

Ternacci

CS1010 Practical Exam 0 Practice

| Background and Task

We are going to compute the result of “Ternacci”, similar to the well known “Fibonacci” sequence. The formula is:

$$\begin{aligned} \text{Ternacci}(n) &= 0 \text{ if } n = 0 \\ &0 \text{ if } n = 1 \\ &1 \text{ if } n = 2 \\ &\text{Ternacci}(n-1) + \text{Ternacci}(n-2) + \text{Ternacci}(n-3) \text{ if } n \geq 3 \end{aligned}$$

Your Task: Given as input a non-negative integer n , your task is to return $\text{Ternacci}(n)$.

| Code

You are given the following empty function(s) in `student.c` which you should implement. Only make changes within `student.c`.

```
unsigned long ternacci(unsigned long n) {  
}
```

Implement all your code in `student.c`. Use recursion. Do not use loops, arrays, global variables, pointers, dynamic allocation, or function-local static variables (ignore these if you don't know what they are). You may add helper functions in `student.c`.

To compile, run `make`, to test, run `make test`, and to manually run, do `./ternacci`. Enter one integer n , where $0 \leq n \leq 20$, then press Enter.

Note: We will handle all input reading and outputting for you.

| Examples

Input	Output
0	0
1	0
2	1
10	81
20	35890

WizzBuzz

CS1010 Practical Exam 0 Practice

| Background

The “CS1010 Wizard” is casting a spell. He counts from 1 to N , printing out the numbers, but sometimes replaces the numbers with words according to the table below. He ends the spell with “Abracadabra”.

Condition	What to say
Divisible by 2	Hocus
Divisible by 5	Pocus
Divisible by both 2 and 5	HocusPocus
Otherwise	The number itself

| Your Task

Implement the following function in `student.c` using recursion:

```
void wizzbuzz(int n);
```

Print the spell from 1 up to n , followed by Abracadabra. Print a newline after every integer or phrase. For $n = 0$, the only output is Abracadabra followed by a newline. Input reading is handled for you.

Only modify `student.c`. You must not use loops, arrays, global variables, pointers, dynamic allocation, or function-local static variables in your solution (ignore if you don’t know what these are). You may add recursive helper functions in `student.c`.

| Input and Output

Enter one integer N , where $0 \leq N \leq 1000$, then press Enter. Print the spell as described above.

| Example

Sample Input:

10

Expected Output:

1
Hocus
3
Hocus
Pocus
Hocus
7
Hocus
9
HocusPocus
Abracadabra

| Running and Testing

- Compile: `make`
- Run: `./wizzbuzz`
- Test: `make test`

Parcel Quote

CS1010 Practical Exam 0 Practice

| Background

A courier offers standard ('S') and express ('E') delivery. Help it calculate the postage for a parcel.

All prices below are in **cents**. The rules are:

- A parcel heavier than 5000 grams costs **1800 cents**, regardless of service. No other charges apply.
- Otherwise, the first 1000 grams cost **400 cents**. Each additional 1000 grams, or part thereof, costs **200 cents**.
- For parcels weighing at most 5000 grams, express delivery adds **300 cents** to the weight-based charge.

For example, a 1001-gram parcel is $400 + 200 = 600$ cents with standard delivery, while a 2000-gram parcel with standard delivery is also 600 cents.

| Your Task

Implement the following function in `student.c`:

```
unsigned int quote_postage(unsigned int weight, char service);
```

Return the postage in cents. Use integer arithmetic. Only modify `student.c`. Do not use loops, arrays, global variables, pointers, dynamic allocation, or function-local static variables (ignore if you don't know what these are).

| Input and Output

Enter the weight in grams and the service character, separated by a space, then press Enter. The weight is from 1 to 10000 inclusive, and the service is always 'S' or 'E'.

Input reading and printing are handled for you. The driver prints the returned price as an integer, followed by a newline.

| Examples

Input	Output
1000 S	400
1001 S	600
2000 E	900
5000 E	1500
5001 E	1800

The last parcel costs 1800 cents because it is heavier than 5000 grams. There is no express surcharge in this case.

| Running and Testing

- Compile: `make`
- Run: `./parcel-quote`
- Test: `make test`

Target Temperature

CS1010 Practical Exam 0 Practice

| Background

A lab needs a system to pick a DNA sample that is closest to a target temperature. It will measure samples one at a time, and you will have to return the **sample number** (which starts from 1) and **sample temperature** of the one that's closest to the target. If two samples are equally close, prefer the cooler one. If their temperatures are identical, choose the first sample measured.

| Your Task

In `student.c`, define the fields of `State` and implement:

```
State first_sample(State state, int target, int temperature);
State next_sample(State state, int temperature);
void finish_samples(State state);
```

Our code calls `first_sample` once with the target temperature and the first sample's temperature. It then calls `next_sample` for each remaining sample in order, passing the state returned by the previous call (similar to your Volumes). Finally, it calls `finish_samples` to print the selected sample number and temperature. Note again that sample numbers start at 1.

Replace the placeholder fields of `State` with the scalar numerical fields you need. Initialise all your fields in `first_sample`. Only modify `student.c`. Do not use loops, arrays, global variables, pointers, dynamic allocation, or function-local static variables (ignore if you don't know what these are).

| Input and Output

The first line contains the target temperature and the number of samples, from 1 to 1000. The second line contains that many sample temperatures. All temperatures, including the target, are integers from -100 to 200 inclusive, measured in degrees Celsius.

Input reading is handled for you. Print two lines using the labels below, with a newline after each line.

| Examples

Input	Output
50 4 42 61 54 46	Sample: 4 Temperature: 46
20 3 18 18 23	Sample: 1 Temperature: 18

In the first example, the target temperature is 50 degrees, and there are going to be 4 samples. Since the samples at 54 and 46 degrees are equally close to the target 50 degrees, we choose 46. In the second example, both samples at 18 are equally suitable, so we choose the first.

| Running and Testing

- Compile: `make`
- Run: `./target-temperature`
- Test: `make test`