DigiPen
INSTITUTE OF TECHNOLOGY
SINGAPORE

[CS 225] Advanced C/C++

Lecture 14: Bit manipulation *(continued)*

Sławomir "Swavek" Włodkowski
(Fall 2020)

Agenda

- Bit manipulation
  - Bit fields
  - Unions and `std::variant<typename... Ts>`
- Complex pointer declarations
  - Pointers for C programmers
  - Pointers for C++ programmers

Bit fields

Integer `class`/`struct` data members can have their size trimmed to a number of bits (0 means start a new byte).

Multiple such fields can share a byte.

```
#include <iostream>
struct MyStruct {
        unsigned char b1 : 3;
        unsigned char b2 : 3;

        unsigned char    : 0; // start a new byte
        unsigned char    : 1; // anonymous, padding
        unsigned char b3 : 1;
};
int main() { // 2
        std::cout << sizeof(MyStruct) << std::endl;
}
```

Unions

Unions are special `struct` types that can hold only one of its non-static data members at time.

Memory:

- All non-static data members may overlap in some way.
- Only one of data members should be used at each moment.
- The standard does not guarantee reading memory as one member when it was set as another (g++ does).

Unions

```
struct Mouse {
        int x;
        int y;
};

using KeyCode = char;

enum EventType {
        Mouse_Move,
        Key_Press
};

union EventData {
        Mouse mouse;
        KeyCode keyCode;
};

struct Event {
        EventType eventType;
        EventData eventData;
};
```

```
#include <iostream>
void print(Event event) {
        switch (event.eventType) {
        case Mouse_Move:
                std::cout
                << event.eventData.mouse.x
                << ", "
                << event.eventData.mouse.y
                << std::endl;
                return;
        case Key_Press:
                std::cout
                << event.eventData.keycode
                << std::endl;
                return;
        }
}
int main() {
 print(Event{Mouse_Move, {10, 20}});
 print(Event{Key_Press, 'X'});
}
```

Unions

Unions cannot have:

- Inheritance relationships.
- Virtual member functions.
- Non-static data members of reference types.

Anonymous unions (nameless, without variable definition) cannot also have:

- Member functions.
- Static data members.
- Non-public data members.

Unions

```
using KeyCode = char;

struct Event
{
        enum
        {
                Mouse_Move,
                Key_Press
        } eventType;
        union
        {
                struct Mouse
                {
                        int x;
                        int y;
                } mouse;
                KeyCode keyCode;
        };
};
```

```
#include <iostream>
void print(Event event) {
        switch (event.eventType) {
        case Event::Mouse_Move:
                std::cout
                << event.mouse.x
                << ", "
                << event.mouse.y
                << std::endl;
                return;
        case Event::Key_Press:
                std::cout
                << event.keyCode
                << std::endl;
                return;
        }
}
int main() {
 print(Event{Event::Mouse_Move, {10, 20}});
 print(Event{Event::Key_Press, 'X'});
}
```

std::variant<typename… Ts>

C++17 offers a type-safe alternative to unions.

`std::variant` (`<variant>`) keeps track of an active member, and destroys previous one if different is activated.

```
std::variant<int, float> v, w;
v = 123;      // Activate int
try {
       int i = std::get<int>(v);
       int j = std::get<0>(v);
       w = std::get<int>(v);
       w = v;
}
catch (const std::bad_variant_access&) {
}
```

```
std::variant<int, float> v, w;
v = 123;      // Activate int
try {
       float f = std::get<float>(w);
}
catch (const std::bad_variant_access&) {
       // Exception
}
```

Complex declarations

Practical exercises:

Pointers for C programmers

- Pointers to objects
- Arrays of pointers
- Pointers to arrays
- Pointers to functions

```
int* pi;
int* ap[4];
int (*pa)[2][3];
int (*pf)(float);
```

Complex declarations

Practical exercises:

Pointers for C++ programmers

- Pointers to data members `int (C::*pi);`
- Pointers to member functions `int (C::*pf)(float);`
- Pointer-to-member dereference operators
  - `.*`
  - `->*`