Programmazione

Docenti: Giovanni Da San Martino
Francesco Gavazzo
Lamberto Ballan
<lamberto.ballan@unipd.it>

Programmazione, A.A. 2023/24
SCQ0093758 - LT Informatica

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Benvenuti a “programmazione 3/3”

- Benvenuti all’ultima parte di programmazione!
- Stessa organizzazione… e la transizione dalle prime due parti dovrebbe essere “smooth”

Benvenuti a “programmazione 3/3”

UNIVERSITÀ DEGLI STUDI DI PADOVA

- Chi sono? (giusto due parole sul mio background…)

Ingegneria Informatica

PhD Ingegneria Informatica, TLC, Multimedia

UNIVERSITÀ DEGLI STUDI DI PADOVA …

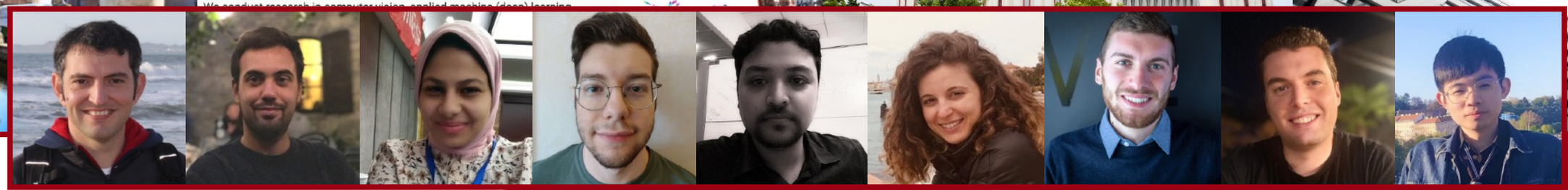
2000 — 2006 — 2011 — 2014 — 2017

UNIVERSITÀ DEGLI STUDI DI FIRENZE

Stanford University

Visual Intelligence & Machine Perception (VIMP) *group*

http://vimp.math.unipd.it



Benvenuti a “programmazione 3/3”

- Approccio algoritmico:

- “Introduzione all’Apprendimento Automatico”

  (corso opzionale al II semestre del 2 o 3 anno)

Studieremo tecniche per apprendere i programmi dai dati

Benvenuti a “programmazione 3/3”

Università degli Studi di Padova

- Argomenti rimanenti e calendario delle lezioni:
  - Elaborazione di File in C
  - Richiami ad allocazione dinamica della memoria e strutture
  - Strutture dati elementari: liste ed alberi

~ 3+ settimane

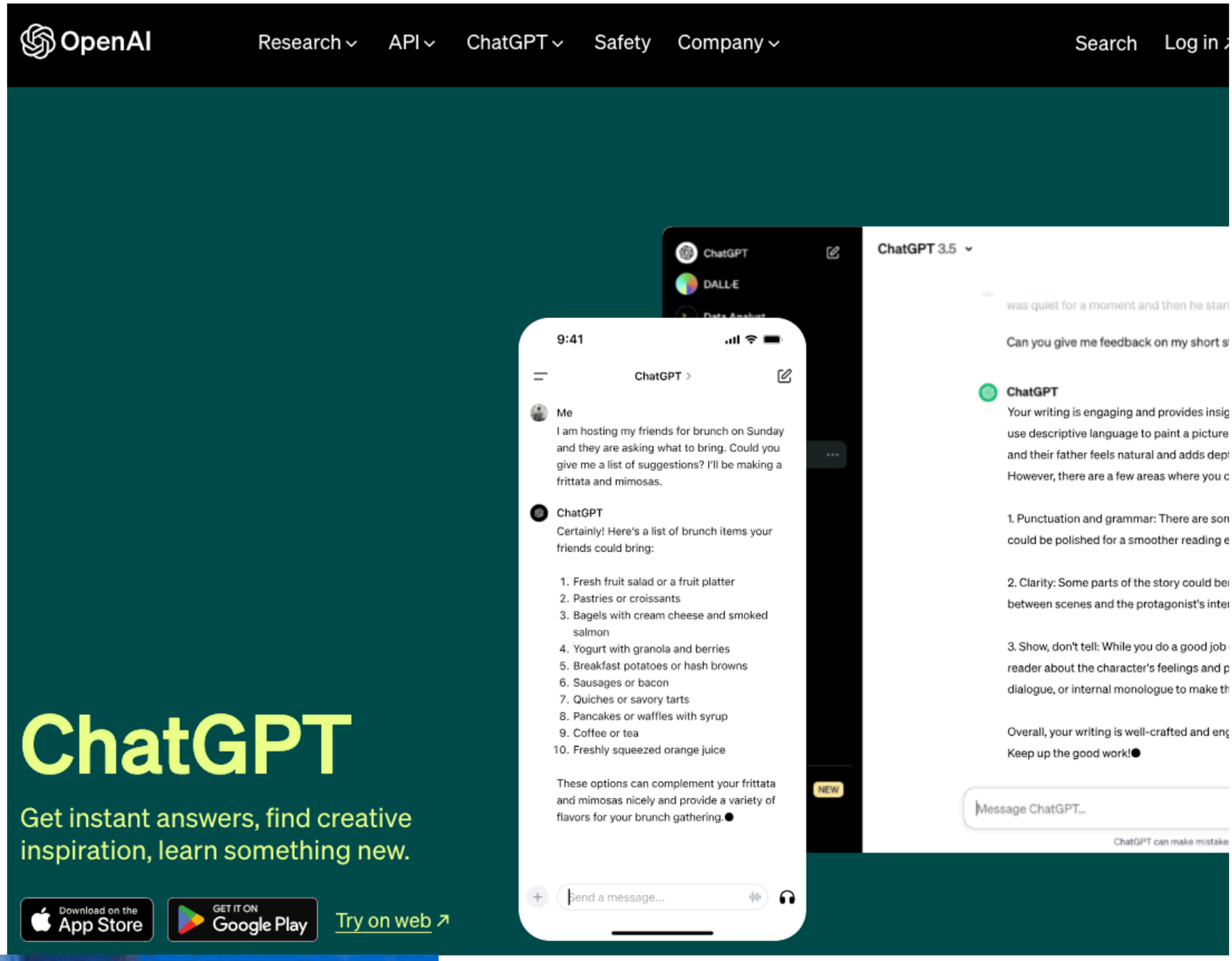
Programmazione ai tempi dei LLMs...

Università degli Studi di Padova

Released on November 30, 2022

In 5 days it reached ~1 million users

~180.5 million users by December 2023



Hello GPT-4o

We're announcing GPT-4o, our new flagship model that can reason across audio, vision, and text in real time.

May 13, 2024

https://chat.openai.com/

Elaborazione di file in C

- Finora abbiamo fatto riferimento alla memorizzazione dei dati (variabili) nella memoria temporanea
  - Quando abbiamo fatto riferimento alla memoria abbiamo sempre considerato il caso della RAM / memoria di lavoro
  - I file vengono utilizzati e sono uno degli strumenti principali per la memorizzazione a lungo termine (storage) dei dati

File e stream in C

- Il C vede ogni file semplicemente come uno *stream* (flusso) sequenziale di byte
  - I file non terminano con un carattere “speciale” ma alcune funzioni restituiscono il valore EOF (end-of-file)
  - Un file può essere “paragonato” ad una lunga stringa; il file viene prima aperto, associato ad uno stream, e infine chiuso
- Tre stream standard vengono aperti automaticamente quando inizia l’esecuzione di un programma:
  - Lo **standard input** (che riceve input dalla tastiera)
  - Lo **standard output** (che stampa output a schermo)
  - Lo **standard error** (che stampa messaggi di errore a schermo)

File e stream in C

- Gli stream forniscono un “canale di comunicazione” tra programmi e file
  - Ad esempio, lo stream `standard input` permette ad un programma di leggere dati dalla tastiera
  - Lo stream `standard output` permette ad un programma di stampare dati sullo schermo

- Possiamo quindi dire che il sistema di I/O del C crea un livello di astrazione fra programma e dispositivo
  - Lo stream è questa astrazione e il dispositivo su cui avvengono le operazioni è chiamato file

Files: classificazione per accesso

- Per accedere ad un dato specifico occorre posizionarsi al byte esatto in cui inizia il dato stesso
- File ad **accesso sequenziale**
  - Hanno blocchi di informazione (*record*) di lunghezza variabile
  - Per accedere al record n, occorre leggere tutti i precedenti n-1
- File ad **accesso diretto** (o casuale)
  - Hanno record di lunghezza fissa
  - La lunghezza L del record viene fissata dal programmatore
  - Per accedere al record n, il byte d'inizio corrisponderà a n*L

Files: classificazione per contenuto

- File di **testo**
  - Sono sequenze di caratteri organizzare in righe
  - Ogni riga è terminata da un carattere speciale di ritorno a capo (in C le funzioni I/O per stringhe e caratteri usano '`\n`')
- File **binari**
  - Sono sequenze "grezze" di byte
- Ci concentreremo innanzitutto sui file di testo ad accesso sequenziale

Struttura di un file in C

- L'apertura di un file restituisce un puntatore a struttura FILE (definita in `stdio.h`)
  - Standard input, output ed error, sono manipolati usando `stdin`, `stdout` e `stderr`
- La libreria standard fornisce varie funzioni di lettura e scrittura su file:
  - `fgetc` e `fputc` (simili a `getchar` e `putchar`)
  - `fgets` e `fputs` (per leggere/scrivere una riga da file)
  - `fscanf` e `fprintf` (analoghi a `scanf` e `printf`)
  - `fread` e `fwrite`

Il tipo FILE

- Le informazioni / caratteristiche di un file sono memorizzate in un'apposita struttura FILE
- Ogni accesso a un file avviene attraverso una variabile di tipo FILE che si riferisce a tale struttura
  - Concretamente occorre quindi dichiarare un puntatore a FILE (ad es. `FILE *fp;`)
- La struttura FILE contiene varie informazioni "di sistema"; tra queste è molto importante il *file position pointer* che memorizza il punto (offset) del prossimo byte da leggere o da scrivere

Apertura di un file (di testo)

- Una volta dichiarata una variabile puntatore a FILE occorre creare una "linea di comunicazione" col file
  - La funzione `fopen` permette di aprire il file associandolo ad un puntatore a struttura FILE
  - La funzione `fopen` ha due argomenti: *i)* una stringa contenente il nome (indirizzo) del file, *ii)* una stringa che indica la modalità di apertura del file stesso
  - Terminata l'elaborazione il file verrà chiuso con `fclose`
  - Vediamo un esempio:

```
FILE *fp;
fp = fopen("studenti.txt","r"); //aperto in lettura
```

Apertura di un file (di testo)

- Se il file non può essere aperto il puntatore sarà uguale a NULL
- Nella pratica è quindi buona regola verificare sempre che l'apertura del file sia andata a termine nel modo corretto

```
FILE *fp;
fp = fopen("studenti.txt","r"); //aperto in lettura
if (fp==NULL){
   fprintf(stderr,"Il file non può essere aperto\n");
   return EXIT_FAILURE;}
else {
   …
   fclose(fp);
}
```

Apertura di un file (di testo)

- Ci sono diverse modalità di apertura del file:

`r`	Apre un file esistente per la lettura.
`w`	Crea un file per la scrittura. Se il file esiste già, *elimina* i contenuti correnti.
`a`	Apre o crea un file per scrivere alla fine del file – cioè, le operazioni di scrittura aggiungono dati al file.
`r+`	Apre un file esistente per l'aggiornamento (lettura e scrittura).
`w+`	Crea un file per l'aggiornamento. Se il file esiste già, *elimina* i contenuti correnti.
`a+`	Append: apre o crea un file per l'aggiornamento; tutta la scrittura è effettuata alla fine del file – cioè, le operazioni di scrittura aggiungono dati al file.
`rb`	Apre un file esistente per la lettura in forma binaria.
`wb`	Crea un file per la scrittura in forma binaria. Se il file esiste già, elimina i contenuti correnti.
`ab`	Append: apre o crea un file per la scrittura alla fine del file in forma binaria.
`rb+`	Apre un file esistente per l'aggiornamento (lettura e scrittura) in forma binaria.
`wb+`	Crea un file per l'aggiornamento in forma binaria. Se il file esiste già, elimina i contenuti correnti.
`ab+`	Append: apre o crea un file per l'aggiornamento in forma binaria; la scrittura è effettuata alla fine del file.

Apertura di un file (di testo)

- Nota: l'apertura di un file in modalità aggiornamento permette sia la lettura che scrittura dello stesso
- Ci sono diverse modalità in aggiornamento:
  - "r+": lettura/scrittura; il file deve esistere
  - "w+": lettura/scrittura; crea il file vuoto
  - "a+": lettura/scrittura al fondo; se il file non esiste lo crea, e aggiunge alla fine se invece già esiste
- La differenza tra "r+" e "w+" è solo nell'apertura

File ad accesso sequenziale e I/O

- Ogni accesso (lettura o scrittura) avviene in modo sequenziale a partire dalla posizione di offset corrente
- Nei file di testo le funzioni di I/O sono analoghe a quelle per le stringhe (salvo indicazione del puntatore a FILE)
  - `fscanf(fp, <stringa_formato>,<espressioni>);`
  - `fprintf(fp, <stringa_formato>,<espressioni>);`
  - `char c = fgetc(fp);`
  - `fputc(<carattere>, fp);`
  - `char s[20]; fgets(s, 20, fp);`
  - `fputs(s, fp);`

Buffer di output

- Le operazioni su file (cioè su disco) sono lente
- I dati diretti a file di output sono perciò accumulati temporaneamente in memoria (in uno spazio di buffer)
  - Quando il buffer è pieno, o si esegue una `fclose`, il buffer viene svuotato ed i dati sono inviati tutti assieme (*flush*)
  - Possiamo anche forzare una operazione di flush del buffer associato al file puntato da `fp` con: `fflush(fp);`

Altre funzioni sui file

- `remove(<filename>);` cancella il file indicato dalla stringa *filename* e restituisce 0 se ha esito positivo
- `rename(<file1>,<file2>);` rinomina *file1* in *file2* e restituisce 0 se ha esito positivo
- `freopen(<file>,<modo>,fp);` associa uno stream pre-esistente ad un altro file
  - Il file con nome `<file>` viene viene associato al puntatore `fp`; se già aperto lo stream pre-esistente di `fp` viene chiuso
  - La stringa `<modo>` indica la modalità di apertura (funziona in modo analogo a `fopen`)
  - Se ha successo restituisce un puntatore a FILE, altrimenti NULL

Altre funzioni sui file

- `rewind(fp);` riporta l'offset del file `fp` all'inizio in modo che la prossima operazione di I/O inizi da 0
- `fseek(fp,<offset>,<posizione>);` sposta l'indicatore di posizione del file puntato da `fp` a <posizione> e si sposta di n bytes = <offset>
  - `int <offset>`: numero di bytes da posizione
  - `int <posizione>`: può assumere tre valori indicati dalle costanti SEEK_END, SEEK_SET, SEEK_CUR
  - Restituisce un intero: 0 se ha successo, un numero diverso da 0 altrimenti

Altre funzioni sui file

- `rewind()` *vs* `fseek()`:
  - La funzione `rewind()` può essere sostituita da `fseek()` (che è più generale); questa istruzione equivale a `rewind()`:
    `fseek(fp,0L,SEEK_SET);`
- `ftell(fp);` restituisce il valore attuale (come long int) dell'indicatore di posizione del file puntato da `fp`

Contatti

- **Ufficio:** Torre Archimede, ufficio 6CD3
- **Ricevimento:** Giovedì 8:30-10:30 *(e-mail per conferma)*

✉ lamberto.ballan@unipd.it
🏠 http://www.lambertoballan.net
🏠 http://vimp.math.unipd.it
@ twitter.com/lambertoballan