Contents

1 Specification **2**
1.1 Memory 2
1.2 Instructions 2

2 Examples **3**
2.1 Descending Numbers 3
2.2 Hello World 3
2.3 Puts 3

3 C++ Reference Implementation **4**

1 Specification

1.1 Memory

The program instructions and data are stored in an 2-dimensional matrix of arbitrary size. The matrix is indexed by 2-dimensional pointers. The instruction pointer (`ip`) automatically increments in the current direction. The data pointer (`dp`) is manually incremented. The direction of the instruction pointer is also a 2-dimensional value and will be referenced as `d`. All values in the program memory are 8-bit words (equivalent to unsigned 8-bit integers).

There are 3 8-bit registers available:

- A is the accumulator register.
- B is a secondary register.
- C is a special register immune to the effects of time travel.

A 'push' refers to B being set to A, followed by A being set to some input value. Along with the `C` register, `stdout` and `stdin` are unaffected by time travel. All uninitialised values are set to 0 and the direction is set to right.

1.2 Instructions

Instruction	Description
>	Set `d` to right. If `d` was already right, move `dp` right by `A`.
<	Set `d` to left. If `d` was already left, move `dp` left by `A`.
v	Set `d` to down. If `d` was already down, move `dp` down by `A`.
^	Set `d` to up. If `d` was already up, move `dp` up by `A`.
0	Push 48.
1	Push 49.
2	Push 50.
3	Push 51.
4	Push 52.
5	Push 53.
6	Push 54.
7	Push 55.
8	Push 56.
9	Push 57.
+	Add `A` and `B`, push the result.
=	Compare `A` to 0. The result is 1 if equal, 0 otherwise. Store result in `A`.
–	Swap `A` and `B`.
:	Push `A`.
\	Copy `A` to `C`.
/	Copy `C` to `A`.
%	Print `A` as ASCII (`stdout`).
$	Set `A` to next input character (`stdin`).
.	Read value at `dp` to `A`.
,	Write value `A` to `dp`.
~	Time travel `A` instructions into the past.
#	Time travel `A` instructions into the past if `B` is zero.
*	Jump `A` instructions after the next ! in the current direction.
?	Jump `A` instructions after the next ! in the current direction if `B` is zero.
!	End program.

All other values are ignored by the interpreter and can be treated as comments or data values.

2 Examples

2.1 Descending Numbers

Print 0-9 in descending order.

```
1 6:+9+3+2+\/7+8+0+0+0+\0+%9:+:+6+−/=−#6:+9+3+2+%!
```

Output: `9876543210`

2.2 Hello World

Print "Hello World!".

```
1 8:+:+4+4+%23+%4:+:+\0−/+\66+%7:+:+7+−/−#96+%0:+:+0+0+%1:+:+1+1+1+%96+% ↙
    ↳ 9:+%66+%2:+%0:+:+0+1+%6:+9+3+2+%!
```

Output: `Hello World!`

2.3 Puts

Similar to the `puts` function in C, but uses a backtick (`` ` ``) as the terminator.

```
1 v                                input text here:The quick brown fox jumped ↙
    ↳ over the lazy dog.`
2 >0>=\4:+:+1+−/+\>.6+6+4+==:=?.%0:+:+0+3+~!6:+9+3+2+%!
```

Output: `The quick brown fox jumped over the lazy dog.`

3 C++ Reference Implementation

```cpp
#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>

#include <unordered_map>
#include <vector>

std::unordered_map<uint32_t, std::vector<uint8_t>> program;

typedef struct pointer
{
    uint32_t x;
    uint32_t y;
} pointer_t;

struct command;

typedef struct state
{
    pointer_t ip, dp, dir;
    uint8_t A, B, C;
} state_t;

typedef struct command
{
    state_t state;
    void (*undo)(struct command);
    uint8_t extra;
} command_t;

std::vector<command_t> commands;

static void push_command(command_t command)
{
    commands.push_back(command);
}

static void undo_command(void)
{
    assert(!commands.empty());
    command_t undo = commands.back();
    commands.pop_back();
    undo.undo(undo);
}

static void program_write(pointer_t p, uint8_t data)
{
    auto& line = program[p.y];
    if(line.empty() || line.size() - 1 < p.x) line.resize(p.x + 1);
    line[p.x] = data;
}

static uint8_t program_read(pointer_t p)
{
    if(program.count(p.y))
    {
        auto& line = program[p.y];
        if(line.size() > p.x) return line[p.x];
```

```
60      }
61      return 0;
62  }
63
64  static void init(const char* filename)
65  {
66      FILE* in = fopen(filename, "r");
67      if(!in) exit(1);
68
69      fseek(in, 0, SEEK_END);
70      size_t size = ftell(in);
71      fseek(in, 0, SEEK_SET);
72
73      uint8_t* programRaw = (uint8_t*)calloc(size, 1);
74      if(fread(programRaw, 1, size, in) != size) exit(1);
75      fclose(in);
76
77      pointer_t pos = {0, 0};
78      for(size_t i = 0; i < size; ++i)
79      {
80          if(programRaw[i] == '\n')
81          {
82              ++pos.y;
83              pos.x = 0;
84              continue;
85          }
86
87          program_write(pos, programRaw[i]);
88          ++pos.x;
89      }
90      free(programRaw);
91  }
92
93  state_t state;
94
95  static void inc(void)
96  {
97      state.ip.x += state.dir.x;
98      state.ip.y += state.dir.y;
99  }
100
101 static void push_undo(void (*undo)(struct command), uint8_t extra = 0)
102 {
103     command_t command = (command_t){.state = state, .undo = undo, .extra = extra};
104     push_command(command);
105 }
106
107 static void simple_undo(command_t undo)
108 {
109     const uint8_t C = state.C;
110     state = undo.state;
111     state.C = C;
112 }
113
114 static void right(void)
115 {
116     push_undo(simple_undo);
117
118     if(state.dir.x == 1 && state.dir.y == 0) state.dp.x += state.A;
119     else state.dir = (pointer_t){1, 0};
120 }
```

```
121
122 static void left(void)
123 {
124     push_undo(simple_undo);
125
126     if(state.dir.x == -1 && state.dir.y == 0) state.dp.x -= state.A;
127     else state.dir = (pointer_t){(uint32_t)-1, 0};
128 }
129
130 static void down(void)
131 {
132     push_undo(simple_undo);
133
134     if(state.dir.x == 0 && state.dir.y == 1) state.dp.y += state.A;
135     else state.dir = (pointer_t){0, 1};
136 }
137
138 static void up(void)
139 {
140     push_undo(simple_undo);
141
142     if(state.dir.x == 0 && state.dir.y == -1) state.dp.y -= state.A;
143     else state.dir = (pointer_t){0, (uint32_t)-1};
144 }
145
146 static void push(uint8_t num)
147 {
148     push_undo(simple_undo);
149
150     state.B = state.A;
151     state.A = num;
152 }
153
154 static void add(void)
155 {
156     push(state.A + state.B);
157 }
158
159 static void cmp(void)
160 {
161     push_undo(simple_undo);
162
163     state.A = (state.A == 0);
164 }
165
166 static void dup(void)
167 {
168     push(state.A);
169 }
170
171 static void rotate(void)
172 {
173     push_undo(simple_undo);
174     const uint8_t tmp = state.B;
175     state.B = state.A;
176     state.A = tmp;
177 }
178
179 static void save(void)
180 {
181     state.C = state.A;
```

```
182     push_undo(simple_undo);
183 }
184
185 static void load(void)
186 {
187     push_undo(simple_undo);
188     state.A = state.C;
189 }
190
191 static void out(void)
192 {
193     push_undo(simple_undo);
194     putchar(state.A);
195 }
196
197 static void in(void)
198 {
199     push_undo(simple_undo);
200     state.A = getchar();
201 }
202
203 static void read(void)
204 {
205     push(program_read(state.dp));
206 }
207
208 static void undo_write(command_t undo)
209 {
210     program_write(undo.state.dp, undo.extra);
211     simple_undo(undo);
212 }
213
214 static void write(void)
215 {
216     push_undo(undo_write, program_read(state.dp));
217     program_write(state.dp, state.A);
218 }
219
220 static void travel(void)
221 {
222     const uint8_t loops = state.A;
223     for(int i = 0; i < loops; ++i)
224     {
225         undo_command();
226     }
227     push_undo(simple_undo);
228 }
229
230 static void travel_cond(void)
231 {
232     if(!state.B) travel();
233     else
234     {
235         push_undo(simple_undo);
236         inc();
237     }
238 }
239
240 static void jmp(void)
241 {
242     push_undo(simple_undo);
```

```
243     for(;;)
244     {
245         state.ip.x += state.dir.x;
246         state.ip.y += state.dir.y;
247         uint8_t instr = program_read(state.ip);
248         if(instr == '!') break;
249     }
250     state.ip.x += state.dir.x * state.A;
251     state.ip.y += state.dir.y * state.A;
252 }
253 
254 static void jmp_cond(void)
255 {
256     if(!state.B) jmp();
257     else
258     {
259         push_undo(simple_undo);
260         inc();
261     }
262 }
263 
264 int main(int argc, char** argv)
265 {
266     if(argc < 2) return 1;
267     init(argv[1]);
268     state.dir = {1, 0};
269 
270     for(;;)
271     {
272         uint8_t instr = program_read(state.ip);
273         switch(instr)
274         {
275             case '>': right(); break;
276             case '<': left(); break;
277             case 'v': down(); break;
278             case '^': up(); break;
279             case '0': case '1': case '2': case '3': case '4': case '5': case '6': case '7':
                ↪  case '8': case '9':
280                 push(instr - '0' + 48);
281                 break;
282             case '+': add(); break;
283             case '=': cmp(); break;
284             case '-': rotate(); break;
285             case ':': dup(); break;
286             case '\\': save(); break;
287             case '/': load(); break;
288             case '%': out(); break;
289             case '$': in(); break;
290             case '.': read(); break;
291             case ',': write(); break;
292             case '~': travel(); continue;
293             case '#': travel_cond(); continue;
294             case '*': jmp(); continue;
295             case '?': jmp_cond(); continue;
296             case '!': return 0;
297             default:
298             {
299                 push_undo(simple_undo);
300             }
301         }
302 
```

```
303             inc();
304         }
305     }
```