

目录




1. Week 2 Exercises
 1. opal.c
 2. sumnum.c
 3. count+.c
 4. recount+.c
 5. recount.c

Week 2 Exercises

opal.c

1.

- a. Define a data type to store all the information of a single ride with the Opal card. Here are 3 sample records.

Transaction number	Date/time	Mode	Details	Journey number	Fare Applied	Fare	Discount	Amount
2013	Mon 30/07/2018 10:16		Flinders St af Oxford St to Anzac Pde D opp UNSW	1		\$1.46	\$0.00	-\$1.46
2011	Mon 30/07/2018 10:05		Victoria St at Liverpool to Oxford St op Palmer St	1		\$2.20	\$0.00	-\$2.20
2009	Sun 29/07/2018 17:35		Bondi Junction to Kings Cross		Day Cap	\$3.54	\$3.54	\$0.00

You may assume that individual stops (such as *Anzac Pde D opp UNSW*) require no more than 31 characters.

- b. Write a C program that defines a variable that has this type, and print the total number of bytes this variable occupies in memory. The output should be:

```
The Opal data structure requires ??? bytes.
```

where ??? is the total number of bytes.

- c. Explain how sum of the data fields in your data structure lead to the total.
d. If you want to store millions of records, how would you improve your data structure?

sumnum.c

Write a program that sums the arguments on the command line and prints the result on *stdout*. If there is no argument, the program generates no output. Examples of the program executing are:

```
prompt$ ./sumnum 1 2 3
6

prompt$ ./sumnum 123
123

prompt$ ./sumnum
```

(You may assume that each argument is well-formed: i.e. it is either numeric, or alphabetic.) You

should use a *Makefile* in this exercise to build the executable.

Make the program robust by generating an error message if one of the arguments is non-numeric. For example,

```
prompt$ ./sumnum 1 fred 2
error: argument fred is non-numeric
```

count+.c

Write a program that prints on *stdout* a sequence of numbers from 0 to a non-negative number given on the command line. The numbers in the sequence are separated by commas. Examples of the program executing are:

```
prompt$ ./count+ 1
0,1

prompt$ ./count+ 10
0,1,2,3,4,5,6,7,8,9,10

prompt$ ./count+ 0
0
```

You may assume that the argument is a non-negative integer, and the output appears on one line. You should extend the previous *Makefile* to build the executable.

Make your program robust by handling the following exceptions: if there are no arguments on the command line, or there is more than 1 argument, then a 'usage' message should be output. For example,

```
prompt$ ./count+
Usage: ./count+ number

prompt$ ./count+ 1 2
Usage: ./count+ number
```

The output of your program should match the above exactly.

recount+.c

Modify the program above to use recursion instead of iteration. The usage is precisely the same as above (but the name of the executable is different of course). Extend the *Makefile* you used above to make the executable.

recount.c

Now modify the recursive program to count backwards as well as forwards. The direction of counting is determined by the sign of the command-line argument. You should use only one recursive function in the program. Again, extend the *Makefile* you used above to make the executable. An example of counting backwards is:

```
prompt$ ./recount -12
0,-1,-2,-3,-4,-5,-6,-7,-8,-9,-10,-11,-12
```

Week2Exercises (2019-06-12 09:30:40由AlbertNymeyer编辑)