

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <conio.h>
4 #include <string.h>
5
6 short n_wide = 16;
7 short n_high = 16;
8 short s_to_n = 3;
9 short r_paths = 8;
10 short inputs = 10;
11 short outputs = 10;
12 int neurons;
13 int synapses;
14
15 int nets = 1000;
16
17 enum boolean {false, true};
18 typedef enum boolean {FALSE, TRUE} Boolean;
19
20 Boolean conn;
21
22 Boolean verbose=false;
23
24 int main(int argc, char *argv[]) {
25
26     FILE * nfile;
27
28     long lSize;
29     char * buffer;
30
31     char header_cmp[17] = {"ANN Routing File"};
32     char header[17];
33
34     char date[11];
35     char time[6];
36
37     long * neuron_in;
38     long * neuron_out;
39     long * synapse_in;
40     long * synapse_out;
41     long * input_out;
42     long * output_in;
43
44
45
46     char c;
47
48     int n;
49
50     long nets=0;
51
52     long fcount;
53
54     enum nodetype {neuron, synapse, input, output};
55     typedef enum nodetype {Neuron, Synapse, Input, Output} NodeType;
56
57     NodeType nodetypefrom;
58     NodeType nodetypeto;
59
60     long nodefrom;
61     long nodeto;
62
```

```
63 long netfrom;
64 long netto;
65
66 printf("-----\n");
67 printf(" A Dynamic Analog Concurrently-Processed Adaptive Chip\n");
68 printf("-----\n");
69 printf("Neural Network Router\n");
70 printf("By Malcolm Stagg\n\n");
71 if (argc < 2) {
72     printf("Please use format: nnrouter <input file name> [options]\n");
73     printf("options:\n");
74     printf("-v: verbose mode\n");
75     return 1;
76 } else {
77     printf("Input file: %s\n\n", argv[1]);
78
79     if (argc>2) {
80         for (n=2;n<argc;n++) {
81             if ((argv[n][0] == '-') || (argv[n][0] == '/')) {
82                 switch (argv[n][1]) {
83                     case 'v':
84                         verbose = true;
85                         break;
86                 }
87             }
88         }
89     }
90
91     if (verbose)
92         printf("Verbose ON\n\n");
93     else
94         printf("Verbose OFF\n\n");
95
96     nfile = fopen(argv[1], "rb");
97     if (nfile != NULL) {
98
99         /* obtain file size. */
100        fseek (nfile , 0 , SEEK_END);
101        lSize = ftell (nfile);
102        rewind (nfile);
103
104        /* allocate memory to contain the whole file. */
105        buffer = (char*) malloc (lSize);
106        if (buffer == NULL) exit (2);
107
108        /* copy the file into the buffer. */
109        fread (buffer,1,lSize,nfile);
110
111        fclose(nfile);
112
113        /* check if the header is valid */
114        for (n=0;n<16;n++) {
115            header[n] = buffer[n];
116        }
117        header[16]=0;
118
119        if (strcmp(header, header_cmp) == 0) {
120
121            /* header is valid */
122
123            /* read in date and time */
124
```

```
125     for (n=0;n<10;n++) {
126         date[n] = buffer[n+17];
127     }
128     date[10]=0;
129
130     for (n=0;n<5;n++) {
131         time[n] = buffer[n+28];
132     }
133     time[5]=0;
134
135     printf("File Date: %s\n", date);
136     printf("File Time: %s\n\n", time);
137
138     /* read in chip options */
139
140     fcount = 35;
141
142     while ((buffer[fcount] != 'X') && (fcount < strlen(buffer))) {
143         switch (buffer[fcount]) {
144             case 'W':
145                 n_wide = ((buffer[fcount+1]-'0')*10 +
146 (buffer[fcount+2]-'0'));
147                 fcount+=4;
148                 break;
149             case 'H':
150                 n_high = ((buffer[fcount+1]-'0')*10 +
151 (buffer[fcount+2]-'0'));
152                 fcount+=4;
153                 break;
154             case 'S':
155                 s_to_n = ((buffer[fcount+1]-'0')*10 +
156 (buffer[fcount+2]-'0'));
157                 fcount+=4;
158                 break;
159             case 'R':
160                 r_paths = ((buffer[fcount+1]-'0')*10 +
161 (buffer[fcount+2]-'0'));
162                 fcount+=4;
163                 break;
164             case 'I':
165                 inputs = ((buffer[fcount+1]-'0')*10 +
166 (buffer[fcount+2]-'0'));
167                 fcount+=4;
168                 break;
169             case 'O':
170                 outputs = ((buffer[fcount+1]-'0')*10 +
171 (buffer[fcount+2]-'0'));
172                 fcount+=4;
173                 break;
174             default:
175                 fcount++;
176         }
177     }
178
179     neurons = n_wide;
180     neurons *= n_high;
181
182     synapses = neurons;
183     synapses *= s_to_n;
184
185     if (verbose) {
186         printf("Width: ..... %d Neurons\n", n_wide);
187     }
```

```
181         printf("Height: ..... %d Neurons\n", n_high);
182         printf("Synapses per Neuron: %d\n", s_to_n);
183         printf("Routing Pathways: .. %d\n\n", r_paths);
184         printf("Inputs: %d\n", inputs);
185         printf("Outputs: %d\n\n", outputs);
186         printf("Total Neurons: %d\n", neurons);
187         printf("Total Synapses: %d\n\n", synapses);
188     }
189
190     fcount += 3;
191
192     /*
193
194     now the routing begins
195
196     I0000N0000 - connects input 0 to neuron 0's input
197     N9999O0000 - connects output 0 to neuron 9999's output
198     N1234S4321 - connects neuron 1234's output to synapse 4321's
input
199
200     I..input
201     O..output
202     N..neuron
203     S..synapse
204
205     order may be arbitrary
206     connections are separated by space. e.g.:
207
208     I0000N0000 N9999O0000 N1234S4321
209
210     */
211
212     printf("Reading In Netlist...\n");
213
214     neuron_in = (long*) malloc (neurons);
215     neuron_out = (long*) malloc (neurons);
216
217     synapse_in = (long*) malloc (synapses);
218     synapse_out = (long*) malloc (synapses);
219
220     input_out = (long*) malloc (inputs);
221     output_in = (long*) malloc (outputs);
222
223     /*
224     int neuron_in[neurons];
225     int neuron_out[neurons];
226     int synapse_in[synapses];
227     int synapse_out[synapses];
228     int input_out[inputs];
229     int output_in[outputs];
230     */
231
232     /* init connection arrays */
233
234     for (n=0;n<neurons;n++) {
235         neuron_in[n]=0;
236         neuron_out[n]=0;
237     }
238
239     for (n=0;n<synapses;n++) {
240         synapse_in[n]=0;
241         synapse_out[n]=0;
```

```
242     }
243
244     for (n=0;n<inputs;n++)
245         input_out[n]=0;
246
247     for (n=0;n<outputs;n++)
248         output_in[n]=0;
249
250
251     /* convert list of connections into nets */
252
253     while ((buffer[fcount] != 'X') && (fcount < strlen(buffer))) {
254         switch (buffer[fcount]) {
255             case 'I':
256                 /* connect an input to __ */
257                 nodetypefrom = input;
258                 nodefrom = ((buffer[fcount+1]-'0')*1000 +
259 (buffer[fcount+2]-'0')*100 + (buffer[fcount+3]-'0')*10 +
260 (buffer[fcount+4]-'0'));
261                 netfrom = input_out[nodefrom];
262                 fcount+=5;
263                 break;
264             case 'N':
265                 /* connect a neuron output to __ */
266                 nodetypefrom = neuron;
267                 nodefrom = ((buffer[fcount+1]-'0')*1000 +
268 (buffer[fcount+2]-'0')*100 + (buffer[fcount+3]-'0')*10 +
269 (buffer[fcount+4]-'0'));
270                 netfrom = neuron_out[nodefrom];
271                 fcount+=5;
272                 break;
273             case 'S':
274                 /* connect a synapse output to __ */
275                 nodetypefrom = synapse;
276                 nodefrom = ((buffer[fcount+1]-'0')*1000 +
277 (buffer[fcount+2]-'0')*100 + (buffer[fcount+3]-'0')*10 +
278 (buffer[fcount+4]-'0'));
279                 netfrom = synapse_out[nodefrom];
280                 fcount+=5;
281                 break;
282             default:
283                 fcount++;
284         }
285     }
286     switch (buffer[fcount]) {
287         case 'O':
288             /* connect __ to an output */
289             nodetypeto = output;
290             nodeto = ((buffer[fcount+1]-'0')*1000 +
291 (buffer[fcount+2]-'0')*100 + (buffer[fcount+3]-'0')*10 +
292 (buffer[fcount+4]-'0'));
293             netto = output_in[nodeto];
294             fcount+=6;
295             break;
296         case 'N':
297             /* connect __ to a neuron input */
298             nodetypeto = neuron;
299             nodeto = ((buffer[fcount+1]-'0')*1000 +
300 (buffer[fcount+2]-'0')*100 + (buffer[fcount+3]-'0')*10 +
301 (buffer[fcount+4]-'0'));
302             netto = neuron_in[nodeto];
303             fcount+=6;
304             break;
```

```
294         case 'S':
295             /* connect __ to a synapse input */
296             nodetypeto = synapse;
297             nodeto = ((buffer[fcount+1]-'0')*1000 +
(buffer[fcount+2]-'0')*100 + (buffer[fcount+3]-'0')*10 +
(buffer[fcount+4]-'0'));
298             netto = synapse_in[nodeto];
299             fcount+=6;
300             break;
301         default:
302             fcount++;
303     }
304     if ((netto == netfrom) && (netto != 0)) {
305         /* connection already exists, no need to change netlist */
306         if (verbose)
307             printf("NET %04ld connection already exists\n", netto);
308     } else if ((netto == 0) && (netfrom != 0)) {
309         /* change the net in netto to the net in netfrom */
310         if (verbose)
311             printf("NET %04ld connection made ", netfrom);
312         switch (nodetypeto) {
313             case output:
314                 output_in[nodeto] = netfrom;
315                 if (verbose)
316                     printf("to OUTPUT %04ld\n", nodeto);
317                 break;
318             case neuron:
319                 neuron_in[nodeto] = netfrom;
320                 if (verbose)
321                     printf("to NEURON %04ld (input)\n", nodeto);
322                 break;
323             case synapse:
324                 synapse_in[nodeto] = netfrom;
325                 if (verbose)
326                     printf("to SYNAPSE %04ld (input)\n", nodeto);
327                 break;
328         }
329     } else if ((netfrom == 0) && (netto != 0)) {
330         /* change the net in netfrom to the net in netto */
331         if (verbose)
332             printf("NET %04ld connection made ", netto);
333         switch (nodetypefrom) {
334             case input:
335                 input_out[nodetofrom] = netto;
336                 if (verbose)
337                     printf("to INPUT %04ld\n", nodetofrom);
338                 break;
339             case neuron:
340                 neuron_out[nodetofrom] = netto;
341                 if (verbose)
342                     printf("to NEURON %04ld (output)\n", nodetofrom);
343                 break;
344             case synapse:
345                 synapse_out[nodetofrom] = netto;
346                 if (verbose)
347                     printf("to SYNAPSE %04ld (output)\n", nodetofrom);
348                 break;
349         }
350     } else if ((netfrom == 0) && (netto == 0)) {
351         /* create new net */
352         nets++;
353         if (verbose)
```

```
354         printf("NET %04ld created ..... ", nets);
355     switch (nodetypefrom) {
356     case input:
357         input_out[nodetofrom] = nets;
358         if (verbose)
359             printf("for INPUT   %04ld ..... ", nodetofrom);
360         break;
361     case neuron:
362         neuron_out[nodetofrom] = nets;
363         if (verbose)
364             printf("for NEURON  %04ld (output) ", nodetofrom);
365         break;
366     case synapse:
367         synapse_out[nodetofrom] = nets;
368         if (verbose)
369             printf("for SYNAPSE %04ld (output) ", nodetofrom);
370         break;
371     }
372     switch (nodetypeto) {
373     case output:
374         output_in[nodeto] = nets;
375         if (verbose)
376             printf("and OUTPUT  %04ld\n", nodeto);
377         break;
378     case neuron:
379         neuron_in[nodeto] = nets;
380         if (verbose)
381             printf("and NEURON  %04ld (input)\n", nodeto);
382         break;
383     case synapse:
384         synapse_in[nodeto] = nets;
385         if (verbose)
386             printf("and SYNAPSE %04ld (input)\n", nodeto);
387         break;
388     }
389     } else {
390     /*
391     this is a bit hard: join the two nets together
392     it should never happen
393     add later as an additional feature
394     */
395
396     if (verbose)
397         printf("[NET discrepancy between %04ld and %04ld
found]\n", netfrom, netto);
398     }
399     }
400
401     /*
402     now the nets are sorted out
403     start arranging neurons and synapses, for minimal net lengths
404     */
405
406     return 0;
407     } else {
408     fclose(nfile);
409     printf("File Header is Invalid!\n");
410     return 1;
411     }
412
413     } else {
414     printf("Cannot open file!\n");
```

```
415         return 1;  
416     }  
417 }  
418 }
```