

```

1 #ifndef CALCULATION
2 #define CALCULATION
3
4 #include <stdbool.h>
5
6 #define _USE_MATH_DEFINES
7 #include <math.h>
8
9 #include "../datatypes/enum.h"
10 #include "../datatypes/struct.h"
11
12 #include "print.h"
13 #include "selection.h"
14
15 #define ONES 1
16 #define TENS 10
17 #define HUNDREDS 100
18 #define THOUSANDS 1000
19
20 double GetParameterInput(void (*paramInstructions)(char *parameter), char *parameter)
21 {
22     char *endptr, buffer[100];
23     double number;
24
25     (*paramInstructions)(parameter);
26
27     while (fgets(buffer, sizeof(buffer), stdin))
28     {
29         number = strtod(buffer, &endptr);
30         if (endptr == buffer || *endptr != '\n')
31         {
32             NumericInputAlert(false);
33         }
34         else if (number <= 0)
35         {
36             NumericInputAlert(true);
37         }
38         else
39         {
40             return number;
41         }
42     }
43 }
44
45 void AssignRectangleParameter(struct History *history, int base)
46 {
47     history->rectangles[history->count[0]].width = GetParameterInput(ParamaterSelectionInstructions, "width") / base;
48     history->rectangles[history->count[0]].length = GetParameterInput(ParamaterSelectionInstructions, "length") / base;
49 }
50
51 void GetRectangleParameter(struct History *history, enum unit *unit)
52 {
53     switch (*unit)
54     {
55     case m:
56         AssignRectangleParameter(&(*history), ONES);
57         break;
58
59     case dm:
60         AssignRectangleParameter(&(*history), TENS);
61         break;
62
63     case cm:
64         AssignRectangleParameter(&(*history), HUNDREDS);
65         break;
66
67     case mm:
68         AssignRectangleParameter(&(*history), THOUSANDS);
69         break;
70     }
71 }
72
73 void AssignSquareParameter(struct History *history, int base)
74 {
75     history->squares[history->count[1]].length = GetParameterInput(ParamaterSelectionInstructions, "length") / base;
76 }
77
78 void GetSquareParameter(struct History *history, enum unit *unit)
79 {
80     switch (*unit)
81     {
82     case m:
83         AssignSquareParameter(&(*history), ONES);
84         break;
85
86     case dm:
87         AssignSquareParameter(&(*history), TENS);
88         break;
89
90     case cm:
91         AssignSquareParameter(&(*history), HUNDREDS);
92         break;
93
94     case mm:

```

```

95     AssignSquareParameter(&(*history), THOUSANDS);
96     break;
97 }
98 }
99
100 void AssignCircleParameter(struct History *history, int base)
101 {
102     history->circles[history->count[2]].radius = GetParameterInput(ParamaterSelectionInstructions, "radius") / base;
103 }
104
105 void GetCircleParameter(struct History *history, enum unit *unit)
106 {
107     switch (*unit)
108     {
109     case m:
110         AssignCircleParameter(&(*history), ONES);
111         break;
112
113     case dm:
114         AssignCircleParameter(&(*history), TENS);
115         break;
116
117     case cm:
118         AssignCircleParameter(&(*history), HUNDREDS);
119         break;
120
121     case mm:
122         AssignCircleParameter(&(*history), THOUSANDS);
123         break;
124     }
125 }
126
127 void AssignCuboidParameter(struct History *history, int base)
128 {
129     history->cuboids[history->count[3]].width = GetParameterInput(ParamaterSelectionInstructions, "width") / base;
130     history->cuboids[history->count[3]].length = GetParameterInput(ParamaterSelectionInstructions, "length") / base;
131     history->cuboids[history->count[3]].height = GetParameterInput(ParamaterSelectionInstructions, "height") / base;
132 }
133
134 void GetCuboidParameter(struct History *history, enum unit *unit)
135 {
136     switch (*unit)
137     {
138     case m:
139         AssignCuboidParameter(&(*history), ONES);
140         break;
141
142     case dm:
143         AssignCuboidParameter(&(*history), TENS);
144         break;
145
146     case cm:
147         AssignCuboidParameter(&(*history), HUNDREDS);
148         break;
149
150     case mm:
151         AssignCuboidParameter(&(*history), THOUSANDS);
152         break;
153     }
154 }
155
156 void AssignCubeParameter(struct History *history, int base)
157 {
158     history->cubes[history->count[4]].length = GetParameterInput(ParamaterSelectionInstructions, "length") / base;
159 }
160
161 void GetCubeParameter(struct History *history, enum unit *unit)
162 {
163     switch (*unit)
164     {
165     case m:
166         AssignCubeParameter(&(*history), ONES);
167         break;
168
169     case dm:
170         AssignCubeParameter(&(*history), TENS);
171         break;
172
173     case cm:
174         AssignCubeParameter(&(*history), HUNDREDS);
175         break;
176
177     case mm:
178         AssignCubeParameter(&(*history), THOUSANDS);
179         break;
180     }
181 }
182
183 void AssignSphereParameter(struct History *history, int base)
184 {
185     history->spheres[history->count[5]].radius = GetParameterInput(ParamaterSelectionInstructions, "radius") / base;
186 }
187
188 void GetSphereParameter(struct History *history, enum unit *unit)
189 {

```

```

190     switch (*unit)
191     {
192     case m:
193         AssignSphereParameter(&(*history), ONES);
194         break;
195
196     case dm:
197         AssignSphereParameter(&(*history), TENS);
198         break;
199
200     case cm:
201         AssignSphereParameter(&(*history), HUNDREDS);
202         break;
203
204     case mm:
205         AssignSphereParameter(&(*history), THOUSANDS);
206         break;
207     }
208 }
209
210 void AssignConeParameter(struct History *history, int base)
211 {
212     history->cones[history->count[6]].radius = GetParameterInput(ParamaterSelectionInstructions, "radius") / base;
213     history->cones[history->count[6]].height = GetParameterInput(ParamaterSelectionInstructions, "height") / base;
214 }
215
216 void GetConeParameter(struct History *history, enum unit *unit)
217 {
218     switch (*unit)
219     {
220     case m:
221         AssignConeParameter(&(*history), ONES);
222         break;
223
224     case dm:
225         AssignConeParameter(&(*history), TENS);
226         break;
227
228     case cm:
229         AssignConeParameter(&(*history), HUNDREDS);
230         break;
231
232     case mm:
233         AssignConeParameter(&(*history), THOUSANDS);
234         break;
235     }
236 }
237
238 void CalculateProperties(enum shape shape, struct History *history)
239 {
240     enum unit unit;
241
242     UnitSelection(&unit);
243
244     switch (shape)
245     {
246     case Rectangle:
247         GetRectangleParameter(&(*history), &unit);
248
249         history->rectangles[history->count[0]].perimeter = 2 * (history->rectangles[history->count[0]].width + history->rectangles[history->count[0]].length);
250         history->rectangles[history->count[0]].area = history->rectangles[history->count[0]].width * history->rectangles[history->count[0]].length;
251
252         DisplayResults(shape, history->rectangles[history->count[0]].perimeter, history->rectangles[history->count[0]].area);
253         history->count[0]++;
254
255         break;
256
257     case Square:
258         GetSquareParameter(&(*history), &unit);
259
260         history->squares[history->count[1]].perimeter = 4 * history->squares[history->count[1]].length;
261         history->squares[history->count[1]].area = history->squares[history->count[1]].length * history->squares[history->count[1]].length;
262
263         DisplayResults(shape, history->squares[history->count[1]].perimeter, history->squares[history->count[1]].area);
264         history->count[1]++;
265
266         break;
267
268     case Circle:
269         GetCircleParameter(&(*history), &unit);
270
271         history->circles[history->count[2]].circumference = 2 * M_PI * history->circles[history->count[2]].radius;
272         history->circles[history->count[2]].area = M_PI * history->circles[history->count[2]].radius * history->circles[history->count[2]].radius;
273
274         DisplayResults(shape, history->circles[history->count[2]].circumference, history->circles[history->count[2]].area);
275         history->count[2]++;
276
277         break;
278
279     case Cuboid:
280         GetCuboidParameter(&(*history), &unit);
281
282         history->cuboids[history->count[3]].area = 2 * (history->cuboids[history->count[3]].width * history->cuboids[history->count[3]].length + history->cuboids[history->count[3]].width * history->cuboids[history->count[3]].height + history->cuboids[history->count[3]].length * history->cuboids[history->count[3]].height);

```

```

>count[3]].height);
283     history->cuboids[history->count[3]].volume = history->cuboids[history->count[3]].width * history->cuboids[history->count[3]].length * history-
>cuboids[history->count[3]].height;
284
285     DisplayResults(shape, history->cuboids[history->count[3]].area, history->cuboids[history->count[3]].volume);
286     history->count[3]++;
287
288     break;
289
290     case Cube:
291         GetCubeParameter(&(*history), &unit);
292
293         history->cubes[history->count[4]].area = 6 * history->cubes[history->count[4]].length * history->cubes[history->count[4]].length;
294         history->cubes[history->count[4]].volume = history->cubes[history->count[4]].length * history->cubes[history->count[4]].length * history-
>cubes[history->count[4]].length;
295
296         DisplayResults(shape, history->cubes[history->count[4]].area, history->cubes[history->count[4]].volume);
297         history->count[4]++;
298
299         break;
300
301     case Sphere:
302         GetSphereParameter(&(*history), &unit);
303
304         history->spheres[history->count[5]].area = 4 * M_PI * history->spheres[history->count[5]].radius * history->spheres[history->count[5]].radius;
305         history->spheres[history->count[5]].volume = 4 / 3 * M_PI * history->spheres[history->count[5]].radius * history->spheres[history->count[5]].radius
* history->spheres[history->count[5]].radius;
306
307         DisplayResults(shape, history->spheres[history->count[5]].area, history->spheres[history->count[5]].volume);
308         history->count[5]++;
309
310         break;
311
312     case Cone:
313         GetConeParameter(&(*history), &unit);
314
315         history->cones[history->count[6]].area = M_PI * history->cones[history->count[6]].radius * (history->cones[history->count[6]].radius + sqrt(history-
>cones[history->count[6]].radius * history->cones[history->count[6]].radius + history->cones[history->count[6]].height * history->cones[history-
>count[6]].height));
316         history->cones[history->count[6]].volume = M_PI * history->cones[history->count[6]].radius * history->cones[history->count[6]].radius * history-
>cones[history->count[6]].height / 3;
317
318         DisplayResults(shape, history->cones[history->count[6]].area, history->cones[history->count[6]].volume);
319         history->count[6]++;
320
321         break;
322     }
323 }
324
325 void CalculateHistoricalProperties(struct History *history)
326 {
327     enum shape shape;
328     int i, parameters;
329     double *means, *stds;
330
331     ShapeAndObjectSelection(&shape);
332
333     switch (shape)
334     {
335     case Rectangle:
336         parameters = 4;
337         if ((means = (double *)malloc(parameters * sizeof(double))) == NULL)
338         {
339             NoMemoryAlert();
340             exit(1);
341         }
342
343         if ((stds = (double *)malloc(parameters * sizeof(double))) == NULL)
344         {
345             NoMemoryAlert();
346             exit(1);
347         }
348
349         for (i = 0; i < parameters; i++)
350         {
351             means[i] = 0;
352             stds[i] = 0;
353         }
354
355         for (i = 0; i < history->count[0]; i++)
356         {
357             means[0] += history->rectangles[i].width;
358             means[1] += history->rectangles[i].length;
359             means[2] += history->rectangles[i].perimeter;
360             means[3] += history->rectangles[i].area;
361         }
362
363         means[0] /= history->count[0];
364         means[1] /= history->count[0];
365         means[2] /= history->count[0];
366         means[3] /= history->count[0];
367
368         for (i = 0; i < history->count[0]; i++)
369         {
370             stds[0] += pow(history->rectangles[i].width - means[0], 2);
371             stds[1] += pow(history->rectangles[i].length - means[1], 2);

```

```

372         stds[2] += pow(history->rectangles[i].perimeter - means[2], 2);
373         stds[3] += pow(history->rectangles[i].area - means[3], 2);
374     }
375
376     stds[0] = sqrt(stds[0] / history->count[0]);
377     stds[1] = sqrt(stds[1] / history->count[0]);
378     stds[2] = sqrt(stds[2] / history->count[0]);
379     stds[3] = sqrt(stds[3] / history->count[0]);
380
381     DisplayHistoryTable(shape, history, means, stds);
382     free(means);
383     free(stds);
384     break;
385
386 case Square:
387     parameters = 3;
388
389     if ((means = (double *)malloc(parameters * sizeof(double))) == NULL)
390     {
391         NoMemoryAlert();
392         exit(1);
393     }
394
395     if ((stds = (double *)malloc(parameters * sizeof(double))) == NULL)
396     {
397         NoMemoryAlert();
398         exit(1);
399     }
400
401     for (i = 0; i < parameters; i++)
402     {
403         means[i] = 0;
404         stds[i] = 0;
405     }
406
407     for (i = 0; i < history->count[1]; i++)
408     {
409         means[0] += history->squares[i].length;
410         means[1] += history->squares[i].perimeter;
411         means[2] += history->squares[i].area;
412     }
413
414     means[0] /= history->count[1];
415     means[1] /= history->count[1];
416     means[2] /= history->count[1];
417
418     for (i = 0; i < history->count[1]; i++)
419     {
420         stds[0] += pow(history->squares[i].length - means[0], 2);
421         stds[1] += pow(history->squares[i].perimeter - means[1], 2);
422         stds[2] += pow(history->squares[i].area - means[2], 2);
423     }
424
425     stds[0] = sqrt(stds[0] / history->count[1]);
426     stds[1] = sqrt(stds[1] / history->count[1]);
427     stds[2] = sqrt(stds[2] / history->count[1]);
428
429     DisplayHistoryTable(shape, history, means, stds);
430     free(means);
431     free(stds);
432     break;
433
434 case Circle:
435     parameters = 3;
436
437     if ((means = (double *)malloc(parameters * sizeof(double))) == NULL)
438     {
439         NoMemoryAlert();
440         exit(1);
441     }
442
443     if ((stds = (double *)malloc(parameters * sizeof(double))) == NULL)
444     {
445         NoMemoryAlert();
446         exit(1);
447     }
448
449     for (i = 0; i < parameters; i++)
450     {
451         means[i] = 0;
452         stds[i] = 0;
453     }
454
455     for (i = 0; i < history->count[2]; i++)
456     {
457         means[0] += history->circles[i].radius;
458         means[1] += history->circles[i].circumference;
459         means[2] += history->circles[i].area;
460     }
461
462     means[0] /= history->count[2];
463     means[1] /= history->count[2];
464     means[2] /= history->count[2];
465
466     for (i = 0; i < history->count[2]; i++)

```

```

467     {
468         stds[0] += pow(history->circles[i].radius - means[0], 2);
469         stds[1] += pow(history->circles[i].circumference - means[1], 2);
470         stds[2] += pow(history->circles[i].area - means[2], 2);
471     }
472
473     stds[0] = sqrt(stds[0] / history->count[2]);
474     stds[1] = sqrt(stds[1] / history->count[2]);
475     stds[2] = sqrt(stds[2] / history->count[2]);
476
477     DisplayHistoryTable(shape, history, means, stds);
478     free(means);
479     free(stds);
480     break;
481
482 case Cuboid:
483     parameters = 5;
484
485     if ((means = (double *)malloc(parameters * sizeof(double))) == NULL)
486     {
487         NoMemoryAlert();
488         exit(1);
489     }
490
491     if ((stds = (double *)malloc(parameters * sizeof(double))) == NULL)
492     {
493         NoMemoryAlert();
494         exit(1);
495     }
496
497     for (i = 0; i < parameters; i++)
498     {
499         means[i] = 0;
500         stds[i] = 0;
501     }
502
503     for (i = 0; i < history->count[3]; i++)
504     {
505         means[0] += history->cuboids[i].length;
506         means[1] += history->cuboids[i].width;
507         means[2] += history->cuboids[i].height;
508         means[3] += history->cuboids[i].area;
509         means[4] += history->cuboids[i].volume;
510     }
511
512     means[0] /= history->count[3];
513     means[1] /= history->count[3];
514     means[2] /= history->count[3];
515     means[3] /= history->count[3];
516     means[4] /= history->count[3];
517
518     for (i = 0; i < history->count[3]; i++)
519     {
520         stds[0] += pow(history->cuboids[i].length - means[0], 2);
521         stds[1] += pow(history->cuboids[i].width - means[1], 2);
522         stds[2] += pow(history->cuboids[i].height - means[2], 2);
523         stds[3] += pow(history->cuboids[i].area - means[3], 2);
524         stds[4] += pow(history->cuboids[i].volume - means[4], 2);
525     }
526
527     stds[0] = sqrt(stds[0] / history->count[3]);
528     stds[1] = sqrt(stds[1] / history->count[3]);
529     stds[2] = sqrt(stds[2] / history->count[3]);
530     stds[3] = sqrt(stds[3] / history->count[3]);
531     stds[4] = sqrt(stds[4] / history->count[3]);
532
533     DisplayHistoryTable(shape, history, means, stds);
534     free(means);
535     free(stds);
536     break;
537
538 case Cube:
539     parameters = 3;
540
541     if ((means = (double *)malloc(parameters * sizeof(double))) == NULL)
542     {
543         NoMemoryAlert();
544         exit(1);
545     }
546
547     if ((stds = (double *)malloc(parameters * sizeof(double))) == NULL)
548     {
549         NoMemoryAlert();
550         exit(1);
551     }
552
553     for (i = 0; i < parameters; i++)
554     {
555         means[i] = 0;
556         stds[i] = 0;
557     }
558
559     for (i = 0; i < history->count[4]; i++)
560     {
561         means[0] += history->cubes[i].length;

```

```

562     means[1] += history->cubes[i].area;
563     means[2] += history->cubes[i].volume;
564 }
565
566 means[0] /= history->count[4];
567 means[1] /= history->count[4];
568 means[2] /= history->count[4];
569
570 for (i = 0; i < history->count[4]; i++)
571 {
572     stds[0] += pow(history->cubes[i].length - means[0], 2);
573     stds[1] += pow(history->cubes[i].area - means[1], 2);
574     stds[2] += pow(history->cubes[i].volume - means[2], 2);
575 }
576
577 stds[0] = sqrt(stds[0] / history->count[4]);
578 stds[1] = sqrt(stds[1] / history->count[4]);
579 stds[2] = sqrt(stds[2] / history->count[4]);
580
581 DisplayHistoryTable(shape, history, means, stds);
582 free(means);
583 free(stds);
584 break;
585
586 case Sphere:
587     parameters = 3;
588
589     if ((means = (double *)malloc(parameters * sizeof(double))) == NULL)
590     {
591         NoMemoryAlert();
592         exit(1);
593     }
594
595     if ((stds = (double *)malloc(parameters * sizeof(double))) == NULL)
596     {
597         NoMemoryAlert();
598         exit(1);
599     }
600
601     for (i = 0; i < parameters; i++)
602     {
603         means[i] = 0;
604         stds[i] = 0;
605     }
606
607     for (i = 0; i < history->count[5]; i++)
608     {
609         means[0] += history->spheres[i].radius;
610         means[1] += history->spheres[i].area;
611         means[2] += history->spheres[i].volume;
612     }
613
614     means[0] /= history->count[5];
615     means[1] /= history->count[5];
616     means[2] /= history->count[5];
617
618     for (i = 0; i < history->count[5]; i++)
619     {
620         stds[0] += pow(history->spheres[i].radius - means[0], 2);
621         stds[1] += pow(history->spheres[i].area - means[1], 2);
622         stds[2] += pow(history->spheres[i].volume - means[2], 2);
623     }
624
625     stds[0] = sqrt(stds[0] / history->count[5]);
626     stds[1] = sqrt(stds[1] / history->count[5]);
627     stds[2] = sqrt(stds[2] / history->count[5]);
628
629     DisplayHistoryTable(shape, history, means, stds);
630     free(means);
631     free(stds);
632     break;
633
634 case Cone:
635     parameters = 4;
636
637     if ((means = (double *)malloc(parameters * sizeof(double))) == NULL)
638     {
639         NoMemoryAlert();
640         exit(1);
641     }
642
643     if ((stds = (double *)malloc(parameters * sizeof(double))) == NULL)
644     {
645         NoMemoryAlert();
646         exit(1);
647     }
648
649     for (i = 0; i < parameters; i++)
650     {
651         means[i] = 0;
652         stds[i] = 0;
653     }
654
655     for (i = 0; i < history->count[6]; i++)
656     {

```

```

657     means[0] += history->cones[i].radius;
658     means[1] += history->cones[i].height;
659     means[2] += history->cones[i].area;
660     means[3] += history->cones[i].volume;
661 }
662
663 means[0] /= history->count[6];
664 means[1] /= history->count[6];
665 means[2] /= history->count[6];
666 means[3] /= history->count[6];
667
668 for (i = 0; i < history->count[6]; i++)
669 {
670     stds[0] += pow(history->cones[i].radius - means[0], 2);
671     stds[1] += pow(history->cones[i].height - means[1], 2);
672     stds[2] += pow(history->cones[i].area - means[2], 2);
673     stds[3] += pow(history->cones[i].volume - means[3], 2);
674 }
675
676 stds[0] = sqrt(stds[0] / history->count[6]);
677 stds[1] = sqrt(stds[1] / history->count[6]);
678 stds[2] = sqrt(stds[2] / history->count[6]);
679 stds[3] = sqrt(stds[3] / history->count[6]);
680
681 DisplayHistoryTable(shape, history, means, stds);
682 free(means);
683 free(stds);
684 break;
685 }
686 }
687
688 #endif

```