Zadanie 7 25–29 kwietnia 2022 r.

kurs języka C++

tablica bitów

Instytut Informatyki
Uniwersytetu Wrocławskiego Paweł Rzechonek

Prolog

Ilościowym aspektem informacji zajmuje się statystyczno–syntaktyczna teoria informacji Hartleya i Shannona. Miary ilości informacji są w niej oparte na prawdopodobieństwie zajścia zdarzenia. Jako miarę ilości informacji przyjmuje się wielkość niepewności, która została usunięta w wyniku zajścia zdarzenia (otrzymania komunikatu). Zdarzenia (komunikaty) mniej prawdopodobne dają więcej informacji.

Bit to najmniejsza jednostka informacji potrzebna do określenia, czy zdarzenie zaszło czy też nie. Bit może przyjąć jedną z dwóch wartości, które zwykle określa się jako *0* (zero) i *1* (jeden), choć można przyjąć dowolne inne nazewnictwo, na przykład *prawda* i *fałsz* albo *tak* i *nie*; w pierwszym przypadku bit jest tożsamy z cyfrą w systemie dwójkowym.

Zadanie

Zdefiniuj klasę `tabbit` reprezentującą tablicę bitów. Najprościej implementuje się taką strukturę danych za pomocą zwykłej tablicy typu `uint16_t[]`, przeznaczając na zapamiętanie bitu całe słowo. Jest to rozwiązanie proste, ale bardzo rozrzutne co do zużywanej pamięci – tablica bitów pamiętana w ten sposób jest kilanaście razy obszerniejsza niż potrzeba. A więc takie rozwiązanie nas nie satysfakcjonuje, szczególnie gdy trzeba posługiwać się w programie wieloma dużymi tablicami (chodzi o tablice przechowujące tysiące a nawet miliony bitów).

Należy zatem tak zaprojektować tablicę bitów, aby przydzielona pamięć była wykorzystana co do bitu (modulo rozmiar słowa). W klasie `tabbit` zdefiniuj operator indeksowania `[]`, który umożliwiałby zarówno czytanie z tablicy, jak również pisanie do niej. Oto fragment kodu, który powinien się skompilować i uruchomić:

```
tab_bit t(46); // tablica 46-bitowa (zainicjalizowana zerami)
tab_bit u(45ull); // tablica 64-bitowa (sizeof(uint64_t)*8)
tab_bit v(t); // tablica 46-bitowa (skopiowana z t)
tab_bit w(tab_bit(8){1, 0, 1, 1, 0, 0, 0, 1}); // tablica 8-bitowa
v[0] = 1; // ustawienie bitu 0-go bitu na 1
t[45] = true; // ustawienie bitu 45-go bitu na 1
```

```
bool b = v[1]; // odczytanie bitu 1-go
u[45] = u[46] = u[63]; // przepisanie bitu 63-go do 45-go i 46-go
cout<<t<<endl; // wyświetlenie zawartości tablicy bitów na ekranie
```

Ponieważ nie można zaadresować pojedynczego bitu (a tym samym nie można ustanowić referencji do niego), więc trzeba się posłużyć specjalną techniką umożliwiającą dostęp do pojedynczego bitu w tablicy. Wystarczy zastosować obiekty niewidocznej dla programisty klasy pomocniczej `ref`, potrafiącej odczytać i zapisać pojedynczy bit w tablicy.

```
class tabbit {
    typedef uint64_t slowo; // komorka w tablicy
    static const int rozmiarSlowa; // rozmiar slowa w bitach
    friend istream & operator >> (istream &we, tab_bit &tb);
    friend ostream & operator << (ostream &wy, const tab_bit &tb);
    class ref; // klasa pomocnicza do adresowania bitów
protected:
    int dl; // liczba bitów
    slowo *tab; // tablica bitów
public:
    // wyzerowana tablica bitow [0...rozm]
    explicit tabbit(int rozm);
    // tablica bitów [0...rozmiarSlowa] zainicjalizowana wzorcem
    explicit tabbit(slowo tb);
    // destruktor
    ~tabbit();
    tabbit(const tabbit &tb); // konstruktor kopiujący
    tabbit(tabbit &&tb); // konstruktor przenoszący
    tabbit& operator = (const tabbit &tb); // przypisanie kopiujące
    tabbit& operator = (tabbit &&tb); // przypisanie przenoszące
private:
    bool czytaj(int i) const; // metoda pomocnicza do odczytu bitu
    bool pisz(int i, bool b); // metoda pomocnicza do zapisu bitu
```

```
public:
    // indeksowanie dla stałych tablic bitowych
    bool operator [] (int i) const;
    // indeksowanie dla zwykłych tablic bitowych
    ref operator [] (int i);
    inline int rozmiar() const; // rozmiar tablicy w bitach
public:
    // operatory bitowe: | i |=, & i &=, ^ i ^= oraz !
};
```

Klasa `ref` jest klasą pomocniczą, której zadaniem jest zaadresowanie pojedynczego bitu w tablicy – zastanów się jak powinna ona być zaimplementowana.

Do kompletu zdefiniuj operatory koniunkcji, alternatywy, różnicy symetrycznej w połączeniu z przypisaniem oraz operator negacji, które będą wykonywać działania na całych tablicach bitów. Niektóre operatory powinny się przyjaźnić z klasą `tabbit` a pozostałe powinny być jej składowymi. Nie zapomnij też o operatorach czytania ze strumienia wejściowego i pisania do strumienia wyjściowego. W konstruktorach oraz w funkcjach i operatorach składowych i zaprzyjaźnionych zgłaszaj błędy za pomocą instrukcji `throw`.

Klasę `tabbit` umieść w przestrzeni nazw `obliczenia`.

Na koniec napisz program, który rzetelnie przetestuje klasę `tabbit` pod kątem inicjalizacji, kopiowania i operacji bitowych (operacje na pojedynczych bitach w tablicy oraz na całych tablicach bitów).

Ważne elementy programu

- Użycie przestrzeni nazw `obliczenia`.
- Konstruktory i destruktor w klasie `tabbit`.
- Wykorzystanie priorytetów operatorów do zminimalizowania liczby wypisywanych nawiasów przez metodę `zapis()`.
- Implementacja semantyki kopiowania w tablicy bitów.
- Operator indeksowania korzystający z pomocniczej klasy `ref` adresującej pojedyncze bity.
- Implementacja operatorów bitowych (alternatywa, różnica symetryczna, koniunkcja i negacja).
- Zgłaszanie wyjątków w konstruktorach i funkcjach składowych.
- W funkcji `main()` należy przetestować obiekty wszystkich klas nieabstrakcyjnych.