

Note su Programmazione in C



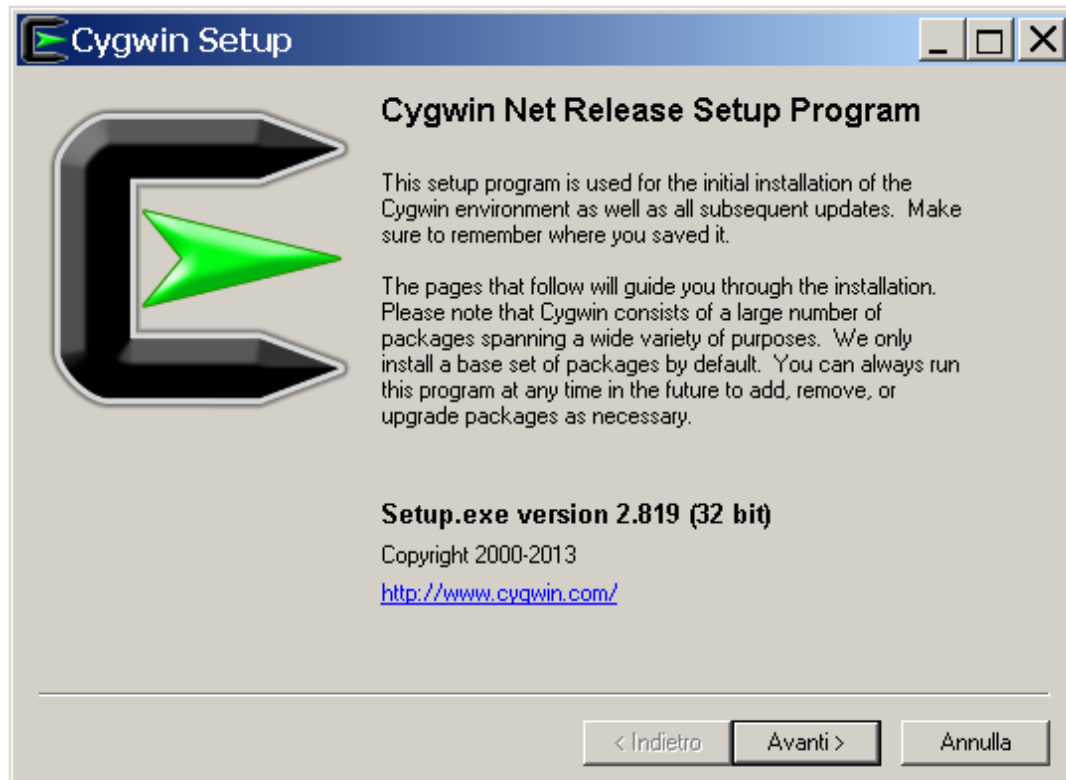
Prof. Ivan Lanese

Elementi di programmazione in C utili per il progetto

- Implementeremo l'assemblatore usando il linguaggio di programmazione C
 - Vedremo quindi insieme alcuni elementi di programmazione in C che bisogna conoscere (ad esempio, la gestione dei file)
- Nelle esercitazioni non useremo un IDE ma useremo un generico editor di testi, compilazione da linea di comando o con Makefile
- Per replicare le esercitazioni esattamente come vengono fatte in aula, per chi usa Windows si rende necessario installare un emulatore di terminale come ad esempio Cygwin
 - È semplice, seguite le istruzioni nelle successive slides

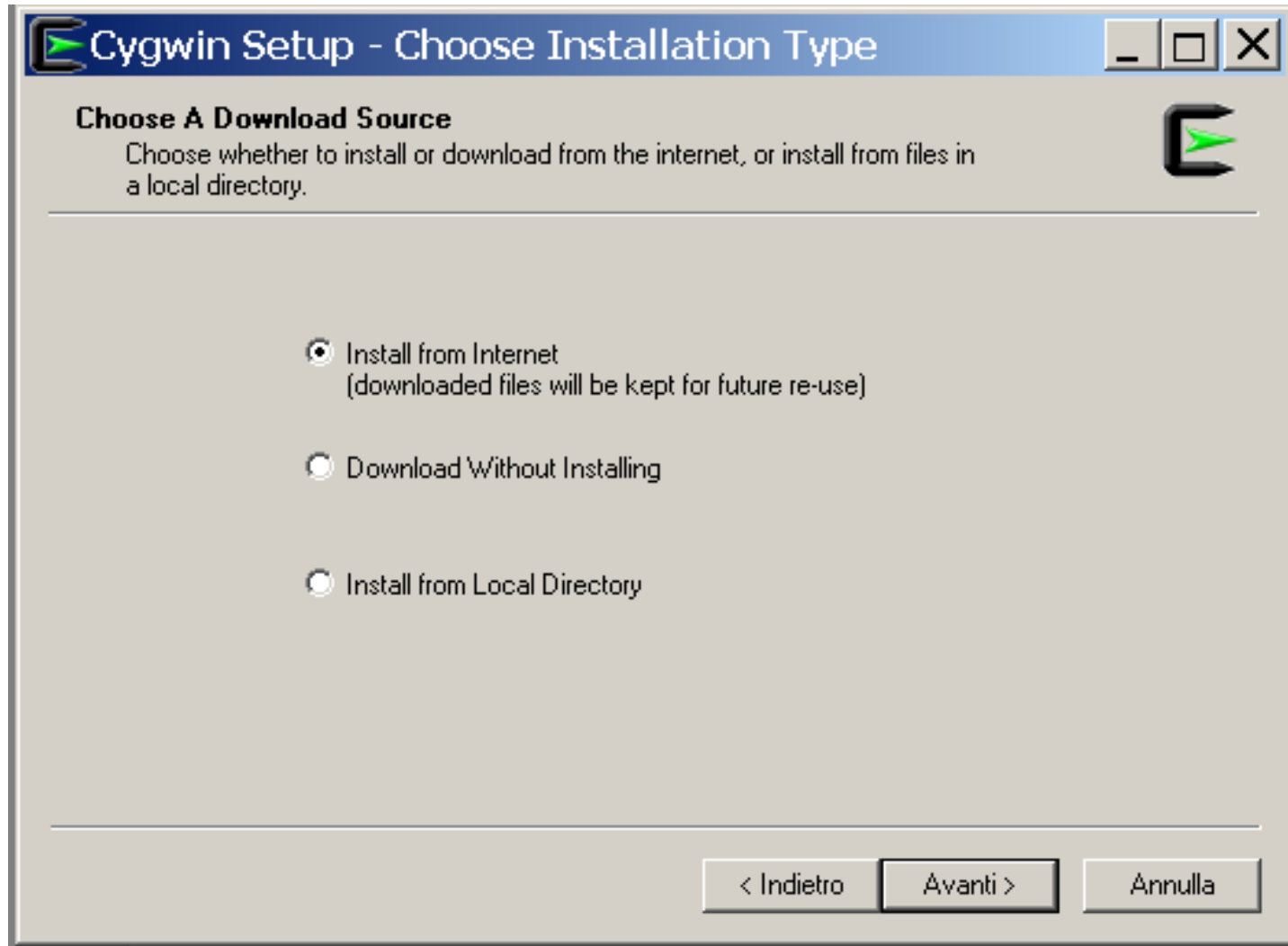
Installare Cygwin (per sistemi Windows)

- Sul sito web www.cygwin.com
- Consultare <http://cygwin.com/install.html>
- Scaricare ed eseguire `setup-x86.exe` (per sistemi a 32 bit) oppure `setup-x86_64.exe` (per sistemi a 64 bit).



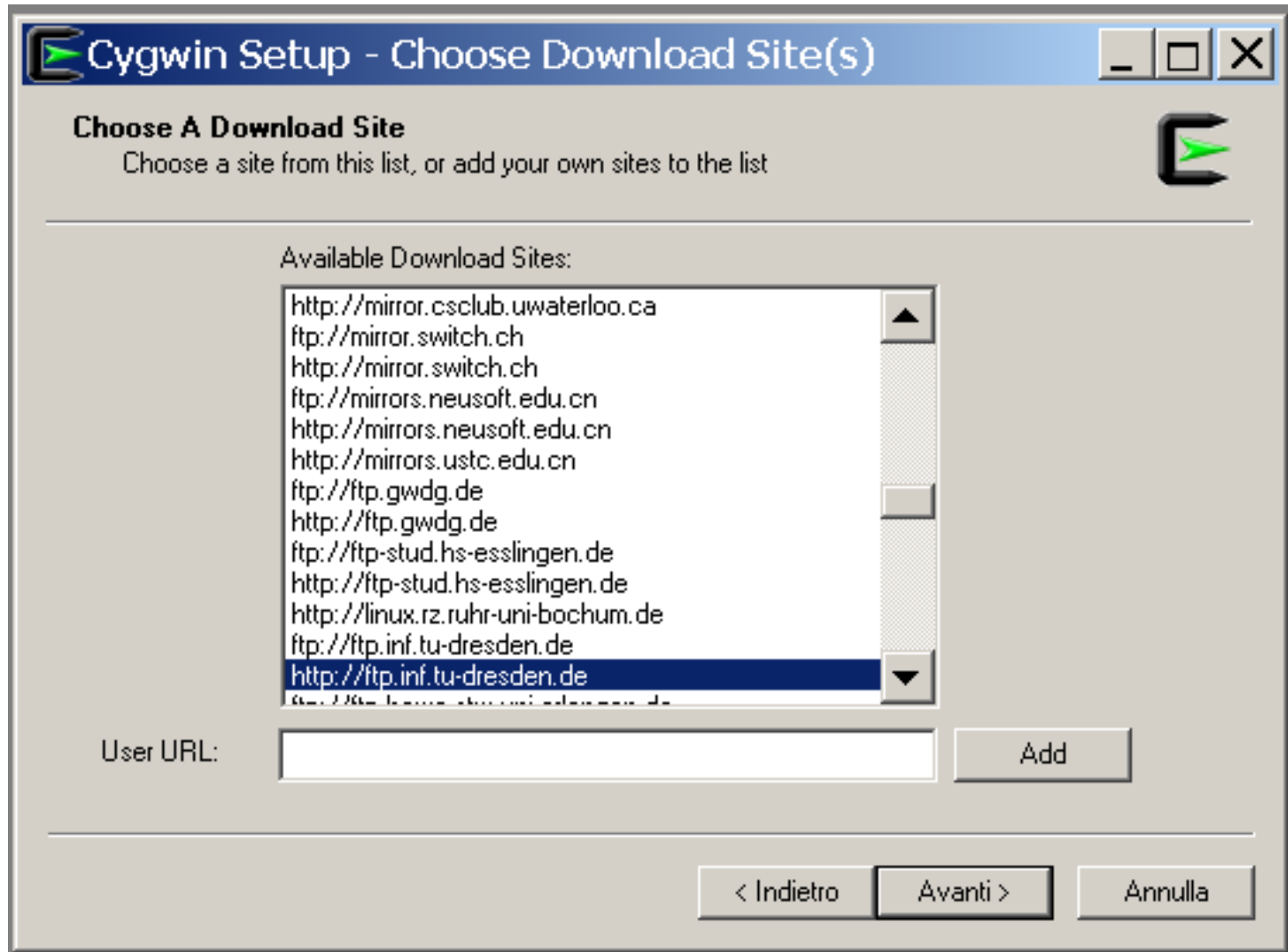
Installare Cygwin (per sistemi Windows)

- Selezionare l'installazione da Internet



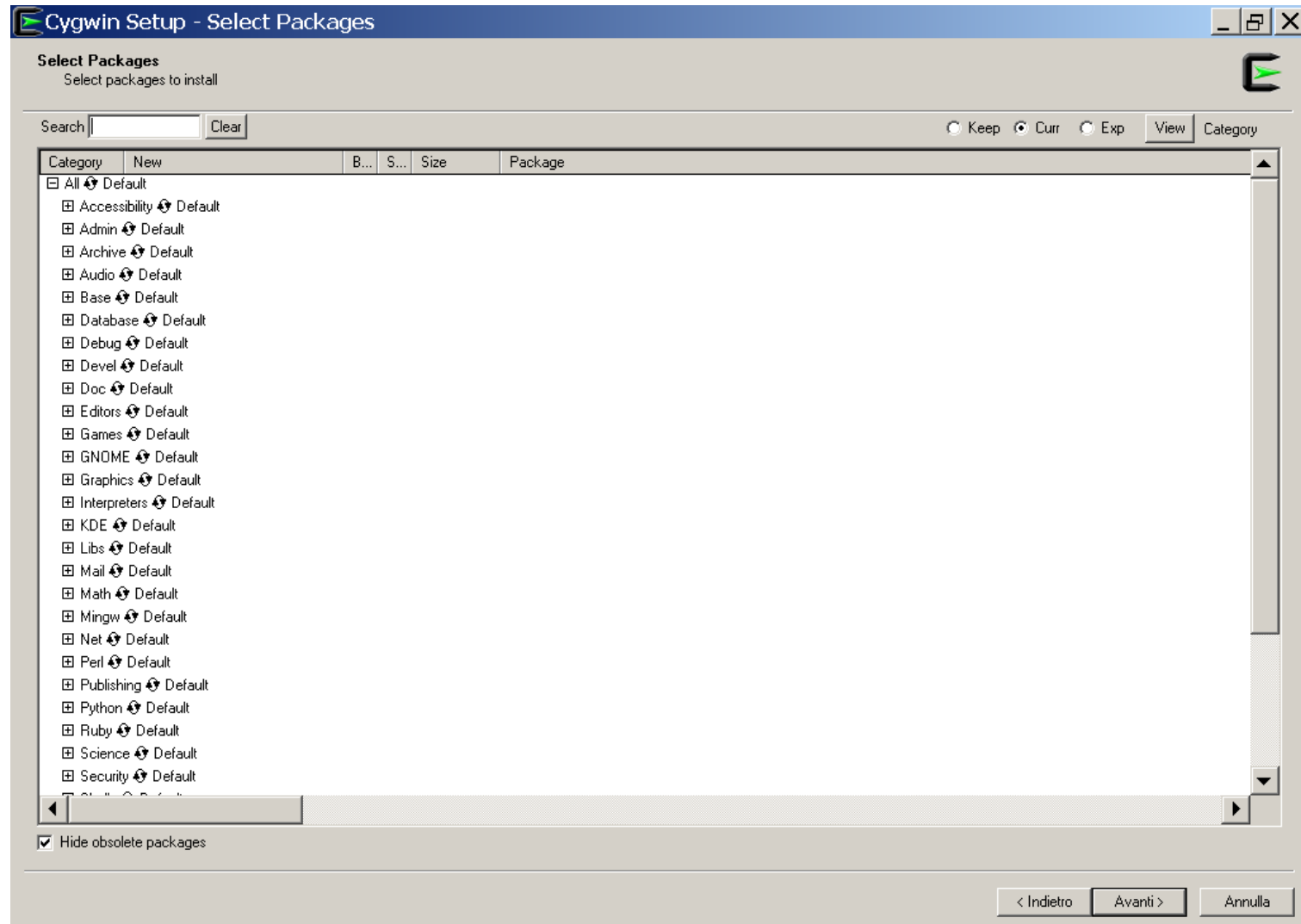
Installare Cygwin (per sistemi Windows)

- Selezionare un mirror da cui scaricare i files



Installare Cygwin (per sistemi Windows)

- Selezionare i pacchetti da installare dalle varie categorie



Installare Cygwin (per sistemi Windows)

- Configurazione minimale necessaria:
 - Nella categoria **devel** selezionare:
 - gcc (compilatore C)
 - g++ (per noi facoltativo) se volete programmare anche in C++
 - make (la utility che useremo per eseguire la compilazione ed il linking)

Usare Cygwin – minimo indispensabile (per sistemi Windows)

- Potete muovervi nella struttura delle directory con `cd`
 - `cd /` va nella directory "root"
 - `cd ..` va nella directory padre
 - `cd pippo` va nella sottodirectory pippo
 - `cd pippo/pluto` va nella sottodirectory pluto di pippo
- Trovate i vostri dati nella directory `/cygdrive/x/` dove `x` è il drive
- Potete richiamare notepad da riga di comando scrivendo `notepad`

Processo di compilazione in C

- Il compilatore C lavora nelle seguenti fasi:
 - Preprocessore:
 - Trasforma i file sorgenti in base alle "preprocessor directives" che contengono
 - Compilatore:
 - Traduce in assembly il programma in C
 - Di default si genera direttamente il "codice oggetto", cioè un file già in linguaggio macchina (con riferimenti ancora da instanziare, vedi ad esempio invocazione di funzioni di libreria)
 - Si può richiedere la generazione di una versione in assembly (human readable)
 - Linking:
 - Collega vari "object file" e librerie (statiche) generando così un "executable file"

Preprocessore in C

- Principali "preprocessor directives":
 - `#include`
 - Include un file esterno
 - `#define MACRO, #define MACRO VALUE`
 - Definisce una macro
 - `#ifdef MACRO .. #endif`
 - Se una macro è definita allora include il testo fra le due direttive, altrimenti lo esclude
 - `#ifndef MACRO .. #endif`
 - Se una macro non è definita allora include il testo fra le due direttive, altrimenti lo esclude
- Invocando il compilatore con `-E` si interrompe il compilatore dopo il preprocessore e si ottiene il risultato in standard output
 - Con `-S` si genera un file `.s` con assembly human readable anziché il file eseguibile

Programmare usando file separati

- E' possibile separare in file distinti un programma C per avere una migliore organizzazione del proprio codice
- Una volta scritto uno di questi file, se mancano gli altri, non è possibile eseguire il linking
 - Per fermarsi alla creazione dell'"object file" si aggiunge -c in fase di compilazione
 - Esempio: `gcc -c pippo.c`
compila il solo file `pippo.c` e genera il relativo "object file" `pippo.o` (attenzione, quest'ultimo non è ancora eseguibile!)
- Come permettere l'interazione fra i diversi file
 - Solitamente, ogni file `.c` viene affiancato da un relativo file `.h` che contiene le dichiarazioni (il prototipo) delle funzioni definite nel file `.c`
 - Se si vogliono usare tali funzioni in un altro file, sarà sufficiente includere il file `.h` contenente le dichiarazioni

Esempio

```
// main.c
#include <stdio.h>
#include "somma.h"

int main() {
    printf("somma:%d\n", somma(5,3));
}
```

```
// somma.h
int somma(int i,int j);
```

```
// somma.c
#include "somma.h"
int somma(int i,int j){
    return i+j;
}
```

Compilazione dell'esempio

- Come procedere nella compilazione dei file nell'esempio
 - Compilare (non linkare) i file .c:
 - `gcc -c main.c`
 - `gcc -c somma.c`

Così si generano due object file `main.o` e `somma.o`
 - Effettuare il linking:
 - `gcc -o main main.o somma.o`

Così si genera un eseguibile con il nome "main"
 - L'eseguibile può essere messo in esecuzione da terminale usando `./main`
 - `./` serve per indicare alla shell dove trovare l'eseguibile (nella current directory), visto che la current directory non è solitamente tra quelle dove un eseguibile viene cercato

Makefile

- Un modo per velocizzare la compilazione in casi di programma scritto in più file è usare la utility make
 - Richiede la preparazione di un file chiamato "Makefile"

```
main: main.o somma.o
    gcc -o main main.o somma.o

main.o: main.c somma.h
    gcc -c main.c

somma.o: somma.c somma.h
    gcc -c somma.c

clean:
    rm *.o main
```

Makefile

Target iniziale:
considerato
eseguendo “make”

Dipendenze

```
main: main.o somma.o
    gcc -o main main.o somma.o

main.o: main.c somma.h
    gcc -c main.c

somma.o: somma.c somma.h
    gcc -c somma.c

clean:
```

Comando che
viene eseguito
se sono state
modificate le
dipendenze

Target tipico per ripulire la directory:
considerato eseguendo “make
clean”

Considerazioni su make

- Ricompila solo le parti che sono state modificate
 - Scorre la catena delle dipendenze
 - Se trova un target che dipende da file modificati dopo l'ultima modifica del target allora esegue il comando associato
- Con "make clean" si eliminano tutti i file generabili da altri file
 - Nel nostro caso di solito gli object file e l'eseguibile
 - Per evitare errori nel tentativo di cancellare file non presenti, solitamente si usa "rm -f"
- I tab come mostrati nell'esempio sono necessari

Tipi primitivi in C

- Il linguaggio C mette a disposizione vari tipi primitivi che si differenziano per spazio occupato in memoria e valori che si possono rappresentare

Tipo	Size (byte)	Valore minimo	Valore Massimo
signed char	1	-128	+127
unsigned char	1	0	255
short int	2	-32768	+32767
unsigned short int	2	0	65535
long int	4	-2^{31}	$+2^{31}-1$
unsigned long int	4	0	$+2^{32}-1$
float	4	$-3.2 \times 10^{\pm 38}$	$+3.2 \times 10^{\pm 38}$
double	8	$-1.7 \times 10^{\pm 308}$	$+1.7 \times 10^{\pm 308}$
void	0	non definito	non definito

Stringhe in C

- In C la stringa vera e propria non esiste
- Si rappresenta una stringa con un vettore di $n+1$ char, in cui al carattere in ultima posizione è assegnato il valore 0, ovvero il carattere nullo '\0'. Questo carattere particolare rappresenta il delimitatore finale della stringa, ovvero l'indicazione che la stringa è finita.

's'	't'	'r'	'i'	'n'	'g'	'\0'
0	1	2	3	4	5	6

Nell'esempio, la lunghezza della stringa è 6, anche se la sua occupazione in memoria è superiore

- Una costante di tipo stringa, ovvero una costante di tipo vettore di caratteri viene scritta come una **sequenza di caratteri racchiusa tra 2 doppi apici, che non fanno parte della stringa.**
 - I singoli apici si usano per delimitare i singoli caratteri (vedi immagine sopra)

■ `int printf(char *format, arg1, arg2, arg3, ...);`

<code>%d</code>	stampa un intero
<code>%10d</code>	stampa un intero e un minimo di 10 caratteri (a dx)
<code>%-10d</code>	stampa un intero e un minimo di 10 caratteri (a sx)
<code>%f</code>	stampa un float
<code>%lf</code>	stampa un double
<code>%10f</code>	stampa un float e un minimo di 10 caratteri (a dx)
<code>%-10f</code>	stampa un float e un minimo di 10 caratteri (a sx)
<code>%10.5f</code>	stampa un float e un minimo di 10 caratteri (5 decimali)
<code>%s</code>	stampa tutti i caratteri di una stringa
<code>%10s</code>	stampa almeno 10 caratteri di una stringa (a dx)
<code>%15.10s</code>	stampa almeno 15 caratteri, prendendone al massimo 10 dalla stringa, con la stringa a destra
<code>%c</code>	stampa un singolo carattere

Input formattato

■ `int scanf(char *format, *arg1, *arg2, *arg3, ...);`

`%c` viene letto un singolo carattere e copiato singolarmente nel primo byte indicato dal puntatore

`%10c` vengono letti 10 caratteri e copiati nei primi 10 byte indicati dal puntatore

`%s` viene letta la stringa e scritta interamente a partire dal puntatore passato, e in fondo viene aggiunto un carattere `'\0'`

`%d` viene letto un intero

`%f` viene letto un float

`%lf` viene letto un double

Modifiche di tipo

- La conversione di tipo **type-casting** in qualche caso viene effettuata implicitamente dal compilatore C, ma può anche essere forzata dal programmatore mediante un operatore unario detto **cast**, che opera in questa maniera:

(nome_nuovo_tipo) espressione

dove "nome_nuovo_tipo" è un tipo di dato primitivo o definito dall'utente, ed "espressione" può essere una variabile, una costante o un'espressione complessa.

- Vediamo subito un esempio di cosa succede:

```
int    i=5,    j=2;
double f;
f = i / j;
/* f ora vale 2 */
```

```
int    i=5,    j=2;
double f;
f = (double)i / (double) j;
/* f ora vale 2.5 */
```

Operatore sizeof()

- L'operatore sizeof prende in input o una variabile o un identificatore di tipo, e restituisce la dimensione in byte del dato.

- Esempio:

```
printf("dimensione del float: %d \n", sizeof(float) );  
/* stampa 4 */
```

- Il dato può anche essere di un tipo definito dall'utente, non solo di un tipo di dato primitivo.

Strutture in C

- Le strutture in C sono record composti da più campi.
 - Ad esempio vediamo come è definito un tipo di dato struct chiamato **luogo** che vuole rappresentare un punto geografico individuato da tre coordinate intere x, y, z ed un nome rappresentato da una stringa di 30 caratteri:

```
struct luogo {  
    char nome[30];  
    int x;  
    int y;  
    int z;  
};
```

- Una volta definito il tipo di dato, possiamo dichiarare una variabile di quel tipo, premettendo però la keyword **struct**:

```
struct luogo monte_bianco;
```

Uso della direttiva typedef

- E' possibile associare un nome identificatore per il nuovo tipo:

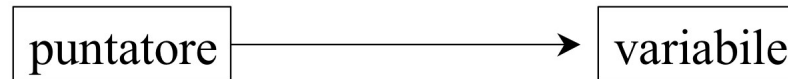
```
typedef struct luogo {  
    char nome[30];  
    int x;  
    int y;  
    int z;  
} LUOGO;
```

- Una volta definito un identificatore lo possiamo usare nelle dichiarazioni di variabili di quel tipo:

```
LUOGO monte_bianco;
```


Puntatori in C

- Un puntatore è un tipo di dato, una variabile di tipo puntatore contiene l'indirizzo in memoria di un'altra variabile, cioè un numero che indica in quale cella di memoria comincia la variabile puntata.



- Si possono avere puntatori a qualsiasi tipo di variabile.
- La dichiarazione di un puntatore include il tipo dell'oggetto a cui il puntatore punta; questo serve al compilatore che deve accedere alla locazione di memoria puntata dal puntatore per sapere cosa troverà, in particolare per sapere le dimensioni della variabile puntata dal puntatore.
- Per dichiarare un puntatore *p* ad una variabile di tipo *tipo*, l'istruzione è:


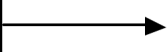

*tipo *p ;*


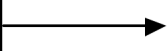
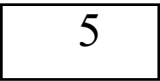
Operatori & e *

- L'operatore & fornisce l'indirizzo di una variabile, perciò
`p = &c` scrive nella variabile p l'indirizzo della variabile c, ovvero:
*tipo c, *p;* dichiaro una var c di tipo *tipo* ed un puntatore p a *tipo*
`p = &c ;` assegno a p l'indirizzo di c
- L'operatore * viene detto operatore di indirezione o deriferimento.
Quando una variabile di tipo puntatore è preceduta da *, si accede al dato puntato, ovvero: `*p` indica la variabile puntata dal puntatore p.

`int c, *p;` dichiaro una var c di tipo int ed un puntatore p a int

p   c
`p = &c ;` assegno a p l'indirizzo di c

p    c
`c = 5;` assegno a c il valore 5

p    c
`printf("%d\n", *p);` stampo il valore puntato dal puntatore p
viene stampato 5

Funzioni malloc e free

- **malloc()** alloca la memoria (chiede al s.o. di riservare un'area di memoria) e restituisce un puntatore a void che punta all'inizio di quest'area di memoria allocata.

Se non è disponibile sufficiente memoria restituisce NULL, ad indicare che l'allocazione non è stata effettuata. E' possibile testare il risultato della malloc() in questo modo:

```
int *p;

p = (int*) malloc( 100*sizeof(int) ); /* chiede di allocare 100 interi */
if( (p==NULL) ) {
    printf("errore: allocazione impossibile");
    exit(1);
} else {
    .... uso la memoria
    free(p); /* rilascia la memoria allocata */
}
```

Esempio: attenzione ai puntatori in strutture

```
typedef struct {  
    char *cognome;  
    int    eta;  
} persona;
```

```
persona p1, p2;
```

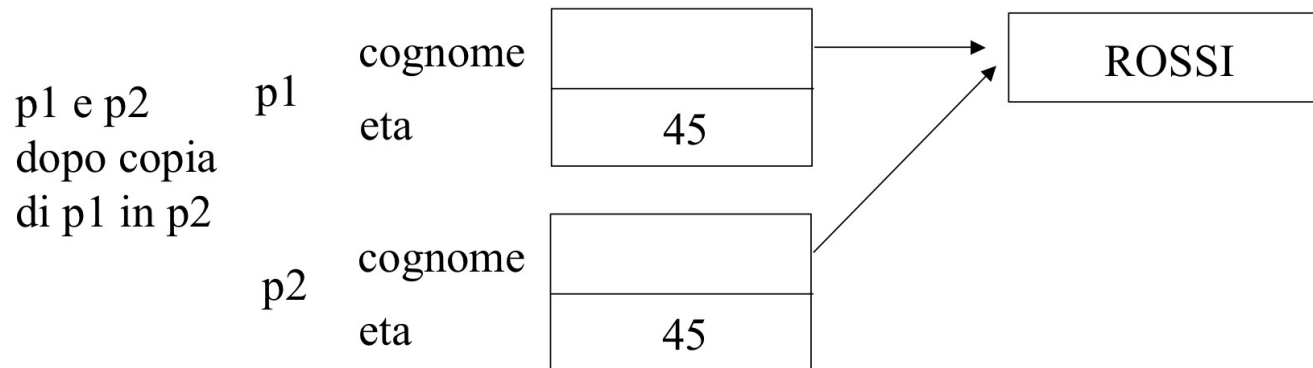
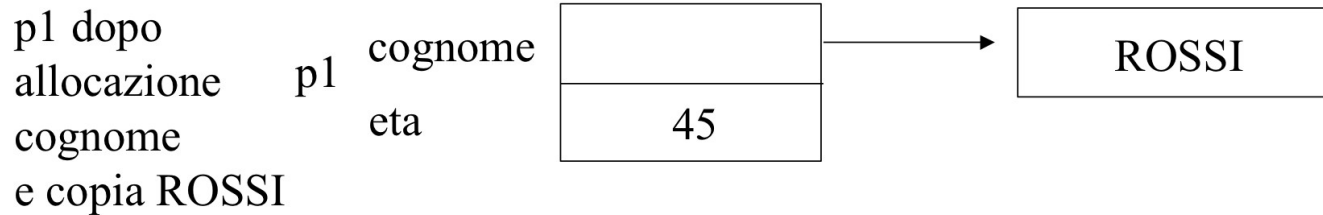
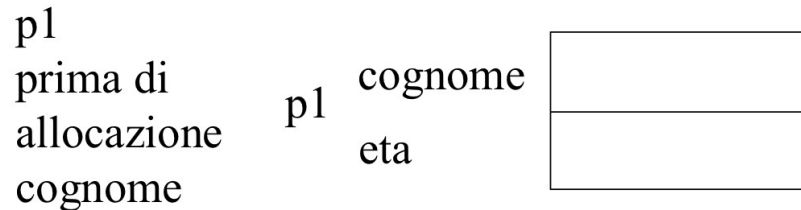
```
/* alloco spazio per il cognome di p1 */  
p1.cognome = (char*) malloc( 50 * sizeof(char));
```

```
/* copio ROSSI nel cognome di p1 */  
strcpy( p1.cognome , "ROSSI" );  
p1.eta = 45;
```

```
/* copio la struttura p1 nella struttura p2 */  
p2 = p1;
```

Esempio: attenzione ai puntatori in strutture (cont.)

risultato: **disastro**, le due variabili p1 e p2 hanno una parte in comune!!



Esempio: attenzione ai puntatori in strutture (cont.)

- La situazione sarebbe stata corretta se io avessi definito la struct persona con un vettore già allocato invece che con un puntatore a char

```
typedef struct {  
    char cognome[50];  
    int  eta;  
} persona;
```

```
persona p1, p2;
```

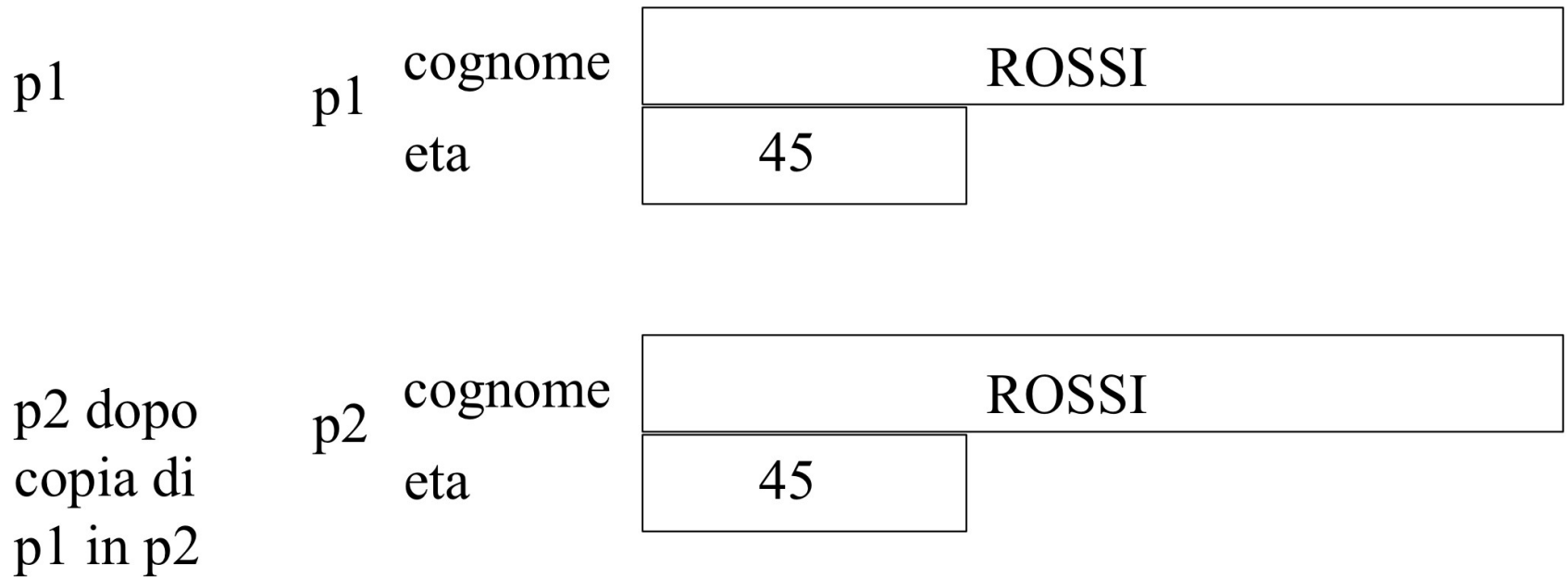
```
/* copio ROSSI nel cognome di p1 */  
strcpy( p1.cognome , "ROSSI" );
```

```
p1.eta = 45;
```

```
/* copio la struttura p1 nella struttura p2 */  
p2 = p1;
```

Esempio: attenzione ai puntatori in strutture (cont.)

risultato: OK !



Puntatori a strutture

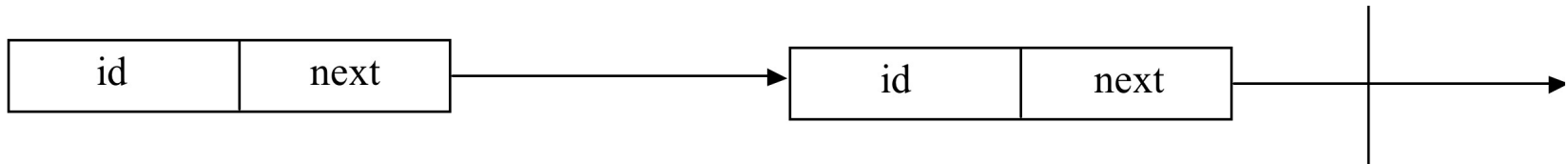
- E' possibile utilizzare puntatori che puntano a dati di tipo strutturato come le struct. La definizione di questo tipo di puntatore ricalca esattamente la definizione generale: **definire il tipo di dato, e poi definire il puntatore a quel tipo di dato.**

```
struct PUNTO {char nome[50]; int x; int y; int z; }; /* definiz. tipo */
struct PUNTO *ptrpunto; /* dichiaraz. puntatore a var. strutturata */
struct PUNTO punto; /* dichiaraz. variabile di tipo strutturato */
ptrpunto = &punto; /* assegno l'indirizzo della var. punto */
```

- In C esiste l'**operatore** **->** che permette l'accesso ad un campo della struttura puntata.
 - Nell'esempio,
ptrpunto -> x = 102;
accede in scrittura al campo **x** della struttura di tipo **PUNTO** puntata da **ptrpunto**.
 - Equivale all'istruzione: **(*ptrpunto).x = 102;**

Puntatori in strutture

- Vogliamo definire una struttura dati che serva come nodo di una lista semplice, una struttura che contenga cioè un intero (il dato della lista) ed un puntatore all'elemento successivo della lista.



```
struct nodolista {  
    int    id;  
    struct nodolista *next;  
};
```

```
struct nodolista nodo;
```

```
typedef struct nodolista {  
    int    id;  
    struct nodolista *next;  
} NODOLISTA;
```

```
NODOLISTA nodo;
```

Un esempio: implementazione di uno Stack

```
//stack.h

typedef struct stackNode {
    int datum;
    struct stackNode *prev;
} stackNode_t;

typedef stackNode_t *Stack;

void init(Stack *S);

void push(Stack *S, int d);

int pop(Stack *S);
```

Un esempio: implementazione di uno Stack (cont.)

```
//stack.c
```

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include "stack.h"
```

```
void init(Stack *S)
{
    *S=NULL;
}
```

```
void push(Stack *S, int d)
{
    Stack s = malloc(
        sizeof(stackNode_t));
    s->prev = *S;
    s->datum = d;
    *S = s;
}
```

```
int pop(Stack *S)
{
    int d;
    Stack old;

    if (*S != NULL) {
        d = (*S)->datum;
        old = *S;
        *S = (*S)->prev;
        free(old);
        return d;
    } else {
        printf("Error pop:
            stack empty\n");
        exit(1);
    }
}
```

Input/output su file in C

- Per poter accedere ad un file in lettura e/o scrittura, bisogna usare un puntatore a file
 - Serve dichiarare delle variabili di tipo FILE*
 - FILE è dichiarato nella libreria stdio.h
- Per associare ad una variabile di tipo FILE* il proprio file si usa la funzione fopen

```
FILE *fin;  
fin=fopen("input.txt","r");  
  
FILE *fout;  
fout=fopen("output.txt","w");
```

- I parametri "r" e "w" indicano apertura in lettura e scrittura rispettivamente ("w" crea il file se non esiste)

Comandi fprintf e fscanf

- Le funzioni tipiche per input/output su stdin e stdout (standard input e standard output) sono printf e scanf
 - i medesimi comandi esistono per scrivere/leggere su file e si chiamano fprintf e fscanf

```
int a,b;  
fscanf(fin,"%d %d",&a,&b) ;  
fprintf(fout,"somma: %d\n",somma(a,b)) ;
```

- la funzione feof restituisce true se si è tentato di leggere oltre la fine del file

```
if (feof(fin))  
{ ... }
```

Chiusura di un file

- Quando un file non viene più usato deve essere chiuso usando `fclose`

```
fclose(fin) ;  
fclose(fout) ;
```

Lettura di una linea da un file di input

- Una funzione utile per leggere un'intera riga (cioè una sequenza di caratteri fino al carattere di fine linea, usualmente `\n`) da un file è `fgets`

```
FILE *fp;  
char str[60];  
fp = fopen("file.txt" , "r");  
fgets(str, 60, fp);
```

- Dati un array di caratteri, una lunghezza massima, ed un puntatore a file aperto in lettura, ritorna la linea corrente (se troppo lunga solo i primi 59 caratteri)
- Come in ogni stringa in C, mette il carattere `'\0'` in fondo alla linea di caratteri
- Restituisce `NULL` se il file è terminato (cioè se il primo carattere letto è EOF)

Parametri da linea di comando

- E' possibile leggere i parametri della riga di comando
- Per farlo è necessario dichiarare il main come:

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

- In questo caso:
 - argc contiene il numero di argomenti, incluso il nome del programma;
 - argv e' un array di stringhe, una per argomento, incluso il nome del programma in argv[0]

Esercizi

- Scrivere un programma che prende da linea di comando il nome di 2 file di testo e copia il primo sul secondo (il secondo viene creato se non esiste)
- Scrivere un programma che prende un parametro da linea di comando e:
 - crea una lista di caratteri contenente i caratteri del parametro
 - elimina dalla lista il carattere in una posizione scelta dall'utente tramite input da tastiera
 - stampa a video la lista risultante
- Dividere il programma dell'esercizio precedente in 4 file sorgente, uno per il main e uno per ogni punto dell'esercizio precedente. Creare un Makefile per la compilazione

Esercizio

- Scrivere un programma che prende da linea di comando il nome di 2 file di testo, legge il primo e scrive sul secondo la lunghezza delle parole del primo, separate da uno spazio e con un a capo ogni volta il primo file va a capo. In questo esercizio una parola è una sequenza massimale di caratteri non spazio/tab/a capo.
- Es. file input: Es. file output:
- Ciao mamma, 4 6
guarda come mi diverto! 6 4 2 8