ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Nhập môn lập trình - CO1003

Bài tập lớn

ỨNG DỤNG TODO

TP. HỒ CHÍ MINH, THÁNG 09/2023

ĐẶC TẢ BÀI TẬP LỚN

Phiên bản 2.0 (Cập nhật ngày 29/10/2023)

1 Chuẩn đầu ra

Sau khi hoàn thành bài tập lớn này, sinh viên ôn lại và sử dụng thành thục:

- Các cấu trúc rẽ nhánh
- Các cấu trúc lặp
- Mảng 1 chiều và mảng 2 chiều
- Xử lý chuỗi ký tự
- Hàm và lời gọi hàm
- Cấu trúc do người dùng tự định nghĩa (struct)

2 Dẫn nhập

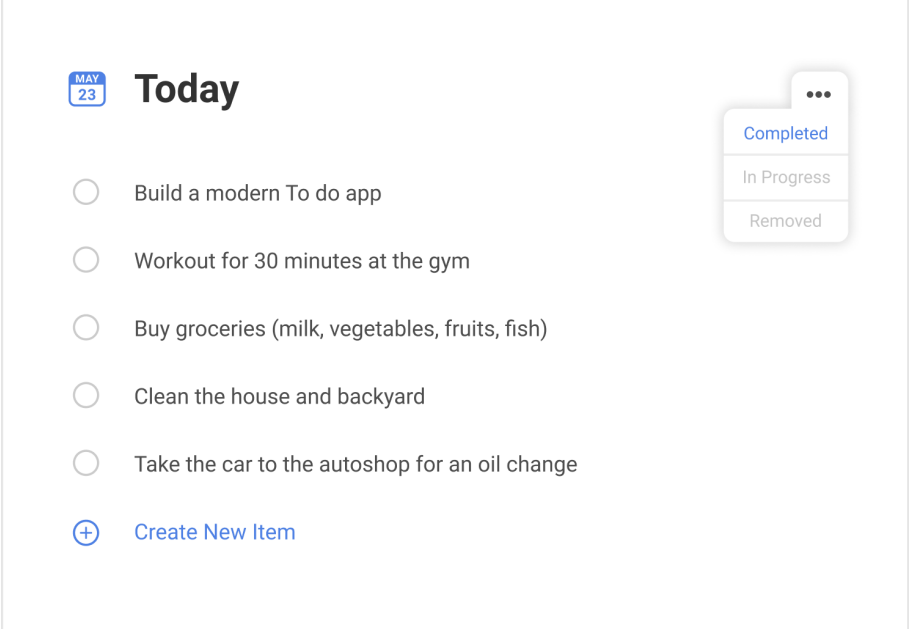
Ứng dụng Todo là một phần mềm hỗ trợ việc quản lý và theo dõi các công việc. Để sử dụng phần mềm, người dùng liệt kê các công việc và thời hạn để thực hiện công việc. Các công việc sẽ được biểu diễn dưới dạng một danh sách và có hiển thị các trạng thái như *Hoàn thành*, *Đang thực hiện*, hoặc *Loại bỏ* (Hình 1). Ứng dụng Todo giúp người dùng quản lý công việc cũng như không quên thực hiện các nhiệm vụ quan trọng. Trong bài tập lớn này, bạn sẽ hiện thực một Ứng dụng Todo đơn giản theo mô tả ở các Mục sau.

3 Yêu cầu ứng dụng

Mục này nêu ra các yêu cầu để xây dựng một Ứng dụng Todo hoàn chỉnh. Để giải quyết các yêu cầu này, sinh viên phải hiện thực các hàm tương ứng theo mô tả yêu cầu. Mỗi yêu cầu có số điểm đi kèm biểu diễn số điểm tối đa đạt được nếu sinh viên hiện thực đúng yêu cầu.

3.1 Kiểu của câu lệnh

Tìm kiểu của lệnh Người dùng tương tác với ứng dụng thông qua các câu lệnh. Các kiểu câu lệnh hợp lệ và định dạng của lệnh là:



Hình 1: Minh họa cho ứng dụng todo

- ***Add***: thêm một công việc mới vào danh sách ứng dụng, định dạng:

```
Add [<title>] [<description>] [<time>]
```

- ***Edit***: thay đổi một thông tin của một công việc đã thêm, có thể có các định dạng:

```
Edit #<num> status:[<status>]
Edit #<num> title:[<title>]
Edit #<num> description:[<description>]
Edit #<num> time:[<time>]
```

- ***Show***: hiển thị các công việc, có thể có các định dạng sau:

```
Show #<num>
Show all
Show head %<quan>
Show tail %<quan>
Show filter title:[<title>]
Show filter description:[<description>]
Show filter status:[<status>]
Show week time:[<date>]
```

- ***Delete***: xóa một công việc đã thêm, định dạng:

```
Delete #<num>
```

- ***Quit***: thoát ứng dụng, định dạng:

```
Quit
```

Chi tiết của mỗi lệnh sẽ được mô tả chi tiết ở các phần sau. *Enum* **CommandType** được sử dụng để biểu diễn kiểu câu lệnh gồm các giá trị ADD, EDIT, SHOW, DELETE, QUIT, INVALID lần lượt tương ứng với các kiểu lệnh Add, Edit, Show, Delete, Quit và lệnh không hợp lệ.

Yêu cầu 1 (0.3 điểm): Hiện thực hàm **getCommandType** trả về kiểu lệnh của một lệnh được truyền vào. Mô tả hàm như sau:

- Tên hàm: **getCommandType**
- Tham số đầu vào:
  - **command (kiểu char *)**: chuỗi chứa một lệnh
- Trả về: giá trị trả về có kiểu là **CommandType** tương ứng với kiểu của lệnh **command** được truyền vào
- Yêu cầu hàm: Gọi ***w*** là từ đầu tiên xuất hiện trong câu lệnh của người dùng. Nếu ***w*** trùng với một trong kiểu câu lệnh được mô tả ở trên thì trả về giá trị *enum* **CommandType** tương ứng. Ngược lại, ***w*** là một từ không hợp lệ thì trả về giá trị **INVALID**.

Ví dụ 1: Các ví dụ về **getCommandType**:

- Lời gọi hàm

```
getCommandType("Show #1")
```

Trả về giá trị:

```
SHOW
```

- Lời gọi hàm

```
getCommandType("Quit")
```

Trả về giá trị:

```
QUIT
```

- Lời gọi hàm

```
getCommandType("Showall")
```

Trả về giá trị:

```
INVALID
```

3.2 Các thành phần của câu lệnh Add

Câu lệnh ***Add*** có định dạng là:

```
Add [<title>] [<description>] [<time>]
```

Trong đó:

- <title> là chuỗi biểu diễn cho tiêu đề của công việc
- <description> là chuỗi biểu diễn mô tả cho công việc
- <time> là chuỗi biểu diễn thời gian cho công việc

Yêu cầu 2 (0.6 điểm): Hiện thực 3 hàm: **getTitleFromAdd**, **getDescriptionFromAdd**, **getTimeFromAdd** lần lượt lấy 3 trường thông tin <title>, <description>, <time> từ câu lệnh ***Add***. Mô tả của 3 hàm như sau:

- Khai báo của 3 hàm:
  - `void getTitleFromAdd(char * command, char * out_title);`
  - `void getDescriptionFromAdd(char * command, char * out_description);`
  - `void getTimeFromAdd(char * command, char * out_time);`
- Đầu vào:
  - **command (kiểu char *)**: chuỗi chứa một câu lệnh ***Add***
- Đầu ra:
  - **out_title**: chứa tiêu đề của câu lệnh ***Add***
  - **out_description**: chứa mô tả của câu lệnh ***Add***
  - **out_time**: chứa thời gian của ***Add***
- Yêu cầu hàm:
  - Hàm **getTitleFromAdd** tìm và gán tiêu đề của câu lệnh add vào **out_title**
  - Hàm **getDescriptionFromAdd** tìm và gán mô tả của câu lệnh add vào **out_description**
  - Hàm **getTimeFromAdd** tìm và gán thời gian của câu lệnh add vào **out_time**

  Mỗi hàm làm đúng trong phần này được **0.2 điểm**.

Ví dụ 2: Cho đoạn code sau:

```
1 char command[] = "Add [Course Intro to Programming] [Room 701-H6]
      [07:00|01/10/2023-12:00|01/10/2023]";
2 char raw_title[200];
3 char raw_description[200];
4 char raw_time[200];
5 getTitleFromAdd(command, raw_title);
6 getDescriptionFromAdd(command, raw_description);
7 getTimeFromAdd(command, raw_time);
```

Sau khi thực hiện dòng 5, **raw_title** có giá trị là:

```
Course Intro to Programming
```

Sau khi thực hiện dòng 6, **raw_description** có giá trị là:

```
Room 701-H6
```

Sau khi thực hiện dòng 7, **raw_time** có giá trị là:

```
07:00|01/10/2023-12:00|01/10/2023
```

3.3 Tiêu đề của câu lệnh Add

Tiêu đề của câu lệnh Add được quy định phải thỏa mãn các điều kiện sau:

- Độ dài tối đa của tiêu đề là 100 ký tự
- Tiêu đề chỉ có thể chứa ký tự là một trong các loại sau:
  - Ký tự chữ cái thường
  - Ký tự chữ cái hoa
  - Ký tự chữ số
  - Các ký tự khác: khoảng trắng, dấu phẩy, dấu chấm, dấu gạch ngang, dấu hai chấm, dấu '|' và '/'
- Tiêu đề không được bắt đầu và không được kết thúc bằng ký tự khoảng trắng

Yêu cầu 3 (0.4 điểm): Hiện thực hàm **checkTitle** với mô tả như sau:

- Khai báo của hàm: **int checkTitle(char * raw_title)**
- Đầu vào:
  - **raw_title (kiểu char *)**: chuỗi chứa tiêu đề cần kiểm tra
- Đầu ra:
  - Nếu tiêu đề là hợp lệ với các điều kiện trên thì trả về giá trị -1;
  - Nếu tiêu đề là không hợp lệ do vi phạm điều kiện về độ dài tối đa thì trả về độ dài hiện tại của tiêu đề
  - Nếu tiêu đề là không hợp lệ do vi phạm các điều kiện còn lại (ngoài điều kiện về độ dài tối đa) thì trả về vị trí của ký tự đầu tiên vi phạm điều kiện.

3.4 Mô tả của câu lệnh Add

Mô tả của câu lệnh ***Add*** phải thỏa các điều kiện giống như điều kiện cho tiêu đề của lệnh ***Add***. Tuy nhiên, điều kiện về độ dài tối đa của mô tả là 200 ký tự.

Yêu cầu 4 (0.4 điểm): Hiện thực hàm **checkDescription** với mô tả như sau:

- Khai báo của hàm: **int checkDescription(char * raw_description)**
- Đầu vào:
  - **raw_description (kiểu char *)**: chuỗi chứa mô tả cần kiểm tra
- Đầu ra:
  - Nếu mô tả là hợp lệ với các điều kiện trên thì trả về giá trị -1;
  - Nếu mô tả là không hợp lệ do vi phạm điều kiện về độ dài tối đa thì trả về độ dài hiện tại của tiêu đề
  - Nếu mô tả là không hợp lệ do vi phạm các điều kiện còn lại (ngoài điều kiện về độ dài tối đa) thì trả về vị trí của ký tự đầu tiên vi phạm điều kiện.

3.5 Thời gian của câu lệnh Add

Thời gian của câu lệnh Add có định dạng là

```
<datetime1>-<datetime2>
```

Trong đó:

- **datetime1** là thời điểm bắt đầu công việc
- **datetime2** là thời điểm kết thúc công việc

datatime1 và datetime2 đều có cùng định dạng là:

```
<hh>:<mm>|<dd>/<mo>/<yyyy>
```

Trong đó:

- **<hh>** biểu diễn giờ, gồm 2 chữ số. Giờ hợp lệ là một số nguyên nằm trong đoạn $[0, 23]$
- **<mm>** biểu diễn phút, gồm 2 chữ số. Phút hợp lệ là một số nguyên nằm trong đoạn $[0, 59]$
- **<dd>** biểu diễn ngày, gồm 2 chữ số. Ngày hợp lệ là một số nguyên có giá trị từ 1 đến ngày tối đa trong tháng
- **<mo>** biểu diễn tháng, gồm 2 chữ số. Tháng hợp lệ là một số nguyên có giá trị từ 1 đến 12. Các tháng 1, 3, 5, 7, 8, 10, 12 có 31 ngày. Các tháng 4, 6, 9, 11 có 30 ngày. Riêng tháng 2 có thể có 28 hoặc 29 ngày. Trong năm nhuận, tháng 2 có 29 ngày
- **<yyyy>** biểu diễn năm, gồm 4 chữ số. Năm hợp lệ là một số nguyên dương.

Thời gian của câu lệnh Add là hợp lệ nếu thỏa mãn các điều kiện sau:

1. **datetime1** là hợp lệ
2. **datetime2** là hợp lệ
3. **datetime2** không được sớm hơn **datetime1**

Yêu cầu 5 (0.5 điểm): Hiện thực hàm **checkTime** với mô tả sau:

- Khai báo của hàm: **int checkTime(char * raw_time)**
- Đầu vào:
  - **raw_time (kiểu char*)**: chuỗi chứa thời gian cần kiểm tra
- Đầu ra:
  - Nếu thời gian là hợp lệ với các điều kiện trên thì trả về giá trị -1;
  - Nếu thời gian là không hợp lệ do vi phạm điều kiện **3.** thì trả về giá trị 0
  - Nếu mô tả là không hợp lệ do vi phạm các điều kiện còn lại thì trả về giá trị như Bảng 1. Lưu ý: nếu **datetime1** và **datetime2** đều không hợp lệ thì trả về giá trị tương ứng với **datetime1**.

Bảng 1: Bảng các giá trị trả về khi thời gian không hợp lệ

Thành phần không hợp lệ	Thời điểm	Giá trị trả về
<hh>	datetime1	11<hh>
<hh>	datetime2	12<hh>
<mm>	datetime1	21<mm>
<mm>	datetime2	22<mm>
<dd>	datetime1	31<dd>
<dd>	datetime2	32<dd>
<mo>	datetime1	41<mo>
<mo>	datetime2	42<mo>
<yyyy>	datetime1	51<yyyy>
<yyyy>	datetime2	52<yyyy>

3.6 Các thành phần của câu lệnh Edit

Câu lệnh ***Edit*** được sử dụng để thay đổi các thông tin liên quan đến một công việc. Câu lệnh Edit có định dạng thuộc một trong các dòng sau:

```
Edit #<num> title:[<title>]
Edit #<num> description:[<description>]
Edit #<num> time:[<time>]
Edit #<num> status:[<status>]
```

Trong đó:

- **<num>** là số để xác định công việc. Công việc đầu tiên được thêm vào ứng dụng có <num> bằng 1. Các công việc được thêm vào sau công việc đầu tiên sẽ bằng <num> của công việc được thêm trước đó cộng thêm 1. Do vậy, <num> hợp lệ là một số nguyên dương
- **<title>** là tiêu đề của công việc
- **<description>** là mô tả của công việc
- **<time>** là thời gian của công việc
- **<status>** là trạng thái của cộng việc. Một công việc có thể có 1 trong 3 trạng thái: ***In Progress*** (công việc đang được thực hiện), ***Done*** (công việc đã hoàn thành), ***Archived*** (công việc chưa hoàn thành nhưng không được thực hiện tiếp). Khi thêm một công việc mới vào ứng dụng, công việc sẽ có trạng thái là ***In Progress***.

Yêu cầu 6 (0.6 điểm): Hiện thực 3 hàm: **getTitleFromEdit**, **getDescriptionFromEdit**, **getTimeFromEdit** lần lượt lấy 3 trường thông tin <title>, <description>, <time>

từ câu lệnh Edit. Mô tả của 3 hàm như sau:

- Khai báo của 3 hàm:
  - `void getTitleFromEdit(char * command, char * out_title);`
  - `void getDescriptionFromEdit(char * command, char * out_description);`
  - `void getTimeFromEdit(char * command, char * out_time);`
- Đầu vào:
  - **command (kiểu char *)**: chuỗi chứa một câu lệnh ***Edit***
- Đầu ra:
  - **out_title**: chứa tiêu đề của câu lệnh ***Edit***
  - **out_description**: chứa mô tả của câu lệnh ***Edit***
  - **out_time**: chứa thời gian của ***Edit***
- Yêu cầu hàm:
  - Hàm **getTitleFromEdit** tìm và gán tiêu đề của câu lệnh add vào **out_title**
  - Hàm **getDescriptionFromEdit** tìm và gán mô tả của câu lệnh add vào **out_description**
  - Hàm **getTimeFromEdit** tìm và gán thời gian của câu lệnh add vào **out_time**

  Mỗi hàm làm đúng trong phần này được **0.2 điểm**.

3.7 Số trong các câu lệnh

Số của công việc xuất hiện trong một số kiểu lệnh khác nhau bên cạnh kiểu lệnh Edit. Tuy nhiên, ý nghĩa và điều kiện hợp lệ của số trong các câu lệnh đều giống với mô tả trong Mục 3.6.

Yêu cầu 7 (0.3 điểm): Hiện thực hàm **getNumFromCommand** với mô tả sau:

- Khai báo của hàm: **int getNumFromCommand(char * command)**
- Đầu vào:
  - **command**: chuỗi chứa câu lệnh
- Đầu ra:
  - Nếu câu lệnh không có thành phần **<num>** thì trả về giá trị -1. Nếu một câu lệnh có thành phần <num> thì nó sẽ nằm sau ký tự '#' trong câu lệnh
  - Nếu câu lệnh có thành phần **<num>** nhưng <num> không hợp lệ thì trả về giá trị 0
  - Nếu câu lệnh có thành phần **<num>** và <num> là hợp lệ thì trả về giá trị <num>

3.8 Thông tin cần thay đổi của câu lệnh Edit

Trong câu lệnh Edit, thông tin cần thay đổi được xác định bằng chuỗi nằm sau khoảng trắng thứ hai và trước dấu hai chấm đầu tiên. Thông tin cần thay đổi chỉ nhận 4 chuỗi hợp lệ là "title" (thay đổi tiêu đề), "description" (thay đổi mô tả), "time" (thay đổi thời gian), "status" (thay đổi trạng thái).

Yêu cầu 8 (0.3 điểm): Hiện thực hàm **getFieldFromEdit** với mô tả sau:

- Khai báo của hàm: **int getFieldFromEdit(char * edit_cmd)**
- Đầu vào:
  - **edit_cmd**: chuỗi chứa câu lệnh Edit
- Đầu ra:
  - Nếu thành phần cần thay đổi là tiêu đề thì trả về giá trị 1
  - Nếu thành phần cần thay đổi là mô tả thì trả về giá trị 2
  - Nếu thành phần cần thay đổi là thời gian thì trả về giá trị 3
  - Nếu thành phần cần thay đổi là trạng thái thì trả về giá trị 4
  - Nếu thành phần cần thay đổi không hợp lệ thì trả về giá trị 0

3.9 Trạng thái trong câu lệnh Edit

Trạng thái trong câu lệnh Edit là một chuỗi gồm có 1 ký tự có ý nghĩa như sau:

- Ký tự 'I' hoặc 'i' tương ứng với ***In Progress***
- Ký tự 'D' hoặc 'd' tương ứng với ***Done***
- Ký tự 'A' hoặc 'a' tương ứng với ***Archived***

Enum **Status** biểu diễn trạng thái gồm các giá trị IN_PROGRESS, DONE, ARCHIVED lần lượt tương ứng với các trạng thái ***In Progress***, ***Done***, ***Archived***.

Yêu cầu 9 (0.3 điểm): Hiện thực hàm **getStatusFromEdit** với mô tả sau:

- Khai báo của hàm: **enum Status getStatusFromEdit(char * edit_cmd)**
- Đầu vào:
  - **edit_cmd**: chuỗi chứa câu lệnh Edit
- Đầu ra:
  - Giá trị enum Status tương ứng với trạng thái trong câu lệnh Edit

3.10 Cấu trúc mô tả công việc

Một công việc trong ứng dụng được biểu diễn bởi cấu trúc (struct) **Task**. Task gồm 4 trường dữ liệu:

- **num**: số của công việc
- **title**: tiêu đề của công việc
- **description**: mô tả của công việc
- **status**: trạng thái của công việc

Hàm **printTask** được hiện thực sẵn để in ra màn hình thông tin của một Task. Sinh viên dùng hàm này nếu cần hiển thị Task trong các yêu cầu ở các mục sau.

Ví dụ 3: Cách sử dụng hàm **printTask**:

```
struct Task task;
printTask(&task);
```

3.11 Các thành phần trong câu lệnh Show

Câu lệnh ***Show*** được sử dụng để hiển thị một hoặc một số công việc. Câu lệnh Show có định dạng thuộc một trong các dòng sau:

```
Show #<num>
Show all
Show head %<quan>
Show tail %<quan>
Show filter title:[<title>]
Show filter description:[<description>]
Show filter status:[<status>]
Show week time:[<date>]
```

Trong đó:

- <num>, <title>, <description>, <status> có ý nghĩa giống như trong các mục trước
- <quan> là số lượng của công việc được hiển thị ra màn hình, <quan> nằm sau ký tự % trong câu lệnh Show

- <date> là một ngày và có định dạng như sau:

```
<DDD>/<dd>/<mo>/<yyyy>
```

  Trong đó: <dd>, <mo>, <yyyy> có ý nghĩa như trong các mục ở trên; <DDD> là một chuỗi có 3 ký tự biểu diễn một ngày trong tuần, <DDD> chỉ có thể nhận 1 trong 7 giá trị: "mon" (thứ 2), "tue" (thứ 3), "wed" (thứ 4), "thu" (thứ 5), "fri" (thứ 6), "sat" (thứ 7), "sun" (chủ nhật).

Yêu cầu 10 (0.3 điểm): Hiện thực hàm **printAllTasks** với mô tả sau:

- Khai báo của hàm: **void printAllTasks(struct Task * array_tasks, int no_tasks)**
- Đầu vào:
  - **array_tasks**: địa chỉ đến phần tử đầu tiên của một mảng các công việc
  - **no_tasks**: số lượng phần tử của mảng array_tasks
- Đầu ra: (không)
- Yêu cầu hàm: in ra các công việc trong mảng **array_task**, sử dụng hàm **printTask** khi in mỗi công việc.

Yêu cầu 11 (0.3 điểm): Hiện thực hàm **printTaskByNum** với mô tả sau:

- Khai báo của hàm: **void printTaskByNum(struct Task * array_tasks, int no_tasks, int num)**
- Đầu vào:
  - **array_tasks**: địa chỉ đến phần tử đầu tiên của một mảng các công việc
  - **no_tasks**: số lượng phần tử của mảng array_tasks
  - **num**: <num> tìm được trong câu lệnh Show
- Đầu ra: (không)
- Yêu cầu hàm: in ra công việc trong mảng **array_task** mà có biến thành viên **num** trùng với giá trị của tham số <num> được truyền vào, sử dụng hàm **printTask** khi in mỗi công việc.

Yêu cầu 12 (0.4 điểm): Hiện thực hàm **printHeadTasks** với mô tả sau:

- Khai báo của hàm: **void printHeadTasks(struct Task * array_tasks, int no_tasks, int quan)**

- Đầu vào:
  - **array_tasks**: địa chỉ đến phần tử đầu tiên của một mảng các công việc
  - **no_tasks**: số lượng phần tử của mảng array_tasks
  - **quan**: <quan> tìm được trong câu lệnh Show
- Đầu ra: (không)
- Yêu cầu hàm: in ra **quan** công việc đầu tiên trong mảng **array_task**, sử dụng hàm **printTask** khi in mỗi công việc. Nếu mảng có ít hơn **quan** công việc thì in ra toàn bộ mảng

Yêu cầu 13 (0.4 điểm): Hiện thực hàm **printTailTasks** với mô tả sau:

- Khai báo của hàm: **void printTailTasks(struct Task * array_tasks, int no_tasks, int quan)**
- Đầu vào:
  - **array_tasks**: địa chỉ đến phần tử đầu tiên của một mảng các công việc
  - **no_tasks**: số lượng phần tử của mảng array_tasks
  - **quan**: <quan> tìm được trong câu lệnh Show
- Đầu ra: (không)
- Yêu cầu hàm: in ra **quan** công việc cuối cùng trong mảng **array_task**, sử dụng hàm **printTask** khi in mỗi công việc. Nếu mảng có ít hơn **quan** công việc thì in ra toàn bộ mảng

Yêu cầu 14 (0.5 điểm): Hiện thực hàm **printFilteredTasksByTitle** với mô tả sau:

- Khai báo của hàm: **void printFilteredTasksByTitle(struct Task * array_tasks, int no_tasks, char * filter_title)**
- Đầu vào:
  - **array_tasks**: địa chỉ đến phần tử đầu tiên của một mảng các công việc
  - **no_tasks**: số lượng phần tử của mảng array_tasks
  - **filter_title**: <title> tìm được trong câu lệnh Show
- Đầu ra: (không)
- Yêu cầu hàm: in ra các công việc trong mảng **array_task** có **filter_title** là một chuỗi con của biến thành viên **title**, sử dụng hàm **printTask** khi in mỗi công việc

Yêu cầu 15 (0.5 điểm): Hiện thực hàm **printFilteredTasksByDescription** với mô tả sau:

- Khai báo của hàm: **void printFilteredTasksByDescription(struct Task * array_tasks, int no_tasks, char * filter_description)**
- Đầu vào:
  - **array_tasks**: địa chỉ đến phần tử đầu tiên của một mảng các công việc
  - **no_tasks**: số lượng phần tử của mảng array_tasks
  - **filter_description**: <description> tìm được trong câu lệnh Show
- Đầu ra: (không)
- Yêu cầu hàm: in ra các công việc trong mảng **array_task** có **filter_description** là một chuỗi con của biến thành viên **description**, sử dụng hàm **printTask** khi in mỗi công việc

Yêu cầu 16 (0.4 điểm): Hiện thực hàm **printFilteredTasksByStatus** với mô tả sau:

- Khai báo của hàm: **void printFilteredTasksByStatus(struct Task * array_tasks, int no_tasks, enum Status filter_status)**
- Đầu vào:
  - **array_tasks**: địa chỉ đến phần tử đầu tiên của một mảng các công việc
  - **no_tasks**: số lượng phần tử của mảng array_tasks
  - **filter_status**: <status> tìm được trong câu lệnh Show
- Đầu ra: (không)
- Yêu cầu hàm: in ra các công việc trong mảng **array_task** có giá trị của biến thành viên **status** trùng với giá trị của **filter_status**, sử dụng hàm **printTask** khi in mỗi công việc

3.12 Thêm công việc vào mảng

Một mảng được sử dụng để lưu trữ các công việc. Số lượng công việc tối đa được lưu trữ trong biến **MAX_NO_TASKS**.

Yêu cầu 17 (0.4 điểm): Hiện thực hàm **addTask** với mô tả sau:

- Khai báo của hàm: **bool addTask(struct Task * array_tasks, int no_tasks, char * new_title, char * new_description, char * new_time)**
- Đầu vào:
  - **array_tasks**: địa chỉ đến phần tử đầu tiên của một mảng các công việc
  - **no_tasks**: số lượng phần tử hiện tại của mảng array_tasks

  - **new_title**: tiêu đề của công việc mới muốn thêm vào mảng
  - **new_description**: mô tả của công việc mới muốn thêm vào mảng
  - **new_time**: thời gian của công việc mới muốn thêm vào mảng
- Đầu ra: trả về giá trị **true** nếu có thể thêm một công việc mới vào mảng, ngược lại trả về giá trị **false**.
- Yêu cầu hàm: Thêm một công việc mới vào cuối mảng với các biến thành viên tương ứng với **new_title**, **new_desciption**, **new_time**. Sinh viên tìm cách gán giá trị thích hợp cho **num** và **status** ở các mục phía trên.

3.13 Xóa công việc khỏi mảng

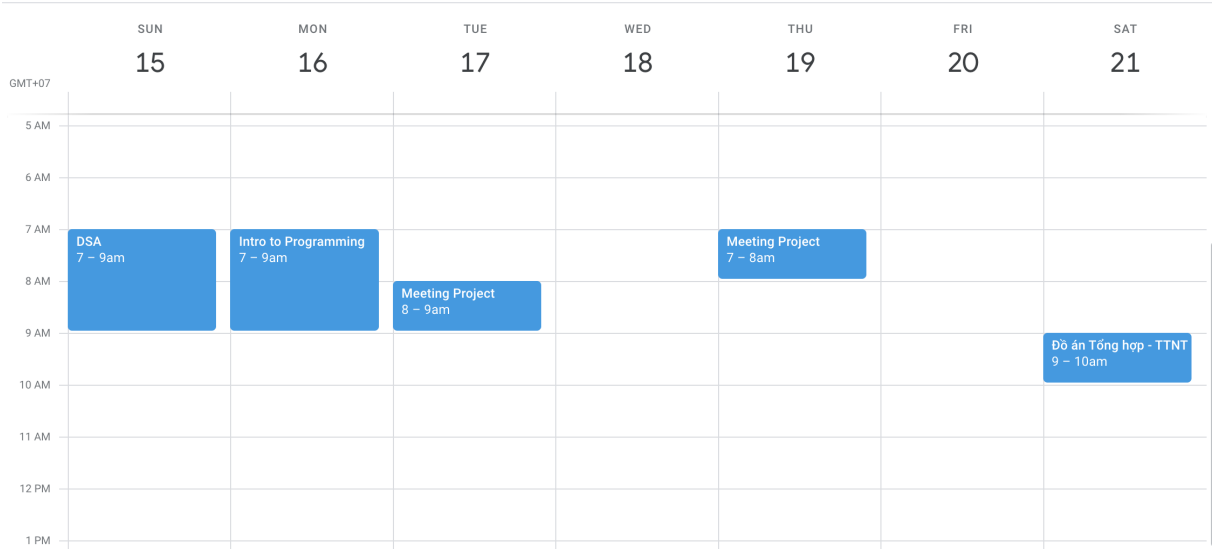
Yêu cầu 18 (0.4 điểm): Hiện thực hàm **deleteTask** với mô tả sau:

- Khai báo của hàm: **bool deleteTask(struct Task * array_tasks, int no_tasks, int num)**
- Đầu vào:
  - **array_tasks**: địa chỉ đến phần tử đầu tiên của một mảng các công việc
  - **no_tasks**: số lượng phần tử hiện tại của mảng array_tasks
  - **num**: số để xác định công việc, được lấy từ <num> trong câu lệnh Delete
- Đầu ra: trả về giá trị **true** nếu có thể xóa công việc được xác định bởi **num** ra khỏi mảng, ngược lại trả về giá trị **false**.
- Yêu cầu hàm: Xóa công việc được xác định bởi **num** ra khỏi mảng **array_tasks**. Sau đó, cập nhật biến thành viên **num** của các công việc trong mảng theo quy tắc đã nêu ở các mục trên.

3.14 Các công việc trong tuần

Ứng dụng Todo cung cấp cách thức hiển thị các công việc trong tuần để người dùng dễ dàng quan sát lịch làm việc. Hình 2 minh họa một cách hiển thị lịch theo tuần trong ứng dụng Calendar của Google. Câu lệnh để thực hiện tính năng này là **Show week**. Bên dưới là các đặc điểm của tính năng hiển thị công việc theo tuần trong Ứng dụng Todo đang được hiện thực.

Dòng đầu tiên của phần hiển thị gồm các thứ trong tuần và ngày, ví dụ về nội dung của 1 ô ở dòng đầu tiên:



Hình 2: Hiển thị lịch làm việc theo tuần trong Google Calendar

```
 MON
16/10
```

Các thứ ở dòng đầu tiên bắt đầu từ Thứ 2 (MON) thay vì Chủ Nhật (SUN) như Hình 2. Ô ở góc trên bên trái hiển thị năm của tuần hiện tại thay vì "GMT+07" như trong Hình 2. Cột đầu tiên hiển thị các mốc thời gian theo giờ từ "00:00" (0 giờ 0 phút), "01:00", "02:00", ..., "13:00" (1 giờ chiều),..., "23:00", "00:00". Cột này đếm thời gian trong ngày là từ 0 đến 24 giờ, thay vì sử dụng 12 giờ AM cho sáng và 12 giờ PM cho chiều. Các chuỗi ký tự trong các ô ở dòng đầu tiên và cột đầu tiên phải được căn giữa theo độ rộng của ô. Độ rộng của các ô ở cột đầu tiên được lưu trong biến **WEEK_CELL_FIRST_COL_WIDTH**. Độ rộng của các ô còn lại ngoài cột đầu tiên được lưu trong biến **WEEK_CELL_OTHER_COL_WIDTH**.

Để in một chuỗi **s** có độ dài là **ws** được căn giữa trong một ô có độ rộng là **wc**, ta tính số lượng khoảng trắng **p** phải in ra trước khi in chuỗi **s** như sau:

$$p = \lfloor (wc - ws)/2 \rfloor$$

Trong đó, $\lfloor x \rfloor$ là phép toán làm tròn xuống (floor) giá trị x.

Các ô không ở cột đầu tiên và không ở dòng đầu tiên là các ô biểu diễn công việc. Một công việc có thể cần 1 ô hoặc nhiều ô liên tục để biểu diễn, ta gọi các ô biểu diễn công việc là khối ô công việc. Khi biểu diễn một khối ô công việc, ta không biểu diễn đường gạch nối giữa các ô trong khối ô. Ví dụ minh họa chi tiết sẽ được cung cấp tại nơi nộp bài trên BKeL. Thông tin in ra màn hình của mỗi khối ô gồm 2 dòng. Dòng 1 gồm số của công việc, thời gian bắt đầu

và thời gian kết thúc công việc. Bên dưới là định dạng của dòng 1, trong đó <hh> và <mm> lần lượt là giờ và phút, gồm 2 ký tự chữ số. <hh>, <mm> trước dấu gạch ngang biểu diễn thời gian bắt đầu. <hh>, <mm> sau dấu gạch ngang biểu diễn thời gian kết thúc. <num> là số của công việc.

```
#<num>|<hh>:<mm>-<hh>:<mm>
```

Dòng 2 ghi tiêu đề của công việc. Chú ý rằng một ô cho công việc chỉ có độ rộng là **WEEK_CELL_OTHER_COL_WIDTH**. Nếu độ dài của tiêu đề công việc dài hơn độ rộng của ô thì tiêu đề được hiển thị một phần và hiển thị dấu 3 chấm cuối cùng để biểu diễn là tên tiêu đề vẫn còn. Do vậy, tiêu đề sẽ được hiển thị ít hơn độ rộng của ô 3 ký tự. 3 ký tự cuối cùng được thay thế bằng 3 ký tự ".".

Để đơn giản, Ứng dụng Todo đang hiện thực chỉ hiển thị công việc theo tuần nếu tất cả các công việc trong tuần đó đều có thời gian bắt đầu và thời gian kết thúc là một thời gian chẵn. Thời gian chẵn là thời gian có phút là "00". Khi phát hiện một công việc trong tuần có thời gian không chẵn, ta coi đó là công việc gây ra lỗi hiển thị theo tuần. Khi phát hiện công việc gây lỗi, ứng dụng gọi hàm **printUnsupportedTime(struct Task * task)** truyền vào công việc có thời gian không chẵn.

Yêu cầu 19 (2.0 điểm): Hiện thực hàm **printWeekTime** với các mô tả sau:

- Khai báo của hàm: **int printWeekTime(struct Task * array_tasks, int no_tasks, char * date)**
- Đầu vào:
  - **array_tasks**: địa chỉ đến phần tử đầu tiên của một mảng các công việc
  - **no_tasks**: số lượng phần tử hiện tại của mảng array_tasks
  - date: gồm thông tin về thứ, ngày, tháng, năm, date tương ứng với <date> từ câu lệnh **Show week time:[<date>]**
- Đầu ra: trả về giá trị **-1** nếu có thể hiển thị công việc theo tuần, ngược lại trả về vị trí của công việc đầu tiên trong mảng **array_tasks** gây ra lỗi không thể hiển thị theo tuần
- Yêu cầu hàm: Hiển thị công việc theo tuần như mô tả ở trên trong mục này. Tuần được hiển thị là tuần được xác định bởi tham số **date**. Nếu không thể hiển thị được theo tuần thì gọi hàm **printUnsupportedTime** và truyền vào công việc gây ra lỗi không thể hiển thị. Nếu có nhiều công việc cùng gây lỗi thì in ra công việc đầu tiên tính từ đầu mảng **array_tasks**.

3.15 Kiểm tra tất cả các hàm

Yêu cầu 20 (0.7 điểm): Một số testcase kiểm tra tổng hợp các hàm với nhau.

4 Nộp bài

Sinh viên download file các file sau từ trang Web của môn học:

todoapp.c	Mã nguồn khởi tạo
NMLT-TodoApp-Assignment.pdf	File mô tả nội dung bài tập lớn

File todoapp.c là mã nguồn khởi tạo. Sinh viên **phải** sử dụng mã nguồn này để viết phần hiện thực nằm giữa 2 dòng sau:

- // —— **Begin: Student Answer** ——
- // —— **End: Student Answer** ——

Khi nộp bài, sinh viên nộp bài trên site e-Learning của môn học. Sinh viên điền code bài tập lớn giống như các bài thực hành khác. Nội dung điền vào sẽ là phần code sinh viên hiện thực nằm giữa 2 dòng trên. Sinh viên **không được phép** include bất kỳ thư viện nào ngoài các thư viện đã có sẵn trong mã nguồn khởi tạo. Sinh viên được cung cấp các nơi nộp bài:

- **Nơi nộp bài thử**: Sinh viên nộp bài làm và được chấm trên **5 testcases** để kiểm tra các lỗi cú pháp, lỗi logic cơ bản có thể có của bài làm sinh viên. Sinh viên được phép nộp bài **vô số lần** ở đây
- Các nơi nộp bài chính thức sẽ được thông báo sau

Trong mỗi phần trên, ngoại trừ **Nơi nộp bài thử**, sinh viên có tối đa **10 lần** làm bài. Đối với mỗi lần nộp bài, sinh viên có **10 phút** để nộp code và kiểm tra. Chỉ có lần nhấn "Kiểm tra" đầu tiên là được tính điểm, các lần sau sẽ không được lấy điểm. Kết quả bài làm chỉ hiển thị sau khi bạn nhấn nút "Hoàn thành bài làm". Điểm cao nhất trong các lần làm bài sẽ được lấy làm điểm cho phần đó.

Thời hạn nộp bài được công bố tại nơi nộp bài trong site nêu trên. Đến thời hạn nộp bài, đường liên kết sẽ tự động khoá nên sinh viên sẽ không thể nộp chậm. Để tránh các rủi ro có thể xảy ra vào thời điểm nộp bài, sinh viên PHẢI nộp bài trước thời hạn quy định ít nhất **một giờ**.

Sinh viên phải kiểm tra chương trình của mình trên MinGW và nơi nộp bài thử trước khi nộp.

5 Harmony cho Bài tập lớn

Bài kiểm tra cuối kì của môn học sẽ có một số câu hỏi Harmony với nội dung của BTL. Giả sử điểm BTL mà sinh viên đạt được là **a** (theo thang điểm 10), tổng điểm các câu hỏi Harmony **b** (theo thang điểm 5). Gọi **x** là điểm của BTL sau khi Harmony, cũng là điểm BTL cuối cùng của sinh viên. Các câu hỏi cuối kì sẽ được Harmony với 50% điểm của BTL theo công thức sau:

- Nếu $a = 0$ hoặc $b = 0$ thì $x = 0$
- Nếu a và b đều khác 0 thì

$$x = \frac{a}{2} + HARM(\frac{a}{2}, b)$$

Trong đó:

$$HARM(x, y) = \frac{2xy}{x + y}$$

Sinh viên phải giải quyết BTL bằng khả năng của chính mình. Nếu sinh viên gian lận trong BTL, sinh viên sẽ không thể trả lời câu hỏi Harmony và nhận điểm 0 cho BTL.

Sinh viên **phải** chú ý làm câu hỏi Harmony trong bài kiểm tra cuối kỳ. Các trường hợp không làm sẽ tính là 0 điểm cho BTL, và bị không đạt cho môn học. **Không chấp nhận giải thích và không có ngoại lệ.**

6 Xử lý gian lận

Bài tập lớn phải được sinh viên TỰ LÀM. Sinh viên sẽ bị coi là gian lận nếu:

- Có sự giống nhau bất thường giữa mã nguồn của các bài nộp. Trong trường hợp này, TẤT CẢ các bài nộp đều bị coi là gian lận. Do vậy sinh viên phải bảo vệ mã nguồn bài tập lớn của mình.
- Sinh viên không hiểu mã nguồn do chính mình viết, trừ những phần mã được cung cấp sẵn trong chương trình khởi tạo. Sinh viên có thể tham khảo từ bất kỳ nguồn tài liệu nào, tuy nhiên phải đảm bảo rằng mình hiểu rõ ý nghĩa của tất cả những dòng lệnh mà mình viết. Trong trường hợp không hiểu rõ mã nguồn của nơi mình tham khảo, sinh viên được đặc biệt cảnh báo là KHÔNG ĐƯỢC sử dụng mã nguồn này; thay vào đó nên sử dụng những gì đã được học để viết chương trình.
- Nộp nhầm bài của sinh viên khác trên tài khoản cá nhân của mình.

Trong trường hợp bị kết luận là gian lận, sinh viên sẽ bị điểm 0 cho toàn bộ môn học (không chỉ bài tập lớn).

KHÔNG CHẤP NHẬN BẤT KỲ GIẢI THÍCH NÀO VÀ KHÔNG CÓ BẤT KỲ NGOẠI LỆ NÀO!

Sau mỗi bài tập lớn được nộp, sẽ có một số sinh viên được gọi phỏng vấn ngẫu nhiên để chứng minh rằng bài tập lớn vừa được nộp là do chính mình làm.

7 Thay đổi so với phiên bản trước

- Cập nhật mã nguồn khởi tạo (source code): Dòng 5-14 trong file init (thay các giá trị const thành define)
- Tại yêu cầu số 19, các testcase sẽ chỉ bao gồm công việc trong một ngày và không có các công việc cùng giờ bắt đầu

——————————HẾT——————————