

```

13 #include <iostream>
14 #include <string>
15 #include <vector>
16
17 enum class ButtonEventType {
18     SHORT_PRESS,
19     LONG_PRESS,
20     VERY_LONG_PRESS
21 };
22
23 enum class ButtonSourceId {
24     BUTTON_0,
25     BUTTON_1
26 };
27
28 class ButtonEvent {
29 private:
30     ButtonEventType typeId;
31     ButtonSourceId sourceId;
32
33 public:
34     ButtonEvent(ButtonEventType type, ButtonSourceId source)
35         : typeId(type), sourceId(source) {}
36
37     ButtonEventType getType() const { return typeId; }
38     ButtonSourceId getSource() const { return sourceId; }
39 };
40
41 class ButtonEventDispatcher {
42 private:
43     std::vector<void(*)(const ButtonEvent&)> handlers;
44     static const uint8_t MAX_BUTTON_HANDLERS = 4;
45 public:
46     void RegisterButtonHandler(void (*handler)(const ButtonEvent& e)) {
47         if (handlers.size() < MAX_BUTTON_HANDLERS) {
48             {
49                 handlers.push_back(handler);
50             }
51             else
52             {
53                 std::cout << "ERROR: Only " << std::to_string(MAX_BUTTON_HANDLERS) << " handlers allowed!" <<
std::endl;
54             }
55         }
56
57         void UnregisterButtonHandler(void(*handler)(const ButtonEvent&)) {
58             for (auto it = handlers.begin(); it != handlers.end(); ++it) {
59                 if (*it == handler) {
60                     handlers.erase(it);
61                     break;
62                 }
63             }
64         }
65
66         void Dispatch(const ButtonEvent& e) const {
67             for (const auto& handler : handlers) {
68                 handler(e);
69             }
70         }
71     };
72
73     void buttonEventHandler(const ButtonEvent& event) {
74         std::string type;
75         std::string source;
76         switch (event.getType()) {
77             case ButtonEventType::SHORT_PRESS:
78                 type = "Short press";
79                 break;
80             case ButtonEventType::LONG_PRESS:
81                 type = "Long press";
82                 break;

```

```

83         case ButtonEventType::VERY_LONG_PRESS:
84             type = "Very long press";
85             break;
86     }
87     switch (event.getSource()) {
88         case ButtonSourceId::BUTTON_0:
89             source = "Button 0";
90             break;
91         case ButtonSourceId::BUTTON_1:
92             source = "Button 1";
93             break;
94     }
95     std::cout << "Button event occurred: " << type << " from " << source << std::endl;
96 }
97
98 void handle1(const ButtonEvent& event) {
99     std::cout << "Handle1 called " << std::endl;
100 }
101
102 void handle2(const ButtonEvent& event) {
103     std::cout << "Handle2 called " << std::endl;
104 }
105
106 void handle3(const ButtonEvent& event) {
107     std::cout << "Handle3 called " << std::endl;
108 }
109
110 void handle4(const ButtonEvent& event) {
111     std::cout << "Handle4 called " << std::endl;
112 }
113
114 int data = 0;
115 int bss;
116
117 int function(int a, int b)
118 {
119     return a + b;
120 }
121
122 int main() {
123     int stack = 0;
124     int* heap = new int;
125     int (*text) (int, int) = function;
126
127     printf("\nStack pointer:%p\nHeap pointer:%p\nData pointer:%p\nBSS pointer:%p\nText pointer:%p\n\n", &stack,
128 heap, &data, &bss, text);
129
130     ButtonEventDispatcher dispatcher;
131     const ButtonEvent event(ButtonEventType::SHORT_PRESS, ButtonSourceId::BUTTON_0);
132
133     // Register the callback functions (maximum is 4)
134     dispatcher.RegisterButtonHandler(buttonEventHandler);
135     dispatcher.RegisterButtonHandler(handle1);
136     dispatcher.RegisterButtonHandler(handle2);
137     dispatcher.RegisterButtonHandler(handle3);
138     dispatcher.RegisterButtonHandler(handle4);
139
140     dispatcher.UnregisterButtonHandler(handle1);
141
142     dispatcher.RegisterButtonHandler(handle4);
143
144     // Dispatch the button event
145     dispatcher.Dispatch(event);
146
147     return 0;
148 }

```