Emoji Grades

Tutorial · CC3036 · 2021

Diogo Peralta Cordeiro

Separate representation from meaning

An emoji is not necessarily one byte, one Unicode code point, or one visible cell. The task becomes simpler when those notions are kept separate. Read bytes, decode Unicode scalar values, map the permitted symbols to ASCII, and only then parse names and grades.

The output uses the literal Portuguese header `NOME  NOTA` and closing sentence specified in the statement, even when you are reading the English version. They are judge data, not interface labels to translate.

Decode UTF-8 explicitly

Use `unsigned char` for input bytes. A signed `char` can turn a byte above 127 into a negative value, making bit tests unreliable.

Leading byte	Bytes that follow	Initial payload
`00–7F`	0	The byte itself
`C2–DF`	1	Lowest 5 bits
`E0–EF`	2	Lowest 4 bits
`F0–F4`	3	Lowest 3 bits

Every continuation byte must satisfy `(byte & 0xC0) == 0x80`. Accumulate each six-bit payload with:

```
cp = (cp << 6) | (byte & 0x3F);
```

For two-, three- and four-byte sequences, the minimum decoded values are `0x80`, `0x800` and `0x10000`. Reject values below those minima, values above `0x10FFFF`, and the surrogate range `0xD800–0xDFFF`. These checks reject overlong encodings and invalid scalar values. A sequence that ends before all continuation bytes arrive is invalid too.

For example, bytes `F0 9F 87 B3` decode to U+1F1F3, regional indicator N. The symbol's appearance is irrelevant once this integer value is known.

Map symbols and consume their suffixes

Regional indicators map directly to the alphabet:

```
letter = 'A' + cp - 0x1F1E6;
```

Handle the five special letter symbols with a small switch table. After one of those symbols, consume U+FE0F if present. Regional indicators do not accept that selector in this problem.

For digits, accept a plain ASCII digit, a digit followed by U+20E3, or a digit followed by U+FE0F and U+20E3. A selector after a digit must have the keycap code point after it. A stray selector or keycap is not independently meaningful.

A useful invariant is: **each successful conversion consumes a complete allowed symbol and emits exactly one ASCII character**. Presentation selectors and keycap marks emit nothing of their own. The solution first decodes a line to code points, then walks that array with lookahead to apply these rules.

Parse the decoded sheet

Remove CR only when it precedes an LF terminator. An unterminated final line is accepted, but an isolated CR is not a line terminator. Decode the line, then remove permitted trailing ASCII spaces.

Names can contain single spaces, while the name/grade separator contains at least two. This makes the first run of two or more spaces an unambiguous boundary. Splitting on every space would incorrectly divide names such as `ANA MARIA`.

The first non-empty line must decode to the two header fields. For student rows, validate the name length and alphabet, separator length and grade range. Read the grade as an integer so `07` prints as 7.

Track whether blank lines have started after the sheet. Once that happens, a later non-empty line is invalid; blank lines cannot occur between students. Empty lines before the header are harmless. The reference implementation validates before printing and sends diagnostics to standard error. Invalid input is outside the judge's promised data, so that diagnostic is not part of the required answer.

Buffer the rows and align the table

The final width depends on the longest decoded name, which may occur in the last row. Store each name and integer grade while updating $W = \max(4, \text{all decoded name lengths})$.

The output names are ASCII, so `strlen` now measures their character count correctly. Measuring the raw emoji bytes instead would produce the wrong width.

For `ANA` and `LUIS`, $W = 4$. The row for ANA needs one padding space followed by two separator spaces:

```
NOME  NOTA
ANA   20
LUIS  7
Cumprimentos algorítmicos!
```

In C, the negative field-alignment flag in `%-*s` provides the padding:

```c
printf("%-*s  NOTA\n", (int)width, "NOME");
printf("%-*s  %d\n", (int)width, name, grade);
```

Do not pad after the grade. Emit LF after every line, including the closing sentence.

Complexity

Let B be the number of input bytes, O the number of output bytes, N the number of students and L the maximum name length. UTF-8 decoding, mapping and parsing make a bounded number of passes over each line. Total time is $O(B + O)$ and storage is $O(NL)$ plus one bounded input line. The supplied solution supports 10000 students and 80-character names with fixed-size buffers.

The seed 42 does not need to be reversed. It affects the converter's choices and padding, which the statement already defines how to interpret. The test generator derives expected output independently from its plain-text student records.

Build and check

```
cc -std=c17 -O2 -Wall -Wextra -Werror -pedantic \
  solution.c -o notas-emoji
./notas-emoji < samples/input.txt > actual.txt
diff -u samples/output.txt actual.txt
python3 test.py
```

Generate a larger case:

```
python3 generate.py --seed 42 --count 10000 \
  --varied --output generated
./notas-emoji < generated/input.txt > actual.txt
diff -u generated/output.txt actual.txt
```

Check every letter, both keycap forms, optional presentation selectors, grades 0 and 20, multiword and 80-character names, CRLF, no final newline and blank margin lines. Malformed UTF-8 and unsupported selector positions are useful defensive tests. For the supplied twenty-student sheet, compare against `samples/classroom-output.txt`.