Allocate a variable in .stack memory

```
 int x = 5;
```

Allocate a pointer variable pointing into .heap memory

```
int* ptr = new int(2);
```

Allocate a variable in .data memory

```
int global_var = 10; //initialized => .data memory
```

Allocate a variable in .bss memory

```
int global_var_not_initialized; // not initialized => .bss (memory is zero initialized before program execution)
```

Allocate a pointer variable pointing into .text section

```
void text_function() {
    std::cout << "Hello from the .text section!" << std::endl;
}
int main() {

    // Call the function through the pointer
    ptr_to_function();

    return 0;
}
```

ButtonEvent.hpp

```
#ifndef BUTTONEVENT_HPP
#define BUTTONEVENT_HPP

#include <string>

class ButtonEvent {
public:
    // Enum class for typeId
    enum class PressTypeId {
        SHORT_PRESS,
        LONG_PRESS,
        VERY_LONG_PRESS
    };

    // Enum class for sourceId
    enum class ButtonSourceId {
        BUTTON_0,
        BUTTON_1
    };

private:
    PressTypeId typeId;
    ButtonSourceId sourceId;

public:
    // Constructor
    ButtonEvent(PressTypeId typeId, ButtonSourceId sourceId);

    // Getters
    PressTypeId getTypeId() const;
    ButtonSourceId getSourceId() const;
};
#endif // BUTTONEVENT_HPP
```

ButtonEvent.cpp

```
#include "ButtonEvent.hpp"

// Constructor implementation
ButtonEvent::ButtonEvent(PressTypeId typeId, ButtonSourceId sourceId)
    : typeId(typeId), sourceId(sourceId) {}

// Getter implementation
ButtonEvent::PressTypeId ButtonEvent::getTypeId() const {
    return typeId;
}

ButtonEvent::ButtonSourceId ButtonEvent::getSourceId() const {
    return sourceId;
}
```

ButtonEventDispatcher.hpp

```
#ifndef BUTTONEVENTDISPATCHER_HPP
#define BUTTONEVENTDISPATCHER_HPP

#include <string>
#include <functional>
#include "ButtonEvent.hpp"

class ButtonEventDispatcher {
private:
   std::function<void(const ButtonEvent&)> handlers[4];
   int numHandlers;

public:

   // Constructor
   ButtonEventDispatcher();

   void RegisterButtonHandler(void (*handler)(const ButtonEvent& e));
   void UnregisterButtonHandler(void (*handler)(const ButtonEvent& e));
   void Dispatch(const ButtonEvent& e) const;
};

#endif // BUTTONEVENTDISPATCHER_HPP
```

ButonEventDispatcher.cpp

```
#include "ButtonEvent.hpp"
#include "ButtonEventDispatcher.hpp"
#include "ButtonEventDispatcher.hpp"

ButtonEventDispatcher::ButtonEventDispatcher() : numHandlers(0) {}

void ButtonEventDispatcher::RegisterButtonHandler(void (*handler)(const ButtonEvent&)) {
  if (numHandlers < 4) {
    handlers[numHandlers++] = handler;
  } else {
    // Optionally handle error: Maximum number of handlers reached
  }
}

void ButtonEventDispatcher::UnregisterButtonHandler(void (*handler)(const ButtonEvent&)) {
  for (int i = 0; i < numHandlers; ++i) {
   // if (handlers[i].target<void(const ButtonEvent&)>() == handler) {
    if (handlers[i].target_type() == typeid(handler)) {
      handlers[i] = handlers[--numHandlers];
      return;
    }
  }
  // Optionally handle error: Handler not found
}

void ButtonEventDispatcher::Dispatch(const ButtonEvent& e) const {
  for (int i = 0; i < numHandlers; ++i) {
    handlers[i](e);
  }
}
```

main.cpp

```
#include "ButtonEvent.hpp"
#include "ButtonEventDispatcher.hpp"
#include <iostream>

// Sample button event handler functions
void ButtonHandler1(const ButtonEvent& e) {
  std::cout << "ButtonHandler1 called with type: " << static_cast<int>(e.getTypeId()) << std::endl;
}
```

```
void ButtonHandler2(const ButtonEvent& e) {
    std::cout << "ButtonHandler2 called with type: " << static_cast<int>(e.getTypeId()) << std::endl;
}



int main() {

// Create ButtonEvent objects
    ButtonEvent shortPress(ButtonEvent::PressTypeId::SHORT_PRESS, ButtonEvent::ButtonSourceId::BUTTON_0);
    ButtonEvent longPress(ButtonEvent::PressTypeId::LONG_PRESS, ButtonEvent::ButtonSourceId::BUTTON_1);
    ButtonEvent veryLongPress(ButtonEvent::PressTypeId::VERY_LONG_PRESS, ButtonEvent::ButtonSourceId::BUTTON_0);

    // Access and print the type and source identifiers
    std::cout << "Button 1: " << static_cast<int>(shortPress.getTypeId()) << ", " << static_cast<int>(shortPress.getSourceId()) <<
std::endl;
    std::cout << "Button 2: " << static_cast<int>(longPress.getTypeId()) << ", " << static_cast<int>(longPress.getSourceId()) <<
std::endl;
    std::cout << "Button 3: " << static_cast<int>(veryLongPress.getTypeId()) << ", " << static_cast<int>(veryLongPress.getSourceId()) <<
std::endl;

    // Create ButtonEventDispatcher object
    ButtonEventDispatcher dispatcher;

    // Register button event handler functions
    dispatcher.RegisterButtonHandler(ButtonHandler1);
    dispatcher.RegisterButtonHandler(ButtonHandler2);

    // Create a sample button event
    ButtonEvent event(ButtonEvent::PressTypeId::VERY_LONG_PRESS, ButtonEvent::ButtonSourceId::BUTTON_1);

    // ButtonHandler1(event);
    // ButtonHandler2(event);

    // Dispatch the event
    std::cout << "Dispatching button event..." << std::endl;
    dispatcher.Dispatch(event);

    // Unregister one of the handlers
    dispatcher.UnregisterButtonHandler(ButtonHandler1);

    // Dispatch the event again
    std::cout << "Dispatching button event after unregistering one handler..." << std::endl;
    dispatcher.Dispatch(event);
    return 0;
}
```