Тема 8: Об'єднання

Виконала студентка 2 курсу
Франчук Олена

Київський національний університет Тараса Шевченка
Механіко-математичний факультет
2 Курс

18 листопада 2024р.

Вступ

Розробка програмного забезпечення на сучасних мовах програмування, таких як C та C++, передбачає використання ефективних інструментів для управління пам'яттю та організації даних. Одними з таких інструментів є `union` у C та `std::variant` у C++. Ці механізми дозволяють ефективно зберігати різні типи даних в одній пам'яті та забезпечувати безпеку під час роботи з цими даними.

`union` є ключовим словом у C, яке дозволяє створювати об'єкти, здатні зберігати значення різних типів, але тільки одне значення за раз. Це забезпечує оптимальне використання пам'яті, але вимагає уважності програміста для уникнення помилок.

З іншого боку, у C++ введено `std::variant`, який не лише дозволяє зберігати дані різних типів, але й додає механізми безпеки, такі як перевірка типу під час виконання. Завдяки цьому інструмент стає важливим для сучасного програмування, дозволяючи створювати надійні й багатофункціональні програми.

У цьому рефераті розглянемо особливості `union` та `std::variant`, їхні переваги та обмеження, а також приклади використання.

Задача 1

У першому коді використовується `union` для зберігання або декартових, або полярних координат:

```
typedef union {
    struct {
        double x;
        double y;
    } cartesian;  % Декартові координати
    struct {
        double r;
        double theta;
    } polar;  % Полярні координати
} Point;
```

Цей фрагмент визначає `union`, що містить дві структури: одну для декартових координат (`x`, `y`), а іншу для полярних координат (`r`, `theta`). Оскільки `union` дозволяє зберігати лише один з типів у будь-який момент часу, це економить пам'ять, зберігаючи лише один набір координат.

У другому коді використовуються `std::variant` для зберігання або декартових, або полярних координат, що має кілька переваг:

```
using Point = std::variant<Cartesian, Polar>;
```

У цьому випадку, `std::variant` дозволяє зберігати або декартові координати (`Cartesian`), або полярні координати (`Polar`). `std::variant` надає безпеку типів, оскільки забезпечує перевірку типу при доступі до значення, що робить програму більш безпечною в порівнянні з `union`. З `std::variant` ми можемо гарантувати, що

значення завжди буде належати одному з типів, і спроба доступу до неправильного типу викличе помилку на етапі компіляції.

Основні відмінності:

- **Типи даних:**
  - У C використовується `union`, що дозволяє зберігати один із кількох можливих типів даних у одному і тому ж місці пам'яті.
  - У C++ використовується `std::variant`, який є типобезпечним контейнером для зберігання одного з кількох типів.
- **Читабельність і безпека:**
  - Код на C менш типобезпечний, оскільки `union` не зберігає інформації про те, який тип наразі активний.
  - Код на C++ використовує `std::get` для отримання значень, що гарантує безпечне приведення типу.
- **Час виконання:** Обидва підходи мають схожий алгоритм для обчислення відстані. Однак обчислення в C++ є трохи повільнішим через додаткову перевірку типу `std::variant`.

Порівняємо час для обчислення:

- C: 0.000002 секунди.
- C++: 0.000019 секунди.

Задача 2

Опис основних фрагментів коду

union Money (C):

```
typedef union {
    HryvniasAndKopecks asHryvnias;
    TotalKopecks asKopecks;
} Money;
```

Використовує `union`, щоб зберігати дві можливі форми представлення грошей: в гривнях та копійках або лише в копійках.
std::variant Money (C++):

```
using Money = std::variant<HryvniasAndKopecks, TotalKopecks>;
```

Використовує `std::variant` для зберігання двох можливих типів даних: `HryvniasAndKopecks` або `TotalKopecks`.

Ці типи дозволяють зберігати грошові значення в різних формах і ефективно працювати з ними в програмі.

1. Функції для перетворення вартості:

- Обидві версії мають аналогічні функції для перетворення між гривнями і копійками, однак у C-версії використовуються звичайні функції, тоді як у C++-версії ці функції працюють із структурами та використовують `std::variant` для більш гнучкого представлення.

2. Виведення даних:

- **C-версія:** Використовує просту функцію `printf`, яка перевіряє, чи потрібно вивести значення у гривнях і копійках або у копійках, використовуючи простий умовний оператор.
- **C++-версія:** Використовує `std::cout` і перевірку типу в `variant` через `std::holds_alterna` що дозволяє ефективніше працювати з різними типами даних.

Порівняємо час для обчислення:

- C: 0.000136 секунди.
- C++: 0.000118 секунди.

Задача 3

union PointOrVector (C):

```
typedef union {
    Point p;
} PointOrVector;
```

За допомогою `union` можна зекономити пам'ять, зберігаючи лише один з типів в певний момент часу.

std::variant<Point> (C++):

```
using PointOrVector = std::variant<Point>;
```

Опис: Використовує `std::variant`, щоб зберігати тип `Point` (загалом може зберігати різні типи, хоча в цьому випадку зберігається лише точка).
`std::variant` дозволяє безпечно працювати з різними типами даних в одній змінній. Це безпечніше, ніж `union`, оскільки компілятор перевіряє тип на момент використання.

Перевірка колінеарності трьох точок:

Опис: Обчислюється детермінант, який перевіряє, чи є три точки колінеарними (якщо детермінант 0, то точки колінеарні).

Функція в C:

```
int areCollinear(Point p1, Point p2, Point p3) { ... }
```

Функція в C++:

```
bool areCollinear(const Point& p1, const Point& p2, const Point& p3) { ...
```

Особливість: Використовується стандартна формула для перевірки колінеарності.

Порівняння версій

Типи даних:

- **C-версія:** Використовує `struct` для точки, тип `Point` з двома полями (`x`, `y`).
- **C++-версія:** Аналогічно, використовує `struct` для точки, але має більше можливостей для розширення.

Функція перевірки колінеарності:

- **C-версія:** Використовує стандартну функцію для обчислення детермінанту (площі), порівнюючи його з нулем.
- **C++-версія:** Логіка така ж, як і версії с.

Час виконання:

- **C-версія:** Вимірює час через `clock()`. І обчислення триває 0.000158 секунд
- **C++-версія:** Час обчислення: 0.000185 секунд.

Задача 4

Опис фрагменту коду: union (C)

union (C):

```
typedef struct {
    enum { CARTESIAN, POLAR, SPHERICAL } type;
    union {
        struct { double x, y, z; } cartesian;
        struct { double r, theta; } polar;
        struct { double r, theta, phi; } spherical;
    } coord;
} Point;
```

Опис: Це структура, яка зберігає точку в різних системах координат (Декартова, Полярна, Сферична) за допомогою `union`. `enum` дозволяє вказати, яка саме система координат використовується. Залежно від цього, у `coord` буде зберігатися відповідна структура з координатами.

Опис фрагменту коду: variant (C++)

variant (C++):

```
using Point = std::variant<Cartesian, Polar, Spherical>;
```

Опис: Використовуємо `std::variant` для зберігання різних типів координат. Це аналог `union` в C++, але з додатковими можливостями для безпечної роботи з різними типами. `Point` може містити або структуру `Cartesian`, або `Polar`, або `Spherical`.

Опис програм

Програма на C:

- **Відстань:** Для кожної системи є окремі функції для обчислення відстані між точками. Для полярних та сферичних координат координати спочатку переводяться в декартову систему.
- **Вимірювання часу:** Використовується функція `clock()` для вимірювання часу виконання.

Програма на C++:

- **Відстань:** Аналогічно до програми на C, кожна система координат має свою функцію для обчислення відстані. Для сферичних координат координати переводяться в декартову систему.
- **Вимірювання часу:** Так само, як у C, використовуються функції для вимірювання часу виконання.

Основні відмінності:

- **Типи даних:** C++ використовує `std::variant`, що зручніше і компактніше.
- **Продуктивність:** Час виконання у обох програмах подібний, але C++ є точнішим завдяки сучасним інструментам для вимірювання часу.

Порівняємо час для обчислення:

- C: декартові: 0.000156 секунд; полярні: 0.000222 секунд; сферичні: 0.000168 секунд;
- C++: декартові:0.000192 секунд; полярні: 0.000256 секунд; сферичні: 0.000198 секунд;.

Задача 5

1. Програма на C

C - union для зберігання фігур:

```
typedef union {
    struct { double radius; } circle;
    struct { double side; } square;
    struct { double a, b, c; } triangle;
    struct { double length, width; } rectangle;
    struct { double a, b, h; } trapezoid;
} Shape;
```

Опис: Це `union`, яке дозволяє зберігати різні типи геометричних фігур в одній змінній. Всі фігури (круг, квадрат, трикутник, прямокутник, трапеція) зберігаються в одному просторі пам'яті, що дозволяє економити місце, але тільки одна фігура може бути використана одночасно.

Програма на C використовує `union` для представлення різних фігур, що дозволяє економити пам'ять. Оскільки всі фігури зберігаються в одному блоці пам'яті, доступ до даних для кожної фігури є швидким. Функції для обчислення площі та периметра використовують `switch`, що дозволяє чітко розподіляти обчислення в залежності від типу фігури.

Переваги:

- Мала витрата пам'яті завдяки використанню `union`.
- Швидкість обчислень через прості структури.

2. Програма на C++

C++ - Використання `variant` для зберігання фігур:

```
using Shape = std::variant<Circle, Square, Triangle, Rectangle, Trapezoid>
```

Опис: `std::variant` дозволяє зберігати один із декількох типів у єдиній змінній. Це заміна для `union` в C++, з додатковими безпечними можливостями. Замість того, щоб мати одну змінну з різними типами, як у `union`, `variant` дає зручний доступ до кожного типу через механізм відвідувачів (*visitors*).

У програмі на C++ використовується `std::variant` для представлення фігур, що дозволяє зберігати різні типи в одному об'єкті і підтримує більш гнучку обробку типів через `std::visit`. Тут застосовуються функціональні об'єкти для обчислення площі та периметра.

Переваги:

- Вища гнучкість через використання `std::variant`.
- Зручніше додавати нові фігури або обчислення, не змінюючи структуру коду.

- `std::visit` дозволяє чисто працювати з типами, що знижує ймовірність помилок.

Швидкість виконання:

У програмах обчислюються однакові операції, однак програма на C має менше накладних витрат і може бути трохи швидшою завдяки простоті структури.

Гнучкість:

Програма на C++ більш гнучка та розширювана, особливо якщо потрібно додавати нові типи фігур.

Задача 6

Опис програм

Програма на C

C - Використання `union`:

```
typedef union {
    int i;
    double d;
    char str[50];
} value;
```

Це визначення `union`, що дозволяє зберігати різні типи даних у одній змінній. В даному випадку, `union` може містити або ціле число (`int`), або число з плаваючою комою (`double`), або рядок символів (`char[50]`). Тільки один з цих типів може бути використаний одночасно, що дозволяє економити пам'ять.

Функції:

- `readNumber`: зчитує числа з консолі з обранням типу.
- `printNumber`: виводить значення в залежності від типу.
- `performOperation`: виконує арифметичні операції, перевіряючи типи.

Програма на C++

C++ - Використання `std::variant`:

```
#include <variant>

using Number = std::variant<int, double, std::string>;
```

Опис: `std::variant` — це шаблонний клас у C++, який дозволяє зберігати кілька різних типів даних в одному об'єкті. У даному випадку, `Number` може зберігати одне з трьох значень:

- `int` — ціле число,

- `double` — число з плаваючою комою,
- `std::string` — рядок символів.

Завдяки `std::variant`, можна зберігати один із типів у змінній одночасно, але лише один тип в даний момент часу.

Функції:

- `readNumber`: зчитує числа з консолі аналогічно.
- `performOperation`: виконує арифметичні операції за допомогою `std::visit` для роботи з різними типами.
- Використовуються `std::optional` для перевірки значення та типу.

Особливості:

- Використання `std::variant` забезпечує більшу гнучкість і зручність порівняно з `union`.
- Для операцій над числами використовується `std::visit`, що є елегантним методом для роботи з різними типами в C++.

Оцінка швидкості

Перша версія програми є трохи швидшою, оскільки не використовує механізмів для обробки варіантів типів, таких як `std::variant` і `std::visit`. Вона безпосередньо працює з типами за допомогою умовних операторів і є більш "легкою".

Друга версія використовує більш складні конструкції C++, що додає певну накладну (особливо через механізм типів `std::variant` і необхідність виклику `std::visit` для обробки різних типів).

Можна зробити такі загальні підсумки щодо розглянутих завдань:

Версія `C` є швидшою завдяки прямому управлінню пам'яттю та простим конструкціям, що дозволяє уникати додаткових накладних витрат. Однак цей підхід менш гнучкий і складніший для розширення, оскільки додавання нових типів вимагає значних змін у коді.

Версія C++ забезпечує більшу гнучкість завдяки використанню `std::variant`, що дозволяє легше додавати нові типи без значних змін у коді. Проте цей варіант має певні накладні витрати через використання шаблонів та абстракцій, що може знизити швидкість виконання.

Загалом, для проектів, де важлива швидкість і мінімальні накладні витрати, краще обирати C. Для більших проектів, де потрібна гнучкість та простота розширення, C++ буде кращим вибором. Однак, якщо проект вимагає обробки різних типів даних і постійних змін у структурі, C++ надає більше можливостей для масштабування без значних змін в основному коді.