

Язык C++

Структуры, объединения

Структура

- это одна или несколько переменных (возможно, различных типов), которые для удобства работы с ними сгруппированы под одним именем.

Структура

```
struct Point
{
    int x;
    int y;
} pt1, pt2, pt3;

Point p4;
```

- Объявление структуры определяет тип
- Перечисленные в структуре переменные называются *элементами (членами)*

Структура

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

```
int main(int argc, char* argv[]) {  
    Point pt;  
    Point max_point = {200,300};  
  
    pt.x = 200;  
    pt.y = 250;  
  
    return 0;  
}
```

Оператор '.' позволяет
обращаться к ее членам

Вложенные структуры

```
struct Rect {  
    Point pt1;  
    Point pt2;  
};
```

```
int main(int argc, char* argv[]) {  
    Point pt;  
    pt.x = 200;  
    pt.y = 250;  
  
    Rect rec;  
    rec.pt1 = pt;  
    rec.pt2.x = 1;  
    rec.pt2.y = 2;  
    return 0;  
}
```

}

Инициализация

```
Point p1 = {1, 2};
```

```
Point p2 = {.x = 1, .y = 2};
```

```
Point p3 {.x = 1, .y = 2};
```

```
Rect r1 = { {1, 2}, p2};
```

```
Rect r2 = { .pt1 = {1, 2}, .pt2 = p2};
```

Анонимные структуры

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

```
struct {  
    size_t width;  
    size_t height;  
};  
};
```

```
Button btn = {.x = 0, .y = 100, .width = 400, .height = 80};
```

Неименованная структура

Неименованная структура, позволяет
обращаться к ее полям напрямую

Операции над структурами

- Копирования
- Присваивания
- Взятие адреса
- Доступ к элементам

Операции со структурами

```
Point MakePoint(int x, int y) {
```

```
    Point result;
```

```
    result.x = x;
```

```
    result.y = y;
```

```
    return result;
```

```
}
```

```
int main(int argc, char* argv[]) {
```

```
    Point pt = MakePoint(239, 1);
```

```
    return 0;
```

```
}
```

Операции со структурами

```
Point AddPoint(  
    Point p1,  
    Point p2  
) {  
    p1.x += p2.x;  
    p1.y += p2.y;  
  
    return p1;  
}
```

Массивы структур

```
struct Record {  
    char name[10];  
    char surname[10];  
    long phone;  
};  
  
Record phonebook[200];
```

Указатели на структуры

```
Record* FindRecord(  
    long phone,  
    Record* records,  
    int count  
) {  
    for(int i=0; i < count; ++i) {  
        if(records[i].phone == phone)  
            return &records[i];  
    }  
  
    return nullptr;  
}
```

Указатели на структуры

```
Record* key = FindRecord("22345", phonebook, 10);
```

```
if(key != nullptr) {  
    std::printf("Name: %s Surname: %s", key->name, key->surname);  
}
```

Если ***p*** – указатель на структуру, то ***p->элемент структуры*** ее отдельный элемент

Выравнивание структур

```
struct Foo {  
    char    a;    // 1 byte  
    int64_t b;    // 8 bytes  
    uint8_t c;    // 1 buty  
    uint32_t d;    // 4 byte  
};
```

Size of Foo 24

```
int main() {  
  
    std::cout << std::format("Size of Foo {} ", sizeof(Foo)) << std::endl;  
  
    return 0;  
}
```

Выравнивание структур

```
struct Foo {  
    int64_t    b;    // 8 bytes  
    uint32_t   d;    // 4 byte  
    char       a;    // 1 byte  
    uint8_t    c;    // 1 buty  
};
```

Size of Foo 16

```
int main() {  
  
    std::cout << std::format("Size of Foo {}", sizeof(Foo)) << std::endl;  
  
    return 0;  
}
```

Выравнивание структур

```
struct Foo {  
    char    a;    // 1 byte  
    int64_t b;    // 8 bytes  
    uint8_t c;    // 1 buty  
    uint32_t d;    // 4 byte  
};
```

```
int main() {
```

```
    std::cout << std::format("Size of Foo {}", sizeof(Foo)) << std::endl;
```

```
    return 0;
```

```
}
```

Компилятор может вставить пустые
байты между полями и в конце
(padding)

Выравнивание структур

```
// a x x x x x x x b b b b b b b b c x x x d d d d
```

```
struct Foo {  
    char        a;        // 0  
    int64_t     b;        // 8  
    uint8_t     c;        // 16  
    uint32_t    d;        // 20  
};
```

Каждое поле выравнивается по адресу, кратному размеру самого поля.

```
// b b b b b b b b d d d d a c x x
```

```
struct Foo {  
    int64_t     b;        // 8 bytes  
    uint32_t    d;        // 4 byte  
    char        a;        // 1 byte  
    uint8_t     c;        // 1 buty  
};
```

Общий размер структуры округляется вверх до кратного выравниванию самого крупного поля, чтобы массивы таких структур были корректно выровнены.

Выравнивание структур

```
#pragma pack(1)
struct Foo {
    char      a;
    int64_t   b;
    uint8_t   c;
    uint32_t  d;
};
#pragma pop
```

Экономит память, но уменьшает производительность

Size of Foo 14

Выравнивание структур

```
struct alignas(64) Foo {  
    char      a;  
    int64_t    b;  
    uint8_t    c;  
    uint32_t   d;  
};
```

Для специализированных оптимизаций
можно увеличить выравнивание

Size of Foo 64

Объединения

- это переменная, которая может содержать (в разные моменты времени) объекты различных типов и размеров. Все требования относительно размеров и выравнивания выполняет компилятор. Объединения позволяют хранить разнородные данные в одной и той же области памяти без включения в программу машинно-зависимой информации.

Объединение

```
union Name {  
    struct {  
        char name[13];  
        char code[3];  
    };  
  
    struct {  
        int32_t i1;  
        int32_t i2;  
        int32_t i3;  
        int32_t i4;  
    };  
};
```

Объединение

```
bool NameCompare(const Name& a, const Name& b) {  
    return (std::strcmp(a.name, b.name) == 0 && std::strcmp(a.code, b.code)) ;  
}
```

```
bool IntCompare(const Name& a, const Name& b) {  
    return (a.i1 == b.i1 && a.i2 == b.i2 && a.i3 == b.i3 && a.i4 == b.i4);  
}
```

Объединение

```
int main() {  
    Name a = {.name = "0123456789AB", .code = "12"};  
    Name b = {.name = "0123456789AB", .code = "10"};  
  
    const uint64_t retry = 1000000000000;  
  
    return 0;  
}
```

Объединение

```
const uint64_t retry = 1000000000000;
```

```
std::chrono::system_clock::time_point begin = std::chrono::system_clock::now();
```

```
for(int i = 0; i < retry; ++i)
```

```
    NameCompare(a, b);
```

```
std::chrono::system_clock::time_point end = std::chrono::system_clock::now();
```

```
std::cout
```

```
<< std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count()
```

```
<< std::endl;
```


Google Benchmark

```
static void NameComparison(benchmark::State& state) {  
    Name a = {.name = "0123456789AB", .code = "12"};  
    Name b = {.name = "0123456789AB", .code = "10"};  
  
    benchmark::DoNotOptimize(a);  
    benchmark::DoNotOptimize(b);  
    for (auto _ : state) {  
        bool result = NameCompare(a, b);  
        benchmark::DoNotOptimize(result);  
    }  
}  
  
BENCHMARK(NameComparison);
```




[Google Benchmark](#) - library to
benchmark code snippet

Google Benchmark

```
static void IntComparison(benchmark::State& state) {  
    Name a = {.name = "0123456789AB", .code = "12"};  
    Name b = {.name = "0123456789AB", .code = "10"};  
  
    benchmark::DoNotOptimize(a);  
    benchmark::DoNotOptimize(b);  
    for (auto _ : state) {  
        bool result = IntCompare(a, b);  
        benchmark::DoNotOptimize(result);  
    }  
}  
  
BENCHMARK(IntComparison);
```

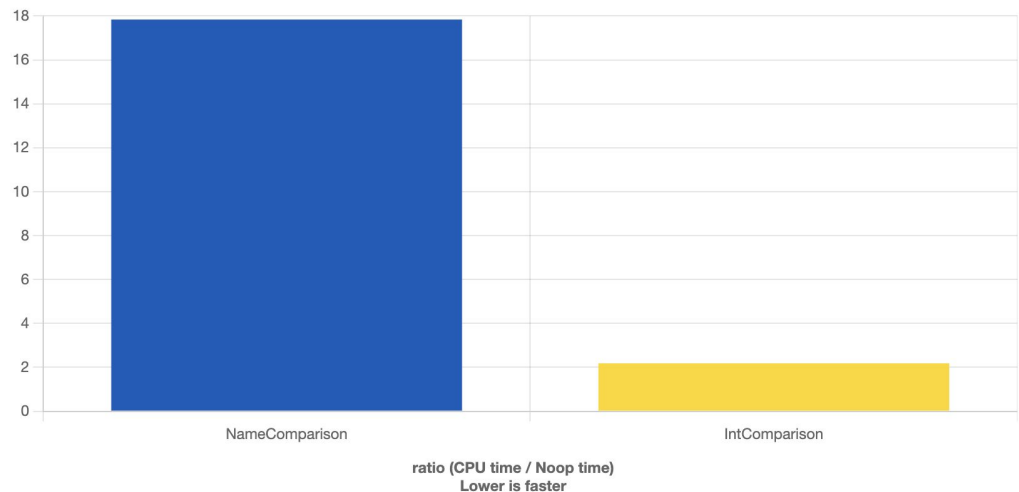
quick-bench.com

compiler = Clang 17.0 ▾ std = c++20 ▾ optim = O3 ▾ STL = libstdc++(GNU) ▾ Other flags ▾

Run Benchmark Disassembly = Intel ▾ ☐ Clear cached results   

Charts

Assembly



 ☐ Show Noop bar

Объединение

```
struct Triangle {  
    Point a;  
    Point b;  
    Point c;  
};
```

```
struct Rect {  
    Point left_top;  
    Point right_top;  
    Point left_bottom;  
};
```

Объединение

```
struct Circle {  
    Point center;  
    float r;  
};
```

```
enum FigureType{  
    Triangle,  
    Rect,  
    Circle,  
};
```

Объединение

```
union FigureU {  
    Triangle triangle;  
    Rect      rect;  
    Circle    circle;  
};
```

```
struct Figure {  
    FigureType type;  
    FigureU    fig;  
};
```

Figure позволяет хранить
одновременно несколько типов фигу

Теперь например можно создавать
массивы таких объектов

Но мы пожертвовали память!