```
1 // file: combine.h
2 #define INT
3
4 #ifdef FLOAT
5 typedef double data_t;
6 #define DATA_NAME "Float"
7 #endif
8
9 #ifdef DOUBLE
10 typedef double data_t;
11 #define DATA_NAME "Double"
12 #endif
13
14
15 #ifdef EXTEND
16 typedef long double data_t;
17 #define DATA_NAME "Extended"
18 #endif
19
20 #ifdef INT
21 typedef int data_t;
22 #define DATA_NAME "Integer"
23 #endif
24
25 #ifdef LONG
26 typedef long data_t;
27 #define DATA_NAME "Long"
28 #endif
29
30 #ifdef CHAR
31 typedef char data_t;
32 #define DATA_NAME "Char"
33 #endif
34
35 #ifdef PROD
36 #define IDENT 1
37 #define OP  *
38 #define OP_NAME "Product"
39 #else
40 #ifdef DIV
41 #define OP /
42 #define IDENT 1
43 #define OP_NAME "Divide"
44 #else
45 #define IDENT 0
46 #define OP  +
47 #define OP_NAME "Sum"
48 #endif /* DIV */
49 #endif /* PROD */
50
51 #include "vec.h"
52
53 void combine1(vec_ptr v, data_t *dest);
54 void combine2(vec_ptr v, data_t *dest);
55 void combine3(vec_ptr v, data_t *dest);
56 void combine4(vec_ptr v, data_t *dest);
57
```

```
58 // file: combine.c
59
60 #include "combine.h"
61
62 void combine1(vec_ptr v, data_t *dest)
63 {
64     *dest = IDENT;
65     for (long i = 0; i < vec_length(v); i++) {
66         data_t val;
67         get_vec_element(v, i, &val);
68         *dest = *dest OP val;
69     }
70 }
71
72 // move call to vec_length out of loop
73 void combine2(vec_ptr v, data_t *dest)
74 {
75     long length = vec_length(v);
76
77     *dest = IDENT;
78     for (long i = 0; i < length; i++) {
79         data_t val;
80         get_vec_element(v, i, &val);
81         *dest = *dest OP val;
82     }
83 }
84
85 // direct access to vector data
86 void combine3(vec_ptr v, data_t *dest)
87 {
88     long length = vec_length(v);
89     data_t *data = get_vec_start(v);
90
91     *dest = IDENT;
92     for (long i = 0; i < length; i++) {
93         *dest = *dest OP data[i];
94     }
95 }
96
97 // accumulate result in local variable
98 void combine4(vec_ptr v, data_t *dest)
99 {
100     long length = vec_length(v);
101     data_t *data = get_vec_start(v);
102     data_t acc = IDENT;
103
104     for (long i = 0; i < length; i++) {
105         acc = acc OP data[i];
106     }
107
108     *dest = acc;
109 }
110
```

```
111 // file: vec.h
112
113 /* Create abstract data type for vector */
114 typedef struct {
115     long len;
116     data_t *data;
117 } vec_rec, *vec_ptr;
118
119 /* Create vector */
120 vec_ptr new_vec(long len);
121
122 /* Free storage used by vector */
123 void free_vec(vec_ptr v);
124
125 /*
126  * Retrieve vector element and store in dest.
127  * Return 0 (out of bounds) or 1 (successful)
128  */
129 int get_vec_element(vec_ptr v, long index, data_t *dest);
130
131 /* Macro version */
132 #define GET_VEC_ELEMENT(v,index,dest) \
133     !((index) < 0 || (index) >= (v)->len) && \
134     *(dest) = (v)->data[(index)], 1;
135
136 data_t *get_vec_start(vec_ptr v);
137
138 /*
139  * Set vector element.
140  * Return 0 (out of bounds) or 1 (successful)
141  */
142
143 int set_vec_element(vec_ptr v, long index, data_t val);
144
145 /* Get vector length */
146 long vec_length(vec_ptr v);
147
148 // file: vec.c
149
150 #include <stdlib.h>
151 #include <stdio.h>
152 #include "combine.h"
153 #include "fcyc.h"
154
155 #define MILLION 1e6
156 #define VEC_SIZE 100000000
157
158 int main(int argc, char **argv)
159 {
160     data_t result;
161     vec_ptr v = new_vec(VEC_SIZE);
162
163     printf("combine1\tcombine2\tcombine3\tcombine4\n");
164     for (int i=0; i < VEC_SIZE; i++) {
165         v->data[i] = i;
166     }
167
```

```
168     start_counter();
169     combine1(v,&result);
170     printf("%f\t",get_counter()/VEC_SIZE);
171     fflush(stdout);
172 
173     start_counter();
174     combine2(v,&result);
175     printf("%f\t",get_counter()/VEC_SIZE);
176     fflush(stdout);
177 
178     start_counter();
179     combine3(v,&result);
180     printf("%f\t",get_counter()/VEC_SIZE);
181     fflush(stdout);
182 
183     start_counter();
184     combine4(v,&result);
185     printf("%f\n",get_counter()/VEC_SIZE);
186 
187     free_vec(v);
188     return 0;
189 }
190 
191 
192 /* Create vector of specified length */
193 vec_ptr new_vec(long len)
194 {
195     /* Allocate header structure */
196     vec_ptr result = (vec_ptr) malloc(sizeof(vec_rec));
197     data_t *data = NULL;
198     if (!result)
199         return NULL;  /* Couldn't allocate storage */
200     result->len = len;
201     /* Allocate array */
202     if (len > 0) {
203         data = (data_t *)calloc(len, sizeof(data_t));
204         if (!data) {
205             free((void *) result);
206             return NULL; /* Couldn't allocate storage */
207         }
208     }
209     /* data will either be NULL or allocated array */
210     result->data = data;
211     return result;
212 }
213 
214 /* Free storage used by vector */
215 void free_vec(vec_ptr v) {
216     if (v->data)
217         free(v->data);
218     free(v);
219 }
220 
221 /*
222  * Retrieve vector element and store at dest.
223  * Return 0 (out of bounds) or 1 (successful)
224  */
```

```
225 int get_vec_element(vec_ptr v, long index, data_t *dest)
226 {
227     if (index < 0 || index >= v->len)
228         return 0;
229     *dest = v->data[index];
230     return 1;
231 }
232
233 /* Return length of vector */
234 long vec_length(vec_ptr v)
235 {
236     return v->len;
237 }
238
239 data_t *get_vec_start(vec_ptr v)
240 {
241     return v->data;
242 }
243
244 /*
245  * Set vector element.
246  * Return 0 (out of bounds) or 1 (successful)
247  */
248 int set_vec_element(vec_ptr v, long index, data_t val)
249 {
250     if (index < 0 || index >= v->len)
251         return 0;
252     v->data[index] = val;
253     return 1;
254 }
```