Roter Text kennzeichnet Ihre Eingaben in der Konsole, Ihre Eingaben werden nicht tatsächlich rot sein.
Weißer Text kennzeichnet die Ausgaben Ihres Programms.

Aufgabe

Schreiben Sie ein Programm das es dem Benutzer ermöglicht einen Zug, Waggon für Waggon, zusammen zu stellen. Dabei hat jeder Waggon einen von drei vordefinierten Typen (Passagierwagen, Schlafwagen und Speisewagen) und eine Kapazität (von 20 bis 130 Plätze). Ein Zug besteht aus maximal 10 Waggons. Jeder Waggon hat eine eindeutige Position innerhalb des Zuges.

Ihr Programm soll ein Menü ausgeben, um folgende Aktionen beliebig oft durchzuführen:

1. den Zug mit allen Waggons anzeigen (**print train**)
2. einen neuen Waggon erfassen und an einer bestimmten Position in den Zug einfügen (**new carriage**)
3. eine Statistik zum Zug ausgeben (**print stats**)
4. das Programm beenden (**exit**)

Beispiel Aus- und Eingaben:

```
Choose Action: print train (p), new carriage (n), print stats (s) or exit (x): p

Train: No Carriages!
Choose Action: print train (p), new carriage (n), print stats (s) or exit (x): n

Choose type of carriage: passenger (p), sleeper (s) or diner (d): p

Choose capacity (20 - 130): 20

Choose position for the new carriage (0-0): 0

Choose Action: print train (p), new carriage (n), print stats (s) or exit (x): n

Choose type of carriage: passenger (p), sleeper (s) or diner (d): p

Choose capacity (20 - 130): 10

Input invalid! Try again: 20

Choose position for the new carriage (0-1): 0

Choose Action: print train (p), new carriage (n), print stats (s) or exit (x): n

Choose type of carriage: passenger (p), sleeper (s) or diner (d): e

Input invalid! Try again: p

Choose capacity (20 - 130): 20
```

```
Choose position for the new carriage (0-2): 2

Choose Action: print train (p), new carriage (n), print stats (s) or exit (x): n

Choose type of carriage: passenger (p), sleeper (s) or diner (d): p

Choose capacity (20 - 130): 20

Choose position for the new carriage (0-3): 8

Input invalid! Try again: 1

Choose Action: print train (p), new carriage (n), print stats (s) or exit (x): n

Choose type of carriage: passenger (p), sleeper (s) or diner (d): s

Choose capacity (20 - 130): 50

Choose position for the new carriage (0-4): 2

Choose Action: print train (p), new carriage (n), print stats (s) or exit (x): p

Train: 0:[P:020]-1:[P:020]-2:[S:050]-3:[P:020]-4:[P:020] Length: 5
Choose Action: print train (p), new carriage (n), print stats (s) or exit (x): n

Choose type of carriage: passenger (p), sleeper (s) or diner (d): d

Choose capacity (20 - 130): 50

Choose position for the new carriage (0-5): 1
```

```
Choose Action: print train (p), new carriage (n), print stats (s) or exit (x): s

Train: 0:[P:020]-1:[D:050]-2:[P:020]-3:[S:050]-4:[P:020]-5:[P:020] Length: 6
Capacities:
 Passenger: 80
 Sleeper: 50
 Diner: 50
Choose Action: print train (p), new carriage (n), print stats (s) or exit (x): x
```

Allgemein zur Formatierung:
In CodeRunner wird jede einzelne Eingabe von einem Zeilenumbruch bestätigt, der in der Ausgabe aber nicht sichtbar ist. Wenn Sie außerhalb von CodeRunner das Programm starten, selber Werte eingeben (siehe Beispielausgabe) und dabei Zeilenumbrüche produzieren, wirkt sich das auf die Ausgabe aus (mehr Zeilenumbrüche als beim automatischen Testen in CodeRunner).

Allgemein zu Ausgaben:
Geben Sie Zeilenumbrüche immer vor einer neue Zeile aus und nicht am Ende. Wenn Sie sich daran halten, sollte die Formatierung leichter zur Übereinstimmung mit der erwarteten Ausgabe gebracht werden können. Das Programm startet daher auch mit einem Zeilenumbruch.

Lesen Sie die gesamte Angabe bevor Sie mit der Implementierung starten.

Folgende Bedingungen **muss** ihr Programm erfüllen:

- Verwenden Sie eine Enumeration zur Definition des Typs (type) eines Waggons:
  - Passagierwagen (`passenger`)
  - Schlafwagen (`sleeper`)
  - Speisewagen (`diner`)
- Verwenden Sie eine Struktur um einen Waggon (`carriage`) zu speichern:
  - Enumeration Typ (`type`)
  - Integer Kapazität (`capacity`)
- Verwenden Sie eine weitere Struktur um einen Zug (`train`) zu speichern:
  - Feld von Waggons (`carriages`) für maximal 10 Waggons
  - Integer Zuglänge (`length`), entspricht Anzahl der Waggons

Schreiben Sie folgende Funktionen zur Ausgabe:

- `void printCarriage(struct carriage)`: Die Funktion gibt den als Parameter übergebenen Waggon in folgendem Format aus: "[<Typ>:<Kapazität>]". Dabei soll der Typ Passagierwagen mit 'P', Schlafwagen mit 'S' und Speisewagen mit 'D' angezeigt werden. Die Kapazität soll dreistellig und mit führenden Nullen angezeigt werden. Die Funktion soll keinen Zeilenumbruch ausgeben.

```
[P:040]
```

- `void printTrain(struct train *)`: Die Funktion gibt den als Parameter übergebenen Zug (Vorsicht: Zeiger) mit allen Waggons in folgendem Format aus: "Train: <Position>:[<Typ>:<Kapazität>]-<Position>:[<Typ>:<Kapazität>]-... Length: <Zuglänge>". Die Position entspricht dabei dem Index des Waggons im Feld. Verwenden Sie zur Ausgabe der Waggons die Funktion zur Ausgabe eines Waggons.

```
Train: 0:[P:020]-1:[S:020] Length: 2
```

- Wenn der Zug keine Waggons hat, muss "Train: No Carriages!" ausgegeben werden.

```
Train: No Carriages!
```

Allgemein zu Eingaben:
Leerzeichen und Zeilenumbrüche sollen ignoriert werden. Wenn etwas Ungültiges eingegeben wurde, muss "Input invalid! Try again: " ausgegeben und erneut eingelesen werden. Sie können sich aber darauf verlassen, dass Zeichen eingegeben werden, wenn Zeichen erwartet werden und Zahlen eingegeben werden, wenn Zahlen erwartet werden.

Schreiben Sie folgende Funktionen zur Eingabe:

- `char getMenu()`: Die Funktion erfragt welche Aktion ausgeführt werden soll und liefert ein entsprechendes Zeichen zurück.

```
Choose Action: print train (p), new carriage (n), print stats (s) or exit (x): w

Input invalid! Try again: x
```

- `struct carriage getCarriage()`: Die Funktion erfragt den Waggontyp und die Kapazität des Waggons und liefert einen Waggon mit entsprechenden Werten zurück.

```
Choose type of carriage: passenger (p), sleeper (s) or diner (d): n

Input invalid! Try again: p

Choose capacity (20 - 130): -10

Input invalid! Try again: 131

Input invalid! Try again: 130
```

- `int getPosition(int)`: Die Funktion erhält die Zuglänge als Parameter, erfragt die Position für einen neu einzuhängenden Wagon und liefert diese zurück. Beispiel: Bei Zuglänge 2 gibt es daher 3 gültige Positionen: 0, 1, 2.

```
Choose position for the new carriage (0-2): 3

Input invalid! Try again: 0
```

Schreiben Sie folgende Funktion zum Einfügen eines Waggons in den Zug:

- `int insertCarriage(struct train *, int, struct carriage)`: Die Funktion erhält einen Zeiger auf den Zug, eine Position an welcher der neue Waggon eingefügt werden soll und den einzufügenden Waggon. Wenn 0 als Position übergeben wird, soll der Waggon am Anfang eingefügt werden. Die Funktion muss bei Erfolg

eingefügt werden. Die Funktion muss bei Erfolg 0 und bei Fehler einen negativen Fehlercode zurückliefern: -1 wenn der Zug bereits voll ist und -2 wenn die Position nicht gültig ist. Ein Speisewagen darf nur zwischen zwei Passagierwagen eingefügt werden. Falls diese Bedingung nicht erfüllt ist, muss -3 als Fehlercode zurückgeliefert werden.

- Mögliches Vorgehen beim Einfügen: Um einen neuen Waggon einzufügen, kann das Feld vom Ende bis zur übergebenen Position durchlaufen werden. Dabei wird zuerst der letzte Waggon um eins nach rechts verschoben, danach der Vorletzte, usw. bis die Position erreicht ist, an der der neue Waggon eingefügt werden soll. Nachdem auch dieser Waggon verschoben wurde und die Position nun frei ist, kann der neue Waggon eingefügt werden.
- In der aufrufenden Funktion main muss dann, abhängig vom Fehlercode, eine Fehlermeldung ausgegeben werden:
  - Bei -1 Ausgabe:
    Error: Train too long!
  - Bei -2 Ausgabe:
    Error: Position not possible!
  - Bei -3 Ausgabe:
    Error: Diner only possible between two passenger carriages!

Schreiben Sie folgende Funktionen zum Ermitteln der Kapazitäten und zur Ausgabe der Statistik:

- `int sumCapacity(struct train *, enum type)`: Die Funktion erhält einen Zeiger auf den Zug und den Typ eines Waggons und liefert die Summe der Kapazitäten aller Waggons des Zuges die diesen Typ haben zurück.
- `void printTrainStats(struct train *)`: Die Funktion erhält einen Zeiger auf den Zug und gibt den Zug und eine Statistik zu den Kapazitäten der drei Waggontypen formatiert aus. Zur Ausgabe des Zuges und zur Ermittlung der Kapazitätssummen nutzen Sie die Funktionen printTrain und sumCapacity. Nach der Ausgabe des Zuges folgt eine Zeile mit "Capacities:" und dann eine Zeile pro Waggontyp (Einrückung 2 Leerzeichen):
  - für Passagierwagen: " Passenger: <Summe>"
  - für Speisewagen: " Diner: <Summe>"
- Falls die Kapazität von Plätzen in den Speisewagen größer als 60 ist, soll hinter der entsprechenden Summe " - invalid" ausgegeben werden.

```
Train: 0:[P:020]-1:[D:030]-2:[P:020]-3:[S:050]-4:[P:020]-5:[D:031]-6:[P:020] Length: 7
Capacities:
  Passenger: 80
  Sleeper: 50
  Diner: 61 - invalid
```

Alle Funktionsdefinitionen (Funktionsname, Parameteranzahl, Typen der Parameter) müssen der Angabe entsprechen.
Parameternamen die in der Angabe in Klammern angegeben sind, sind Vorschläge und können verändert werden.

Das Verwenden von globalen Variablen ist nicht erlaubt!

Hinweise

- Setzen Sie die Angabe um und implementieren Sie nicht anhand der Testfälle.
- Nutzen Sie eine lokale Entwicklungsumgebung und testen Sie Ihr Programm selbst, bevor Sie CodeRunner nutzen.
- Es empfiehlt sich, zuerst die Enumeration und die Strukturen zu definieren, dann die zwei Funktionen zur Ausgabe von Waggon und Zug zu implementieren, zu testen und erst danach den Rest zu implementieren. Dadurch können Sie die ausgebenden Funktionen zum Debuggen benutzen und sich anzeigen lassen, ob das Verschieben der Waggons beim Einfügen wie gewünscht funktioniert. Dazu geben Sie vorübergehend in der Ausgabefunktion einen Waggon mehr oder auch alle 10 Waggons aus.
- Nutzen Sie bereits implementierte Funktionen und vermeiden Sie redundanten Code.