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

NGÀY HỘI VIỆC LÀM

TP. HỒ CHÍ MINH, THÁNG 10/2024

ĐẶC TẢ BÀI TẬP LỚN

Phiên bản 5.0 (Cập nhật ngày 10/11/2024)

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

Ngày hội việc làm là sự kiện kết nối sinh viên với doanh nghiệp thông qua các gian hàng, từ đó doanh nghiệp có thể giới thiệu về các vị trí tuyển dụng, các sinh viên, ứng viên có thể tham gia nếu cảm thấy phù hợp. Đây cũng là dịp giúp sinh viên tìm hiểu về nhu cầu thực tế của thị trường tuyển dụng.

Ngoài ra khi đăng kí doanh nghiệp cũng cần biết số lượng gian hàng trống hiện tại và danh sách các gian hàng trống hiện tại.

Mỗi doanh nghiệp khi đăng kí cần có một tên đầy đủ và một tên viết tắt. Trong bài toán này, chúng ta quy định tên viết tắt của mỗi doanh nghiệp là chữ cái đầu của mỗi từ trong tên doanh nghiệp.

Gian hàng được bố trí thành 2 dãy đối diện nhau. Trong bài tập lớn này, mỗi dãy sẽ có 15 gian hàng.

Trong Bài tập lớn này, chúng ta sẽ tạo ra những tính năng cơ bản của một ứng dụng hỗ trợ ban tổ chức (BTC) sắp xếp gian hàng, tổ chức sự kiện.

3 Yêu cầu ứng dụng

Mảng trạng thái gian hàng: Đây là một mảng hai chiều số nguyên với mỗi chiều gồm 15 phần tử. Chỉ số của gian hàng trong hàng đầu tiên có giá trị từ 0 đến 14. Đối với hàng thứ hai, chỉ số của các gian hàng được tính bằng chỉ số của hàng đầu cộng thêm 15, tức là chỉ số bắt đầu từ 15 và tăng dần (gian hàng đầu tiên là 15, gian hàng thứ hai là 16, v.v.). Trạng thái của gian hàng bao gồm:

- Gian hàng trống: 0
- Gian hàng đã có đăng kí: 1

Mảng các doanh nghiệp : Là một mảng 1D kiểu **struct Enterprise** với 30 phần tử, trong đó mỗi giá trị là thông tin mỗi doanh nghiệp đã đăng kí thành công lần lượt được thêm vào **theo thứ tự đăng kí thành công**, không ảnh hưởng bởi thứ tự của chỉ số gian hàng.

Trong bài toán này, để làm đơn giản sơ đồ bố trí và trạng thái gian hàng, mảng giá trị này được lưu tách biệt với mảng thông tin các doanh nghiệp. Do vậy khi thực hiện thao tác thay đổi dữ liệu, sinh viên cần cập nhật một lượt cả 2 mảng giá trị trên để đảm bảo đồng bộ dữ liệu. Trong trường hợp tối thiểu, để đạt được 50% điểm bài tập này, sinh viên cần cập nhật ít nhất mảng trạng thái gian hàng và bỏ qua mảng các doanh nghiệp.

3.1 Đăng kí tên viết tắt

Như mô tả ở phần trên, mỗi doanh nghiệp cần tên viết tắt. Quy ước tên viết tắt của doanh nghiệp là các chữ cái đầu trong tên doanh nghiệp. Sau đây là một ví dụ:

Ví dụ 1: Các ví dụ về tên viết tắt của doanh nghiệp

- Trường Minh Thịnh: TMT
- VNG:V
- FPT Software: FS

Giả sử trong bài toán này dữ liệu đầu vào sẽ không có doanh nghiệp trùng tên viết tắt.

Yêu cầu 1 : Hiện thực hàm getAbbreviation(char* name, char* abbre)

- Tên hàm: getAbbreviation
- Tham số đầu vào:

  - name (kiểu char*): tên của doanh nghiệp, dữ liệu truyền vào, không được thay đổi trong quá trình xử lý.
  - abbre (kiểu char*): chuỗi chứa tên viết tắt của doanh nghiệp
- Trả về: Tên viết tắt của doanh nghiệp gán trong tham số abbre.
- Yêu cầu: Tạo tên viết tắt cho tên doanh nghiệp

3.2 Kiểu câu lệnh

Do ứng dụng được thao tác thông qua dòng lệnh (command line) nên chương trình cần xác định kiểu của câu lệnh là gì để gọi hàm tương ứng.

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 REGISTER, ALTER, SHOW, DELETE, QUIT, INVALID lần lượt tương ứng với các kiểu lệnh Register, Alter, Show, Delete, Quit và lệnh không hợp lệ.

Yêu cầu 2 (0.5 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 2: Các ví dụ về **getCommandType**:

- Lời gọi hàm

```
getCommandType("Show [:10]")
```

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
```

3.3 Doanh nghiệp

Yêu cầu 3 : Định nghĩa phần thân của struct Enterprise

Yêu cầu: Định nghĩa phần thân của struct Enterprise đã được khởi tạo. Bao gồm các thuộc tính sau:

- booth_index: Kiểu số nguyên, lưu thông tin chỉ số của gian hàng mà doanh nghiệp đăng kí.
- name: Kiểu mảng kí tự, lưu tên đầy đủ của doanh nghiệp.
- abbre: Kiểu mảng kí tự, lưu tên viết tắt của doanh nghiệp.
- itemValue: Giá trị của vật phẩm tại gian hàng (chi tiết tại phần 3.9)

- itemWeight: Khối lượng của vật phẩm tại gian hàng (chi tiết tại phần 3.9)

Yêu cầu 4 : Định nghĩa hàm printEnterpriseDetails(Enterprise e)

- Tên hàm: printEnterpriseDetails
- Tham số đầu vào:
  - Enterprise e: Doanh nghiệp e cần in thông tin
- Yêu cầu: In thông tin doanh nghiệp theo cấu trúc:

```
||booth_index||name||abbre||itemValue||itemWeight||
```

Để tạo doanh nghiệp, chúng ta cần hiện thực một hàm createEnterprise.

Yêu cầu 5 : Định nghĩa hàm createEnterprise(Enterprise* e, int booth_index, int itemValue, int itemWeight, char* name, char* abbre)

- Tên hàm: createEnterprise
- Tham số đầu vào:
  - Enterprise* e: Doanh nghiệp cần tạo ra. Trong trường hợp giá trị truyền vào e là **nullptr**, cần cấp phát vùng nhớ cho e trước khi thực hiện.
  - booth_index, name, abbre, itemValue, itemWeight: Các giá trị gán tương ứng vào thuộc tính của e
- Yêu cầu: Tạo ra doanh nghiệp **e**.

3.4 Các thành phần câu lệnh đăng kí (Register)

Trước khi diễn ra, BTC sẽ gửi cho tất cả các doanh nghiệp có liên quan một sơ đồ gian hàng.

Cú pháp để đăng kí gian hàng là

```
Register [<name>] [itemValue] [itemWeight] [booth_index]
```

Trong đó:

- name: Tên đầy đủ của Doanh nghiệp
- booth_index: Chỉ số gian hàng đăng kí
- itemValue: Giá trị của vật phẩm tại gian hàng (Xem phần 3.9) item itemWeight: Khối lượng của vật phẩm tại gian hàng (Xem phần 3.9)

Nếu phần booth_index trống, mặc định gian hàng được đăng kí của doanh nghiệp là gian hàng tại index tính theo công thức:

$$index = (num\_abbre * 30)\%26$$

trong đó: *num_abbre*: Số ký tự trong tên viết tắt của doanh nghiệp.

Trong riêng trường hợp booth_number trống này, nếu tại *index* tính ra trạng thái không hợp lệ, tức là đã có đăng kí tại vị trí đó, để đăng kí thì tiếp tục tăng index lên 1 đơn vị cho đến khi gặp gian hàng trống. Trong trường hợp không tìm được gian hàng trống khi tăng index đến giới hạn thì lùi index về 1 đơn vị cho đến khi tìm được gian hàng trống. Tại vị trí đó chuyển trạng thái gian hàng về đã đăng kí và trả về đăng kí thành công.

Trong trường hợp còn lại, khi đăng kí gian hàng, nếu trạng thái gian hàng đang trống, chuyển trạng thái thành đã đăng kí và trả về đăng kí thành công. Ngược lại, trả về đăng kí không thành công.

Khi đăng kí thành công, cập nhật tương ứng giá trị trong mảng các doanh nghiệp.

Giả sử các câu lệnh Register gọi vào đều hợp lệ và chỉ thuộc 1 trong 2 trường hợp trên.

Yêu cầu 6 : Hiện thực hàm registerEnterprise(int map[2][15], Enterprise enterpriseArray[30], char* name, int booth, int itemValue, int itemWeight, int* out_booth, char* out_abbre)

- Tên hàm: registerEnterprise
- Tham số đầu vào:
  - map[2][15]: Mảng trạng thái gian hàng
  - enterpriseArray[30]: Mảng các doanh nghiệp
  - name: Tên đầy đủ của doanh nghiệp
  - booth: Chỉ số gian hàng mà doanh nghiệp muốn đăng kí. Trường hợp câu lệnh Register trống tham số này, giá trị của booth được truyền vào là -1.
  - itemValue: Giá trị vật phẩm tại gian hàng.
  - itemWeight: Khối lượng vật phẩm tại gian hàng.
  - out_booth: Giá trị gian hàng đăng kí
  - out_abbre: Tên viết tắt của doanh nghiệp
- Trả về: $booth\_index + 200$ và gán vào *out_booth* nếu đăng kí thành công, nếu không đăng kí thành công trả về $booth\_index + 100$ và gán vào *out_booth*. Tên viết tắt gán vào *out_abbre*.

- Yêu cầu: Giúp doanh nghiệp đăng kí tên viết tắt và gian hàng. Trả về giá trị tương ứng vào 2 biến *out_booth* và *out_abbre*, cập nhật tương ứng vào 2 mảng trạng thái và mảng các doanh nghiệp.

Ví dụ 3: Một câu lệnh Register đầy đủ: `Register [VNG] [1] [2] [2]`
Giả sử hiện tại tất cả trạng thái của gian hàng đều là trống (tức đều là 0).
Nội dung thực hiện bao gồm:

- Tạo ra tên viết tắt cho doanh nghiệp -> Gọi hàm getAbbreviation và trả về chuỗi "V". Gán giá trị này vào out_abbre.
- Kiểm tra và cập nhật map[0][1] = 1. Gán giá trị 1 vào out_booth.
- Tạo ra enterprise với abbre là "V", itemValue là 2, itemWeight là 2 và booth là 1 rồi thêm vào mảng tại vị trí enterpriseArray[0].

Ví dụ 4: Một câu lệnh Register khuyết: `Register [FPT Software] [] [2] [2]`
Giả sử trạng thái của các gian hàng hiện tại đã có đăng kí tại gian *7, 8, 9,...,30*.
Nội dung thực hiện bao gồm:

- Tạo ra tên viết tắt cho doanh nghiệp -> Gọi hàm getAbbreviation và trả về chuỗi "FS". Gán giá trị này vào out_abbre.
- Tính chỉ số của gian hàng đăng kí bằng công thức:

$$index = (2 * 30)\%26 = 8$$

  Tại map[0][8] trạng thái đang là 1 nên tiếp tục tăng index lên, tuy nhiên, các gian hàng tiếp theo đều trạng thái là 1 nên sẽ lùi index lại.
  Tại map[0][7] trạng thái vẫn đang là 1 nên tiếp tục lùi đến $index = 6$, tại đây trạng thái đang là 0. Do vậy $index = 6$ thoả mãn.
- Với $index = 6$, $itemValue = 2$, $itemWeight = 2$ và $abbre = "FS"$, thực hiện việc đăng kí giống các bước ở ví dụ trên.

3.5 Các thành phần câu lệnh hiển thị (Show)

Câu lệnh ***Show*** được sử dụng để hiển thị một hoặc một số gian hàng. Câu lệnh Show có thể là:

```
Show map
Show &0
Show &1
Show #0
Show #1
Show [num]
Show [from_index:to_index]
```

Trong đó:

- Show map: In trạng thái theo sơ đồ gian hàng. Định dạng in như sau:

```
|0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|
--
|status_0|...|
--
|status_15|...|
```

- Show &0: Hiển thị chỉ số tất cả gian hàng đang có trạng thái chưa có đăng kí. In theo cú pháp:

```
Index[index_0, index_1,...]
```

- Show &1: Hiển thị chỉ số tất cả gian hàng đang có trạng thái đã có đăng kí. In theo cú pháp:

```
Index[index_0, index_1,...]
```

- Show #0: Hiển thị tổng số lượng gian hàng đang có trạng thái chưa có đăng kí. In theo cú pháp:

```
Total: total_0
```

- Show #1: Hiển thị tổng số lượng gian hàng đang có trạng thái đã đăng kí. In theo cú pháp:

```
Total: total_1
```

- Show [num]: Hiển thị thông tin chi tiết của gian hàng tại vị trí chỉ số num. Trong trường hợp tại vị trí num đang trống, hiển thị "NONE". Cú pháp in tương đương với cách gọi hàm **printEnterpriseDetails**
- Show [from_index:to_index]: Hiển thị thông tin chi tiết của gian hàng từ vị trí chỉ số from_index đến vị trí to_index. Trong trường hợp tại vị trí num đang trống, hiển thị "NONE". Cú pháp in:

```
Index index_0: [enterprise_at_index_0]
Index index_1: [enterprise_at_index_1]
...
```

Yêu cầu 7 : Hiện thực hàm showMap(int map[2][15], Enterprise enterpriseArray[30])

- Tên hàm: **showMap**
- Tham số đầu vào:
  - **map[2][15]**: Mảng trạng thái gian hàng
  - **enterpriseArray[30]**: Mảng thông tin các doanh nghiệp đã đăng ký.
- Yêu cầu: Hiển thị ra màn hình sơ đồ trạng thái của gian hàng theo cú pháp của câu lệnh "Show map". Mỗi gian hàng sẽ được in theo đúng thứ tự chỉ số của nó.

Yêu cầu 8 : Hiện thực hàm showIndexOfStatus(int map[2][15], int status)

- Tên hàm: **showIndexOfStatus**
- Tham số đầu vào:
  - **map[2][15]**: Mảng trạng thái gian hàng
  - **status**: Trạng thái gian hàng cần in
- Yêu cầu: Hiển thị ra màn hình danh sách các chỉ số gian hàng có trạng thái tương ứng, theo cú pháp của câu lệnh "Show 0" hoặc "Show 1".

Yêu cầu 9 : Hiện thực hàm showTotalOfStatus(Enterprise enterpriseArray[30], int status)

- Tên hàm: **showTotalOfStatus**
- Tham số đầu vào:
  - **enterpriseArray[30]**: Mảng thông tin các doanh nghiệp.
  - **status**: Trạng thái của gian hàng cần đếm (0: trống, 1: đã đăng ký).

- Yêu cầu: Hiển thị ra màn hình tổng số lượng gian hàng có trạng thái tương ứng, theo cú pháp của câu lệnh "Show #0" hoặc "Show #1".

Yêu cầu 10 : Hiện thực hàm showIndexBracket(int map[2][15], Enterprise enterpriseArray[30], int start, int end)

- Tên hàm: **showIndexBracket**
- Tham số đầu vào:
  - **map[2][15]**: Mảng trạng thái gian hàng
  - **enterpriseArray[30]**: Mảng thông tin các doanh nghiệp đã đăng ký.
  - **start**: Chỉ số gian hàng bắt đầu cần hiển thị.
  - **end**: Chỉ số gian hàng kết thúc cần hiển thị. Nếu giá trị của **end** là -1, chỉ in thông tin của gian hàng tại vị trí **start**.
- Yêu cầu: Hiển thị ra màn hình thông tin của các gian hàng từ vị trí **start** đến **end** theo cú pháp của câu lệnh "Show [num]" hoặc "Show [from_index:to_index]".

Yêu cầu 11 : Hiện thực hàm handleShowCommandint map[2][15], Enterprise enterpriseArray[30], char *command)

- Tên hàm: **handleShowCommand**
- Tham số đầu vào:
  - map[2][15]: Mảng trạng thái gian hàng, mỗi phần tử biểu diễn trạng thái của một gian hàng (0: trống, 1: đã đăng ký).
  - **enterpriseArray[30]**: Mảng chứa thông tin các doanh nghiệp đã đăng ký.
  - **command**: Chuỗi chứa câu lệnh "Show" cần phân tích.
- Yêu cầu: Phân tách câu lệnh "Show" truyền vào **command** để gọi hàm tương ứng. Ví dụ: nếu câu lệnh là "Show map", gọi hàm **showMap**; nếu là "Show &0", gọi hàm **showIndexOfStatus**.

3.6 Các thành phần câu lệnh thay đổi thông tin (Alter)

Câu lệnh Alter được sử dụng để thay đổi gian hàng mà doanh nghiệp đã đăng ký. Các thành phần ***itemValue, itemWeight*** được quy định không thể thay đổi bằng câu lệnh Alter.

Cú pháp của câu lệnh:

```
Alter [<abbre>] [reigster_booth] [new_booth]
```

Trong đó:

- abbre: Tên viết tắt của doanh nghiệp
- register_booth: Chỉ số gian hàng đã đăng kí
- new_booth: Chỉ số gian hàng muốn thay đổi sang

Mô tả chi tiết

1. **Trường hợp `new_booth` trống:**
   - Mặc định, nếu tham số `new_booth` không được chỉ định, hệ thống sẽ cố gắng tìm gian hàng trống bằng cách bắt đầu từ chỉ số gian hàng hiện tại (`register_booth`), sau đó tăng dần chỉ số (index) lên 1 đơn vị.
   - Nếu tại chỉ số mới, gian hàng không trống (không hợp lệ để đăng ký), hệ thống sẽ tiếp tục tăng chỉ số cho đến khi tìm thấy một gian hàng trống.
   - Nếu việc tăng chỉ số đến giới hạn mà không tìm thấy gian hàng trống, hệ thống sẽ lùi dần chỉ số xuống 1 đơn vị (từ gian hàng đã đăng ký) cho đến khi tìm được gian hàng trống.
   - Nếu không còn gian hàng trống, trả về kết quả thay đổi không thành công.
2. **Trường hợp `new_booth` được chỉ định:**
   - Nếu chỉ số `new_booth` được chỉ định và gian hàng tại vị trí này đang trống, hệ thống sẽ chuyển gian hàng của doanh nghiệp sang chỉ số mới và trả về kết quả thành công.
   - Nếu chỉ số `new_booth` đã có doanh nghiệp khác đăng ký (gian hàng không trống), hệ thống sẽ trả về kết quả không thành công.

Giả sử các câu lệnh Alter gọi vào đều hợp lệ và chỉ thuộc 1 trong 2 trường hợp trên.

Yêu cầu bổ sung:

Sau khi thay đổi thành công, hệ thống cần cập nhật cả mảng trạng thái gian hàng và mảng thông tin doanh nghiệp để đảm bảo đồng bộ dữ liệu.

Yêu cầu 12 : Hiện thực hàm alterEnterprise(int map[2][15], Enterprise enterpriseArray[30], char* abbre, int registerBooth, int newBooth, int* out_booth, char* out_abbre)

- Tên hàm: alterEnterprise
- Tham số đầu vào:

  - map[2][15]: Mảng trạng thái gian hàng, mỗi phần tử biểu diễn trạng thái của một gian hàng (0: trống, 1: đã đăng ký).
  - enterpriseArray[30]: Mảng chứa thông tin về các doanh nghiệp đã đăng ký.
  - abbre: Tên viết tắt của doanh nghiệp cần thay đổi gian hàng.
  - registerBooth: Chỉ số gian hàng hiện tại mà doanh nghiệp đã đăng ký.
  - newBooth: Chỉ số gian hàng mới mà doanh nghiệp muốn đổi sang. Nếu giá trị này là -1, hệ thống sẽ tự động tìm gian hàng trống gần nhất.
  - out_booth: Con trỏ chứa giá trị gian hàng đã đăng ký sau khi thay đổi.
  - out_abbre: Con trỏ chứa tên viết tắt của doanh nghiệp.

- Trả về: Giá trị trả về là $booth\_index + 200$ nếu việc điều chỉnh thành công, và gán giá trị này vào *out_booth*. Nếu không thành công, trả về $booth\_index + 100$ và gán giá trị vào *out_booth*.
- Yêu cầu: Giúp doanh nghiệp thay đổi gian hàng. Cập nhật giá trị vào *out_booth* và *out_abbre*, đồng thời điều chỉnh cả mảng trạng thái gian hàng và mảng thông tin doanh nghiệp để đảm bảo dữ liệu đồng bộ. Nếu tham số *newBooth* không được chỉ định, hệ thống sẽ tự động tìm và đăng ký gian hàng trống gần nhất.

3.7 Các thành phần câu lệnh xoá (Delete)

Gian hàng đã được đăng ký có thể xoá bằng câu lệnh Delete. Cú pháp của câu lệnh:

```
Delete [<abbre>] [reigsterBooth]
```

Trong đó:

- abbre: Tên viết tắt của doanh nghiệp
- register_booth: Chỉ số gian hàng đã đăng kí

Giả sử gian hàng được xoá là gian hàng của doanh nghiệp tương ứng, không có trường hợp ngoại lệ. Xoá là hành động cập nhật trạng thái trong mảng trạng thái gian hàng tại vị trí đó thành trống và xoá tương ứng doanh nghiệp ra khỏi mảng các doanh nghiệp.

Yêu cầu 13 : Hiện thực hàm deleteEnterprise(int map[2][15], int enterpriseArray[30], char* abbre, int booth, int* totalEmpty)

- Tên hàm: deleteEnterprise
- Tham số đâu vào:

  - map[2][15]: Mảng trạng thái gian hàng, mỗi phần tử biểu diễn trạng thái của một gian hàng (0: trống, 1: đã đăng ký).
  - enterpriseArray[30]: Mảng chứa thông tin về các doanh nghiệp đã đăng ký.
  - abbre: Tên viết tắt của doanh nghiệp cần thay đổi gian hàng.
  - booth: Chỉ số gian hàng hiện tại mà doanh nghiệp đã đăng ký.
- Trả về: Tổng số gian hàng còn đang trống và gán vào *totalEmpty*
- Yêu cầu: Thực thi hành động xoá gian hàng như mô tả

3.8 Xử lý câu lệnh đầu vào

Đây là hàm tổng quát để xử lý và gọi đến tương ứng hàm xử lý của câu lệnh khi người dùng sử dụng.

Yêu cầu 14 : Hiện thực hàm handleCommand(char* command, int map[2][15], Enterprise enterpriseArray[30], CommandType* commandType)

- Tên hàm: handleCommand
- Tham số đầu vào:
  - command: Câu lệnh do người dùng gọi
  - map[2][15]: Mảng trạng thái gian hàng, mỗi phần tử biểu diễn trạng thái của một gian hàng (0: trống, 1: đã đăng ký).
  - enterpriseArray[30]: Mảng chứa thông tin về các doanh nghiệp đã đăng ký.
  - commandType: Kiểu câu lệnh sau khi được xác định và gán vào
- Trả về: Kiểu câu lệnh người dùng gọi và gán vào *commandType*. Xử lý và trả về *map* và *enterpriseArray* tương ứng.
- Yêu cầu: Xử lý chuỗi để xác định kiểu câu lệnh và gọi đến hàm tương ứng.

3.9 Ba lô sinh viên (Knapsack)

Giả sử mỗi sinh viên tham gia có một ba lô để chứa các vật phẩm khi tham gia tại mỗi gian hàng được tặng. Mỗi balo có một sức chứa tối đa, tổng khối lượng chứa lớn nhất có thể là 15. Ở mỗi gian hàng có một loại vật phẩm với một khối lượng và một giá trị, khối lượng cho mỗi vật phẩm là số nguyên nằm trong đoạn [1,10], giá trị của vật phẩm là số nguyên nằm trong đoạn [-5,5].

Bạn sinh viên cần đi qua các gian hàng và thu thập các vật phẩm tối ưu nhất tức là sao

cho tổng giá trị đạt tối đa và đảm bảo khối lượng không vượt quá sức chứa của ba lô và ít nhất có thể.

Yêu cầu 15 : Hiện thực hàm knapsack(Enterprise* enterpriseArray, int maxWeight, int numOfEnterprises, int index)

- Tên hàm: knapsack
- Tham số đầu vào:
  - map[2][15]: Mảng trạng thái gian hàng, mỗi phần tử biểu diễn trạng thái của một gian hàng (0: trống, 1: đã đăng ký).
  - enterpriseArray[30]: Mảng chứa thông tin về các doanh nghiệp đã đăng ký.
  - maxWeight: Khối lượng tối đa của ba lô sinh viên
  - numOfEnterprises: Số lượng doanh nghiệp đang có trong mảng enterpriseArray
  - index: Chỉ số bắt đầu duyệt qua của mảng enterpriseArray, mặc định luôn bắt đầu từ 0 (dùng để phục vụ cho đệ quy)
- Trả về: Tổng giá trị tối đa mà sinh viên có thể thu thập.
- Yêu cầu: Hiện thực hàm để giúp sinh viên tìm được một danh sách các gian hàng có thể tham gia để thu thập được giá trị vật phẩm cao nhất có thể.

Ví dụ 5: Giả sử các thông tin đầu vào:

- Các gian hàng đã đăng kí bao gồm: 0,1,3,5,7,13,14
- Giá trị vật phẩm tại từng gian hàng tương ứng là: 3,-2,4,5,-1,2,5
- Khối lượng vật phẩm tại từng gian hàng tương ứng là: 3,1,4,7,8,2,1
- Khối lượng tối đa của ba lô là 10

Lựa chọn phù hợp nhất và tối ưu nhất là các gian hàng: 0, 3, 14 với:

- Tổng giá trị: 12
- Tổng khối lượng: 8

Lựa chọn tổ hợp gian hàng {5, 13, 14} cũng mang lại giá trị 12, lớn nhất tuy nhiên tổng khối lượng lúc đó là 10. Tuy không vượt quá giới hạn tối đa của ba lô nhưng không phải là phương án tối ưu nhất nên không phải lựa chọn

Gợi ý: Nghiên cứu về bài toán Knapsack.

4 Nộp bài

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

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

File jobfair.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 LMS 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 mô tả hàm của yêu cầu 15 (01/11)
- Cập nhật mô tả hàm, thêm tham số cho yêu cầu 6, 13, 15 (03/11)
- Cập nhật tham số đầu vào hàm yêu cầu 7 (08/11)
- Cập nhật tham số đầu vào hàm yêu cầu 8, 10 (09/11)
- Cập nhật ví dụ phần 3.4 (10/11)

——————CHÚC CÁC BẠN LÀM BÀI THẬT TỐT——————