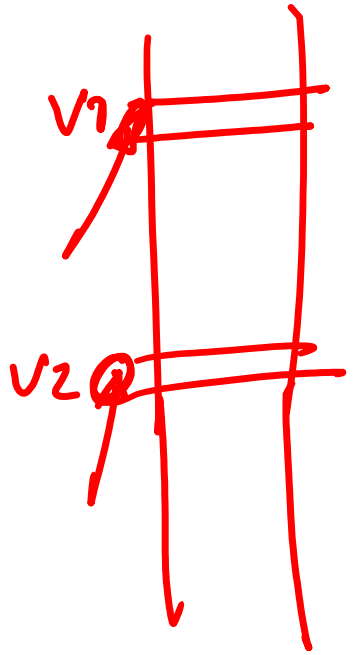
```
incompatible types
when assigning to
type 'int[300]' from
type 'int *'
```



Assegnamento tra array

Non c'è un modo per direttamente assegnare **tutti** i valori in un primo array ad un secondo array

```
#include <stdio.h>
void main()
{     int v1[300], v2[300];
      int i;
      for(i = 0 ; i < 300 ; i++)
            v1[i] = i+1;
      v2 = v1;
}
```



0000 F8AC =

Assegnamento tra array

Non c'è un modo per direttamente assegnare **tutti** i valori in un primo array ad un secondo array

```
#include <stdio.h>
void main()
{     int v1[300], v2[300];
      int i;
      for(i = 0 ; i < 300 ; i++)
            v1[i] = i+1;
      v2 = v1;
}
```

I messaggi di errore ci dicono che **v2** contiene un indirizzi **e che il valore dell'indirizzo è costante!**

Non è possibile modificare l'indirizzo delle variabili (non lo decidiamo noi in C)!!!

Quindi non è possibile modificare il valore contenuto **v2** (inteso come il suo indirizzo… il contenuto delle celle che partono dall'indirizzo è modificabile perché è una variabile)

Assegnamento tra array

Occorre operare su ogni singolo elemento dell’array!

Assegnamento tra array

Occorre **operare su ogni singolo elemento** dell'array!

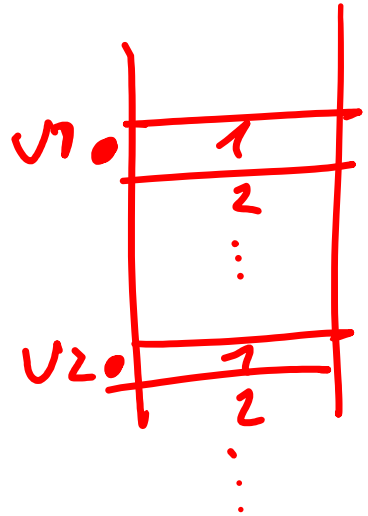
```
#define MAX_LEN 30
#include <stdio.h>
void main()
{       int  v1 [MAX_LEN], v2[MAX_LEN];
        int i;
        // popolo v1
        for(i = 0; i < MAX_LEN; i++)
                v1[i] = i;
        // copio i valori in v2
        for(i = 0; (i < MAX_LEN) ; i++)
                v2[i] = v1[i];
        // stampo
        for(i = 0; (i < MAX_LEN) ; i++)
                printf("\nv1[%d] = %d , v2[%d] = %d", i,
                        v1[i], i, v2[i]);
}
```

Confronto Tra Array

Confronto tra array

Non c'è un modo per direttamente confrontare **tutti** i valori in due array

```
#include <stdio.h>
void main()
{     int v1[300], v2[300];
      int i;
      for(i = 0 ; i < 300 ; i++)
            { v1[i] = i+1;
              v2[i] = v1[i]; }
      if (v1 == v2)
            printf("ok"); }
```



non da errore di compilazione ma non fa quello che vorremmo...

Confronto tra array

Occorre operare su **ogni singolo elemento**!

Confronto tra array

Occorre operare su **ogni singolo elemento**!

```
#define MAX_LEN 300
#include <stdio.h>
void main()
{       int  v1 [MAX_LEN], v2[MAX_LEN];
        int i, uguali;
        for(i = 0; i < MAX_LEN; i++)
        {
                v1[i] = i+1;
                v2[i] = v1[i];
        }
        uguali = 1;
        for(i = 0; (i < MAX_LEN) && uguali; i++)
                if(v1[i] != v2[i])
                        uguali = 0;
        if(uguali)
                printf("ook!");}
```

Confronto tra array

Occorre operare su **ogni singolo elemento**: quindi

```
#define MAX_LEN 300
#include <stdio.h>
void main()
{       int  v1 [MAX_LEN], v2[MAX_LEN];
        int i, uguali;
        for(i = 0; i < MAX_LEN; i++)
        {
                v1[i] = i+1;
                v2[i] = v1[i];
        }
        uguali = 1;
        for(i = 0; (i < MAX_LEN) && uguali; i++)
                if(v1[i] != v2[i])
                        uguali = 0;
        if(uguali)
                printf("ook!");}
```

Variabile di flag, diventa **0** appena trova una cella per cui **v1** e **v2** differiscono

Scorro tutti gli elementi dei vettori. Mi arresto appena trovo due elementi diversi

Sta per (**uguali != 0**)

Variabili di Flag per Verificare Condizioni su Array

Per controllare che una condizione (uguaglianza in questo caso) sia soddisfatta da tutti gli elementi del vettore

```
uguali = 1;
      for(i = 0; (i < MAX_LEN); i++)
            if(v1[i] != v2[i])
                  uguali = 0;
```

Al termine del ciclo, se uguali è rimasta 1 sono certo che la condizione da verificare **non è mai stata negata** (i.e.,`v1[i] != v2[i]` è sempre stata falsa). Quindi che **tutti** gli elementi degli array coincidono.

La variabile di flag (`uguali`) può solo cambiare da `1` in `0`
Ovviamente il ruolo di `0` e di `1` possono essere invertiti nel codice sopra

Errore Frequente

Errore frequente: modificare il valore della variabile di flag nel anche nel verso opposto.

```
uguali = 1;
     for(i = 0; (i < MAX_LEN); i++)
          if(v1[i] != v2[i])
               uguali = 0;
          else
               uguali = 1;
```

Alla fine del ciclo se uguali è 1 posso solo concludere che l'ultima coppia di elementi controllati coincide!

Quando la condizione di permanenza nel ciclo è
`(i < MAX_LEN) && uguali` il problema non si pone ma **`else`** risulta comunque inutile

Copia «Senza Lasciare Buchi»

Copiare alcuni elementi da un array ad un altro

In molti casi è richiesto di **scorrere** un array **`v1`** e di **selezionare** alcuni valori secondo una data condizione.

Tipicamente i valori selezionati in **`v1`** vengono **copiati in un secondo array**, **`v2`**, per poter essere utilizzati.

È buona norma copiare i valori **nella prima parte** di **`v2`**, eseguendo quindi una copia «senza lasciare buchi».

È anche necessario sapere quali sono i valori significativi in **`v2`** e quali no.

Esempio : copiare i numeri pari in **`v1`** in **`v2`**

`v1`

5	6	7	89	568	68	657	989	96	98

`v2`

?	6	?	?	568	68	?	?	96	98

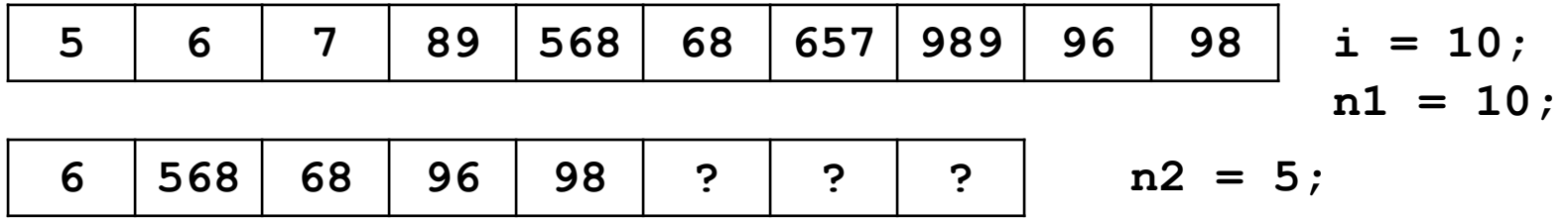
`v2`

6	568	68	96	98	?	?	?

Copiare alcuni elementi da un array ad un altro

Per fare questo è necessario usare **due indici**:

- **`i`** per **scorrere `v1`**: parte da **`0`** e arriva a **`n1`**, la dimensione effettiva di **`v1`**, con **incrementi regolari**.
- **`n2`** parte da **`0`** e viene **incrementata solo quando un elemento viene copiato.**
  - **`N2`** **indica sempre la posizione del primo elemento libero** in **`v2`**,
  - al termine, **`n2`** conterrà il **numero di elementi** in **`v2`**, quindi la sua **dimensione effettiva**



Esempio

Chiedere all'utente di inserire un array di interi (di dimensione definita precedentemente) e quindi un numero intero **`t`**. Il programma quindi:

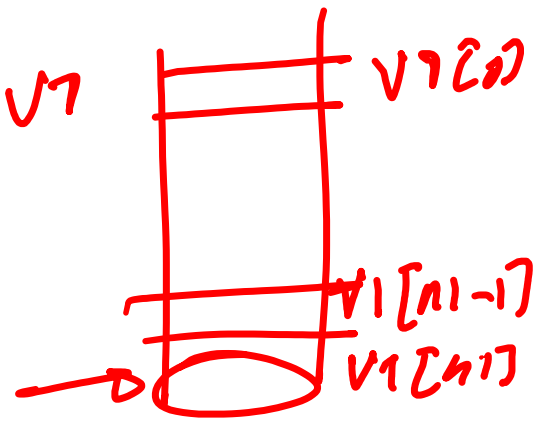
- salva gli elementi inseriti in un vettore **`v1`**.
- Copia tutti gli elementi di **`v1`** che sono maggiori di **`n`** in un secondo vettore **`v2`**.
- La copia deve avvenire nella parte iniziale di **`v2`**, senza lasciare buchi.

Esempio

Chiedere all'utente di inserire un array di interi (di dimensione definita precedentemente) e quindi un numero intero **t**. Il programma quindi:

- salva gli elementi inseriti in un vettore **v1**.
- Copia tutti gli elementi di **v1** che sono maggiori di **n** in un secondo vettore **v2**.
- La copia deve avvenire nella parte iniziale di **v2**, senza lasciare buchi.

```
j=0; // prima pos libera in v2
for(i=0; i<n1; i++)
    if(v1[i] %. 2 ==0)
        { v2[j] = v1[i];
          j++;}
n2 = j; // dim effettiva di v2
```



Esempio

```
printf("\nInserire la soglia");
scanf("%d" , &t);
n2 = 0;
for(i = 0; i < n1; i++)
if(v1[i] > t)
{
 v2[n2] = v1[i];
 n2++;//n2 è la prima posizione libera in v2
}
printf("\n Maggiori di %d sono: [" , t); //n2 ora è la
lunghezza effettiva di v2
for(i = 0 ; i <n2 ; i++)
     printf(" %d, ", v2[i]);
printf("]");
```

Tipi di Dato in C

Tipi di dato

I **tipi di dato** rappresentano:

- un insieme di **valori**
- un insieme di **operazioni** applicabili a questi

Ogni **tipi di dato diversi** hanno **rappresentazioni** in memoria differenti

- Il numero di celle/parole e la codifica utilizzata può cambiare

La memoria utilizzata per allocare le variabili di un determinato tipo cambia con la piattaforma (i.e., compilatore / sistema operativo / hardware)

Tipi di dato

Classificazione sulla base della struttura:

- **Tipi semplici**, informazione logicamente **indivisibile** (e.g. `int, char, float..`)
- **Tipi strutturati**: aggregazione di variabili di tipi semplici

Altra classificazione:

- **Built in**, tipi già presenti nel linguaggio base
- **User defined,** nuovi tipi creati nei programmi «componendo» variabili di tipo built in

Tipi di dato

Classificazione sulla base della struttura:

- **Tipi semplici**, informazione logicamente **indivisibile** (e.g. `int, char, float..`)
- **Tipi strutturati**: aggregazione di variabili di tipi semplici

Altra classificazione:

- **Built in**, tipi già presenti nel linguaggio base
- **User defined,** nuovi tipi creati nei programmi «componendo» variabili di tipo built in

Variabili e Tipo di Dato

In C **tutte le variabili** hanno un **tipo**, associato stabilmente mediante la **dichiarazione**

Il **tipo** di una variabile:

- definisce l'insieme dei **valori ammissibili**
- definisce l'insieme delle **operazioni applicabili**
- permette di **rilevare errori** al momento della compilazione
- definisce lo **spazio in memoria** allocato in corrispondenza alla variabile
  - Questa però dipende anche dalla piattaforma (i.e., compilatore + sistema operativo + hardware)

Tipi Semplici

```
char, int, float, double
```

Tipi Semplici

Ecco i **quattro tipi semplici** del C e la loro dimensione

- **`char:`** 1 Byte
- **`int`**: tipicamente 1 parola di memoria
- **`float:`** dipende dal compilatore (4 Byte spesso)
- **`double:`** dipende dal compilatore (più del **`float`**)

Qualificatori di tipo (per **`int`** e **`char`**)

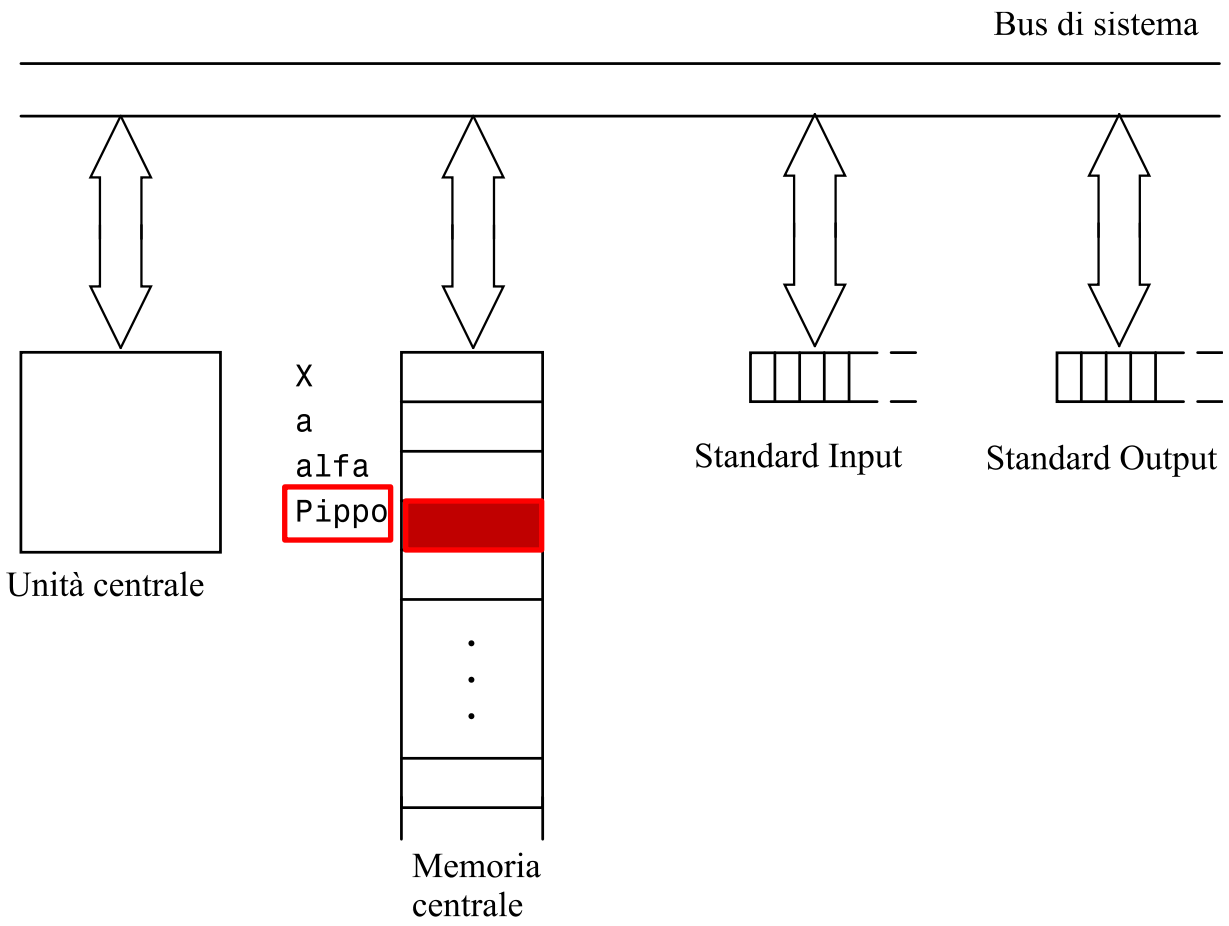
- **`signed`** utilizza una codifica con il segno (CP2)
- **`unsigned`** prevede solo valori positivi
- **NB** Allocano lo stesso spazio

Quantificatori di tipo, modificano la dimensione allocata

- **`short`** (per **`int`**)
- **`long`** (per **`int`** e **`double`**)

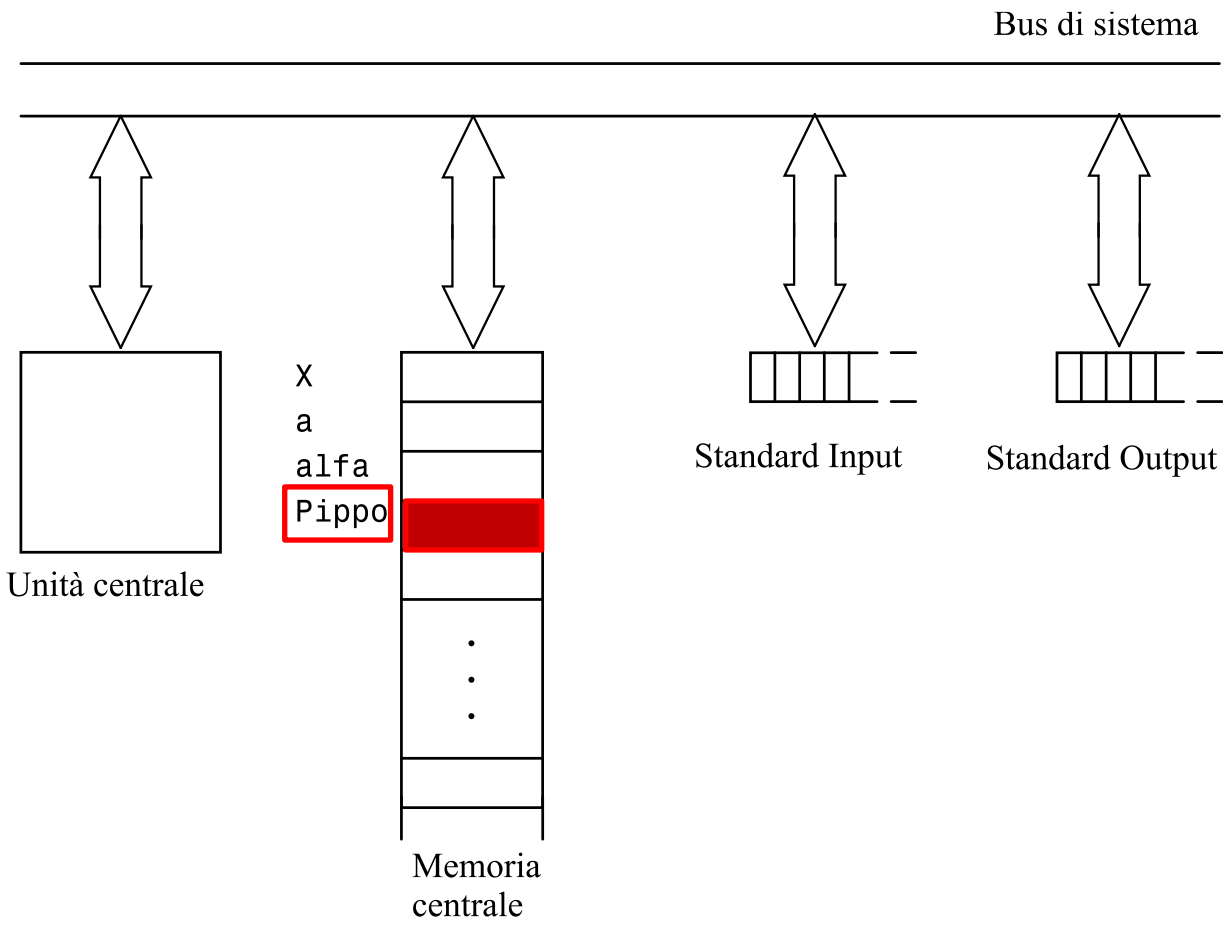
I Qualificatori, Quantificatori e lo spazio allocato

```
int Pippo; //codifica in CP2
```



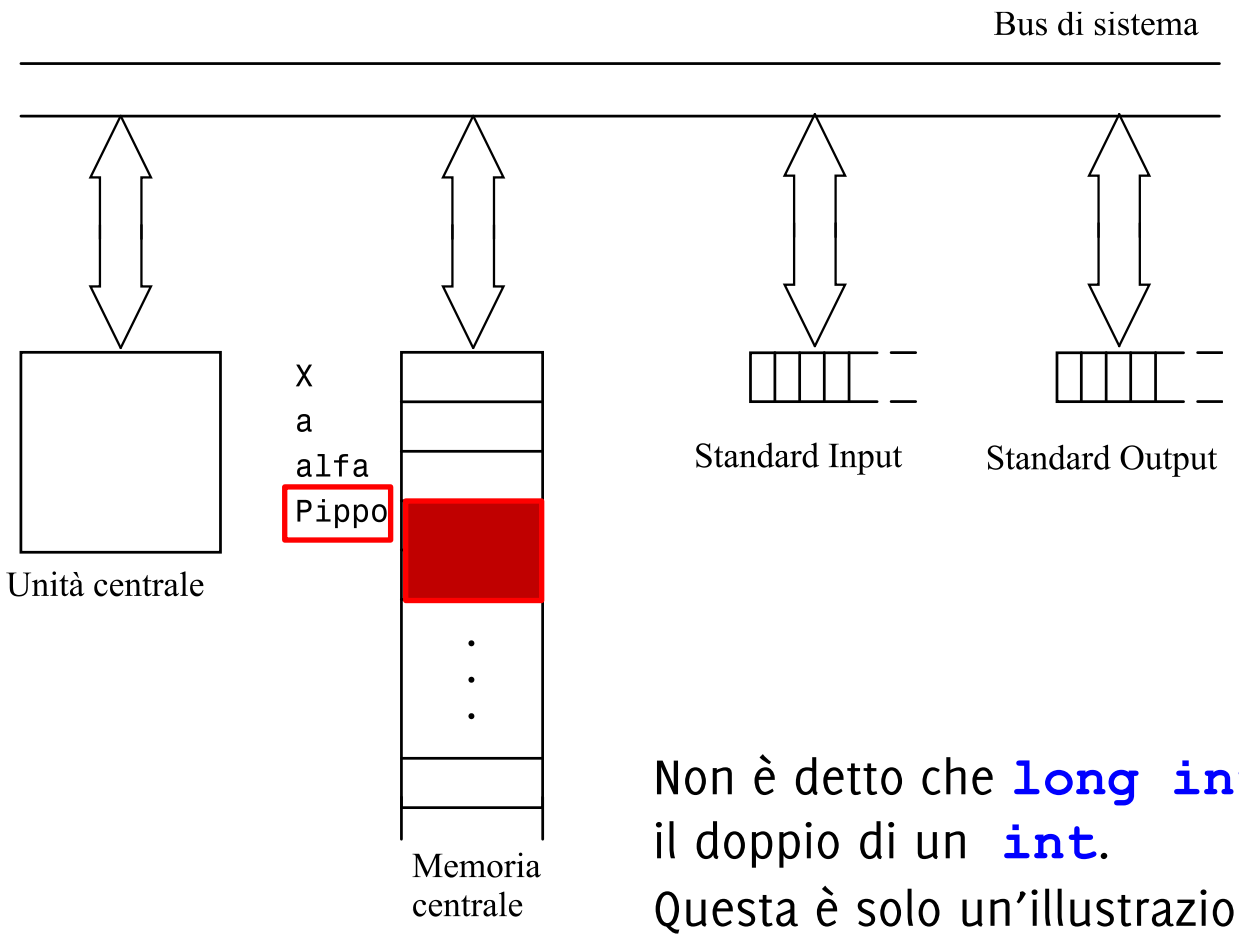
I Qualificatori, Quantificatori e lo spazio allocato

```
unsigned int Pippo; //codifica di un intero positivo
```



I Qualificatori, Quantificatori e lo spazio allocato

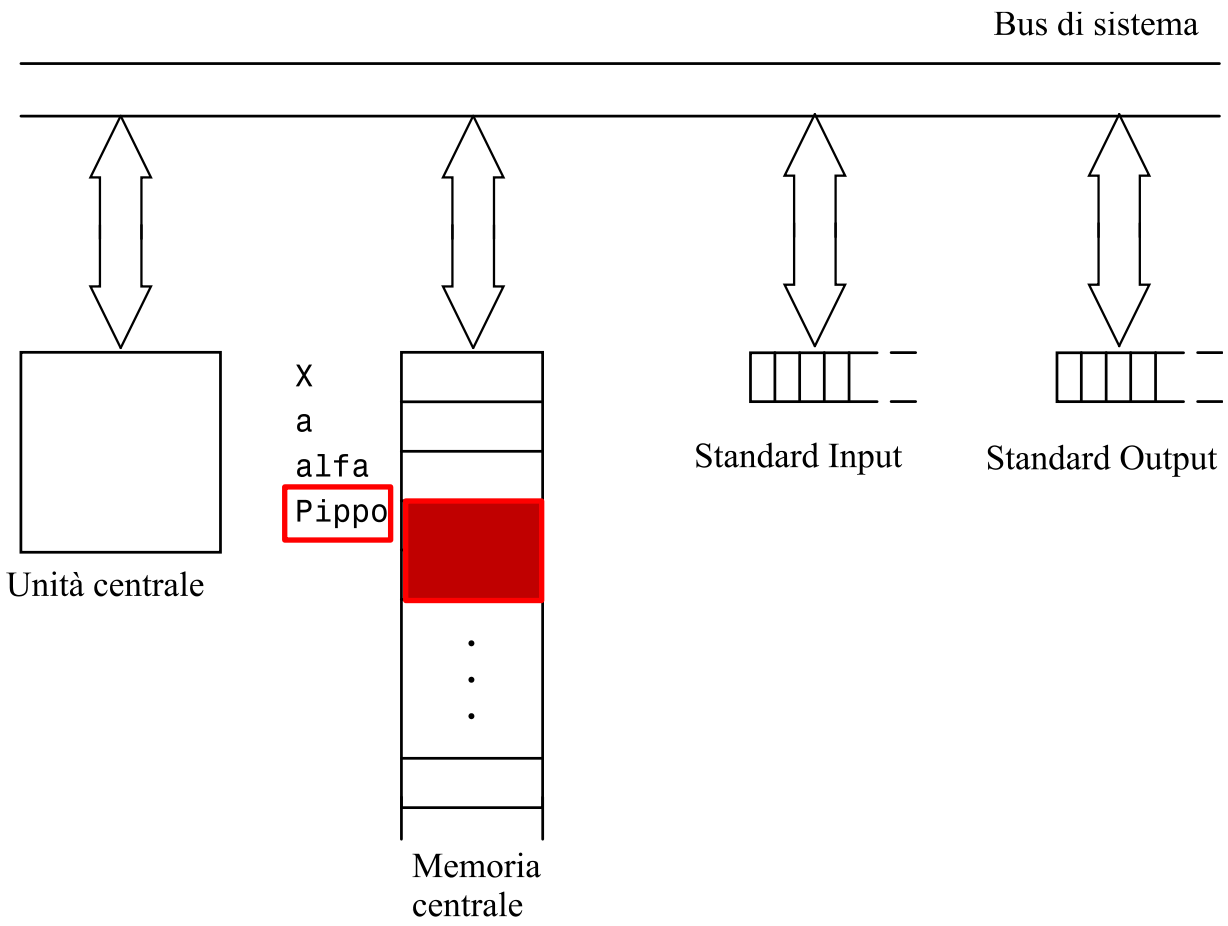
```
long int Pippo;
```



Non è detto che `long int` richieda il doppio di un `int`.
Questa è solo un'illustrazione

I Qualificatori, Quantificatori e lo spazio allocato

```
unsigned long int Pippo;
```



Il tipo `int`

Rappresentano un **sottoinsieme** di $\mathbb{N}$

Lo **spazio** allocato è tipicamente **una parola**, e dipende dalla piattaforma, oltre che dai qualificatori e quantificatori

Fatti garantiti:

- spazio (`short int`) $\leq$ spazio (`int`) $\leq$ spazio (`long int`)
- spazio (`signed int`) $=$ spazio (`unsigned int`)

Es, se la parola è a 32 bit,

- `signed int` $\{-2^{31}, \ldots, 0, \ldots +2^{31}-1\}$, i .e., 2^{32} numeri
- `unsigned int` $\{0, \ldots +2^{32}-1\}$, sempre 2^{32} numeri

Come faccio a sapere i limiti per un intero?

- `#include<limits.h>`, e richiamo le costanti **`INT_MIN,INT_MAX`**

Quando il valore di una variabile int eccede INT_MAX si ha overflow

Nota in C

Le variabili **`int`** sono **codificate in CP2**

Se aggiungo il qualificatore **`unsigned`** , tutti i bit vengono usati solo per i numeri positivi. Quindi **si usa una codifica senza segno** per coprire un range maggiore.

Provare per credere…

-1 in CP2 7 bit

0000001 (+1)

1111110 (complemento)

1111111 (sommo 1)

INT_MAX 0111111

+ 0000001

1000000

```
2 INT_MAX + 1    1111111
             +1  0000001
2 INT_MAX + 2 (1)0000000
```

Nota in C

Le variabili **`int`** sono **codificate in CP2**

Se aggiungo il qualificatore **`unsigned`** , tutti i bit vengono usati solo per i numeri positivi. Quindi **si usa una codifica senza segno** per coprire un range maggiore.

Provare per credere…

`printf("%u", u);` legge il valore di u come **`unsigned int`** e lo stampa
`printf("%d", u);` legge il valore di u come **`int`** e lo stampa

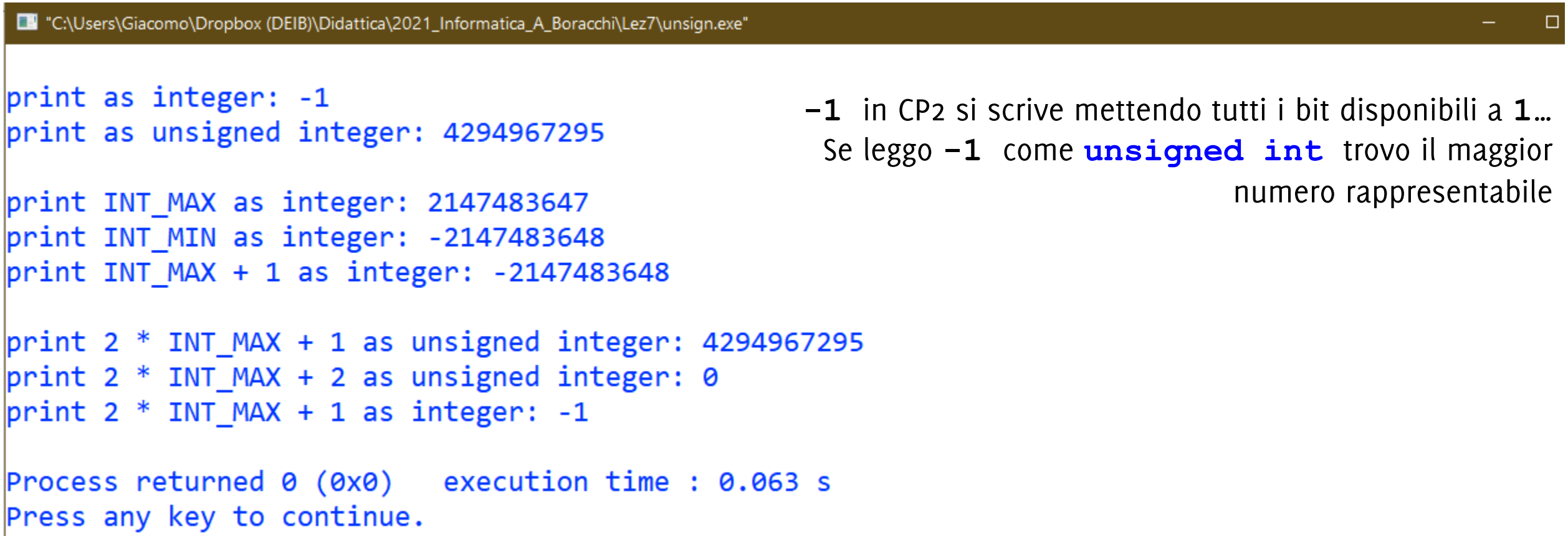
```c
int main()
{
    int u = -1;
    printf("\nprint as integer: %d", u);
    printf("\nprint as unsigned integer: %u", u);

    printf("\n\nprint INT_MAX as integer: %d", INT_MAX);
    printf("\nprint INT_MIN as integer: %d", INT_MIN);
    printf("\nprint INT_MAX + 1 as integer: %d", INT_MAX + 1); //OVERFLOW

    printf("\n\nprint 2 * INT_MAX + 1 as unsigned integer: %u", 2 * INT_MAX + 1);
    printf("\nprint 2 * INT_MAX +2 as unsigned integer: %u", 2 * INT_MAX + 2);
    printf("\nprint 2 * INT_MAX + 1 as integer: %d\n", 2 * INT_MAX + 1);
    return 0;}
```

Nota in C

Le variabili **`int`** sono **codificate in CP2**

Se aggiungo il qualificatore **`unsigned`** , tutti i bit vengono usati solo per i numeri positivi. Quindi **si usa una codifica senza segno** per coprire un range maggiore.

Provare per credere…



`-1` in CP2 si scrive mettendo tutti i bit disponibili a **`1`**…
Se leggo **`-1`** come **`unsigned int`** trovo il maggior numero rappresentabile

Nota in C

Le variabili **`int`** sono **codificate in CP2**

Se aggiungo il qualificatore **`unsigned`** , tutti i bit vengono usati solo per i numeri positivi. Quindi **si usa una codifica senza segno** per coprire un range maggiore.

Provare per credere…

```
"C:\Users\Giacomo\Dropbox (DEIB)\Didattica\2021_Informatica_A_Boracchi\Lez7\unsign.exe"

print as integer: -1
print as unsigned integer: 4294967295

print INT_MAX as integer: 2147483647
print INT_MIN as integer: -2147483648
print INT_MAX + 1 as integer: -2147483648

print 2 * INT_MAX + 1 as unsigned integer: 4294967295
print 2 * INT_MAX + 2 as unsigned integer: 0
print 2 * INT_MAX + 1 as integer: -1

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

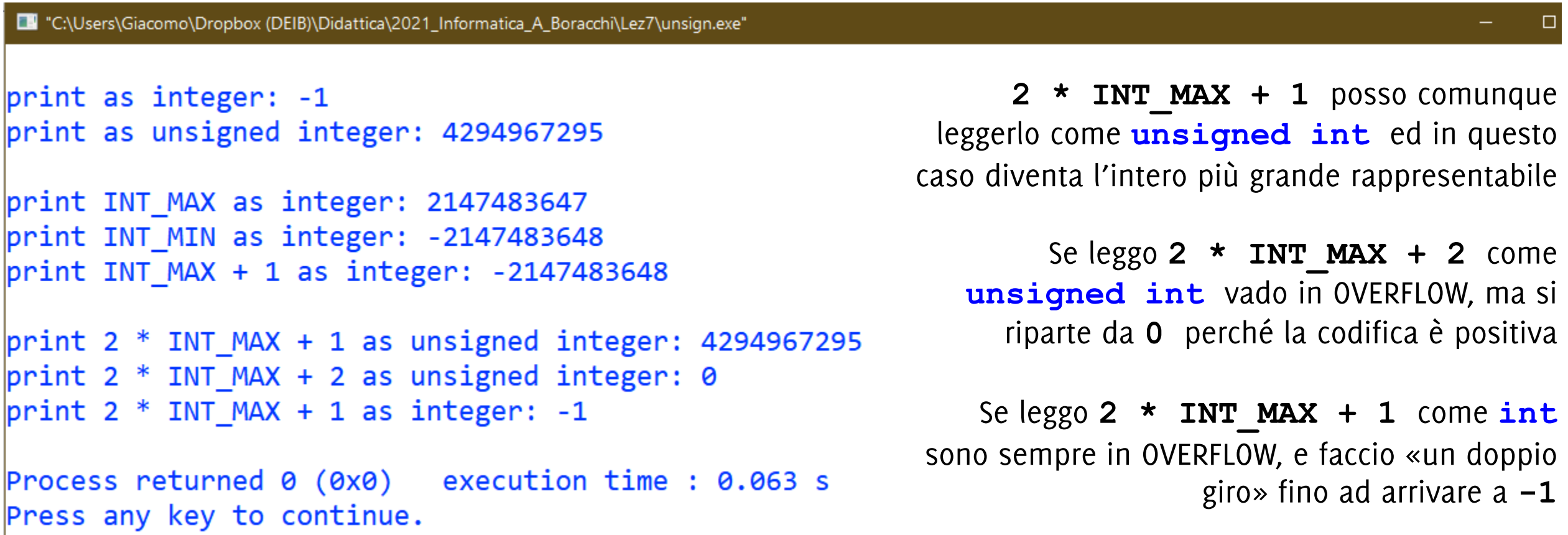
`INT_MAX` | **`INT_MIN`** sono il massimo | minimo **`int`** codificabile in CP2

`INT_MAX + 1` eccede il range dei numeri positivi, la somma con 1 porta il primo bit a zero e si ottiene un numero negativo: **OVERFLOW!**

Le variabili **`int`** sono **codificate in CP2**

Se aggiungo il qualificatore **`unsigned`** , tutti i bit vengono usati solo per i numeri positivi. Quindi **si usa una codifica senza segno** per coprire un range maggiore.

Provare per credere…



`2 * INT_MAX + 1` posso comunque leggerlo come **`unsigned int`** ed in questo caso diventa l'intero più grande rappresentabile

Se leggo **`2 * INT_MAX + 2`** come **`unsigned int`** vado in OVERFLOW, ma si riparte da **`0`** perché la codifica è positiva

Se leggo **`2 * INT_MAX + 1`** come **`int`** sono sempre in OVERFLOW, e faccio «un doppio giro» fino ad arrivare a **`-1`**

Operazioni built-in per dati di tipo `int`

=	Assegnamento di un valore int a una variabile int
+	Somma (tra int ha come risultato un int)
-	Sottrazione (tra int ha come risultato un int)
*	Moltiplicazione (tra int ha come risultato un int)
/	Divisione con troncamento della parte non intera (risultato int)
%	Resto della divisione intera
==	Relazione di uguaglianza
!=	Relazione di diversità
<	Relazione “minore di”
>	Relazione “maggiore di”
<=	Relazione “minore o uguale a”
>=	Relazione “maggiore o uguale a”

Operatoori Aritmetici

Operatoori Relazionali

Il tipo `float` e `double`

Approssimazione di $\mathbb{R}$ (che è un insieme denso), quindi i **valori** vengono approssimati per «magnitudine», e limiti nella **precisione** della rappresentazione

Nella rappresentazione in virgola mobile (floating point) il numero n si scrive come due parti separate da "E":

- m: mantissa
- e: esponente (rispetto alla base 10), tali che $n = m * 10^e$

Ad esempio, 1 780 000.000 0023 in virgola mobile:

178 000.000 000 23 E1
17 800 000 000 023 E-7
1.780 000 000 0023 E+6
ecc.

Spazio su `float` e `double`

Unico fatto certo:

spazio (`float`) $\leq$ spazio (`double`) $\leq$ spazio(`long double`)

Esempio tipico `float` in 4 byte e `double` in 8 byte

$\Rightarrow$ accuratezza: 6 decimali per `float`

15 decimali per `double`

$\Rightarrow$ valori tra 10^{-38} e 10^{+38} per `float`

tra 10^{-308} e 10^{+308} per `double`

Operazioni built-in per dati di tipo `float`

=	Assegnamento di un valore float a una variabile float
+	Somma (tra float , risultato float)
-	Sottrazione (tra float , risultato float)
*	Moltiplicazione (tra float, risultato float)
/	Divisione (tra float, risultato float)
==	Relazione di uguaglianza
!=	Relazione di diversità
<	Relazione “minore di”
>	Relazione “maggiore di”
<=	Relazione “minore o uguale a”
>=	Relazione “maggiore o uguale a”

Operatoori Aritmetici

Operatoori Relazionali

Operazioni tra `float`

operazioni applicabili a **`float`** (anche a **`double`** e **`long double`**) sono le stesse degli **`int`**, ma divisione '/' dà risultato reale

NB: il simbolo dell'operazione è identico a divisione intera

standard library **`math.h`** fornisce funzioni predefinite (**`sqrt`**, **`pow`**, **`exp`**, **`sin`**, **`cos`**, **`tan`**...) applicate a valori double

$\Rightarrow$ si usa **`double`** anche quando basta **`float`**

Il tipo `float` e `double` : le approssimazioni

Nella rappresentazione di un numero decimale possono esserci **errori di approssimazione**

- Non sempre: `(x / y) * y == x`
- Per verificare l'uguaglianza tra `float` o `double`, definire dei bounds : Invece di
  - `if (x == y) ...` è meglio
  - `if (x <= y + .000001 && x >= y - .000001)`

Buona parte delle operazioni algebriche eseguibili tra `float` (es. l'elevamento a potenza, il logaritmo, la radice, il valore assoluto…) sono nella libreria math che occorre includere con

- `#include<math.h>`

Il tipo `char`

La codifica ASCII prevede di allocare sempre 1 Byte per rappresentare caratteri

- alfanumerici
- di controllo (istruzioni legate alla visualizzazione),

C'è una corrispondenza tra i **`char`** e 256 numeri interi

Le operazioni sui **`char`** sono le stesse definite su **`int`**

- hanno senso gli operatori aritmetici (+ - * / %)
- hanno senso gli operatori di relazione (== , > , < ,.. etc)

`unsigned char` coprono l'intervallo $[0, 255]$.

`signed char` coprono l'intervallo $[-128, 127]$.

N.B. non esistono tipi semplici più «piccoli» del **`char`**

Il tipo `char`

I valori costanti di tipo **`char`** nel codice sorgente si delimitano tra apici singoli `' '`

Gli apici doppi `" "` vengono utilizzati per delimitare stringhe, i.e. sequenze di caratteri (non hanno un loro tipo built in)

- le abbiamo già viste in **`printf`** e **`scanf`**

La codifica ASCII (parziale)

DEC	CAR	DEC	CAR	DEC	CAR	DEC	CAR	DEC	CAR
48	0	65	A	75	K	97	a	107	k
49	1	66	B	76	L	98	b	108	l
50	2	67	C	77	M	99	c	109	m
51	3	68	D	78	N	100	d	110	n
52	4	69	E	79	O	101	e	111	o
53	5	70	F	80	P	102	f	112	p
54	6	71	G	81	Q	103	g	113	q
55	7	72	H	82	R	104	h	114	r
56	8	73	I	83	S	105	i	115	s
57	9	74	J	84	T	106	j	116	t
				85	U			117	u
				86	V			118	v
				87	W			119	w
				88	X			120	x
				89	Y			121	y
				90	Z			122	z

Il tipo `char` esempi

```
char a,b;
b = 'q';

a = "q";

a = '\n';

b = 'ps';
a = 75;
a = 'c' + 1;
a = 'c' - 1;
  a = 20;
  a *= 4;
  a -= 10;
a = `1';
```

Il tipo `char` esempi

```
char a,b;
b = 'q'; /* Le costanti di tipo carattere si
      indicano con ' */
a = "q"; /* NO: "q" è una stringa, anche se di
      un solo carattere */
a = '\n'; /* OK: \n è un carattere a tutti gli
      effetti anche sono due elementi*/
b = 'ps'; /* NO:'ps' non è un carattere valido*/
a = 75; /*associa ad a il carattere‘K’ cfr ASCII
a = 'c' + 1; /* a diventa 'd' */
a = 'c' - 1; /* a diventa  'b' */
  a = 20;
  a *= 4; /* sta per a = a * 4, quindi a = 80 (‘P’)*/
  a -= 10; //a <--70 che corrisponde al carattere 'F'
a = `1'; /*a è il carattere 1, corrispondente a 49
```

Riepilogando sui tipi built in

I tipi **integal** sono discreti, rappresentano valori **numerabili**

- sono **`char`** ed **`int`** con tutti i qualificatori e quantificatori (**`signed/unsigned char, short/long int, signed/unsigned int`**)

I tipi **floating** approssimano insiemi **densi**

- sono **`float`** e **`double,`** eventualmente con il quantificatori **`long`**

I tipi **floating** e **integral** assieme compongono i tipi **arithmetic**

Esercizi sui vettori

Esercizio

Scrivere un programma che acquisisce un vettore e quindi

- Stampa l'istogramma orizzontale
- Stampa l'istogramma in verticale

```
"C:\Users\Giacomo\Dropbox (DEIB)\Didattica\2021_Informatica_A_Boracchi\Lez7\plotHist.exe"

v = [6, 7, 3, 6, 1, ]

 istogramma orizzontale

   6 |******
   7 |*******
   3 |***
   6 |******
   1 |*

 istogramma verticale

       *
  *    *         *
  *    *         *
  *    *         *
  *    *    *    *
  *    *    *    *
  *    *    *    *    *
  *    *    *    *    *
-----------------------------
  6    7    3    6    1

Process returned 0 (0x0)   execution time : 8.231 s
Press any key to continue.
```

Problema 1: Array

Scrivere un programma che acquisisca una sequenza di 10 numeri interi ed un indice X tra 0 e 9. Il programma dovrà stampare la somma dei numeri in posizioni minori di X e il prodotto dei numeri in posizioni successive a X.

Problema 2 Array

Scrivere un programma che richiede l'inserimento di voti ottenuti in un certo numero di corsi. Il programma chiede qual è il numero di voti da inserire. Per ciascuno, l'utente deve inserire:

- Il numero di crediti
- Il voto ottenuto (in trentesimi)

Il programma infine stampa la media pesata dei corsi e il numero totale dei crediti. Si ricorda che la media pesata, nel caso di due voti, si calcola come:

$$media = (voto_1 * crediti_1 + voto_2 * crediti_2)/(crediti_1 + crediti_2).$$

Problema 3: Array

Scrivere un programma che acquisisca due sequenze A e B di 5 numeri interi ciascuna, e stampi a video la sequenza “interlacciata” tra A e l'inversa di B, definita come segue:

- il primo elemento di A,
- l'ultimo elemento di B,
- il secondo elemento di A,
- il penultimo di B etc.

Esempio di esecuzione:

A: 1 2 3 4 5

B: 6 7 8 9 10

Uscita: 1 10 2 9 3 8 4 7 5 6