laboratorium: zadanie 4 *termin: 7–8 marca 2012 r.*

KURS JĘZYKA C++

TABLICA BITÓW

Instytut Informatyki Uniwersytetu Wrocławskiego *Paweł Rzechonek*

Zadanie.

Zdefiniuj klasę `tab_bit` reprezentującą tablicę bitów. Najprościej implementuje się taką strukturę danych za pomocą zwykłej tablicy typu `int[]`, przeznaczając na zapamiętanie bitu całe słowo. Jest to rozwiązanie proste, ale bardzo rozrzutne co do zużywanej pamięci — tablica bitów pamiętana w ten sposób jest kilka/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 zawierające tysiące a nawet miliony bitów).

Należy zatem tak zaprojektować tablice bitowe, aby przydzielona pamięć była wykorzystywana co do bitu (modulo rozmiar słowa). W klasie `tab_bit` 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(72);          // tablica 72 bitow
t[0] = 1;               // ustawienie bitu 0-ego bitu na 1
t[71] = true;           // ustawienie bitu 71-go bitu na 1
bool b = t[0];          // odczytanie bitu 0-ego
t[40] = b;              // ustawienie bitu 40-go
t[36] = t[38] = t[71]; // przepisanie bitu 71-go do bitow 38-go i 36-go
cout<<t<<endl;          // wysietlenie zawartosci tablicy bitow na ekranie
```

Ponieważ nie można zaadresować pojedynczego bitu (a tym samym nie można ustamowić referencji do niego), więc trzeba się posłużyć specjalną techniką umożliwiającą dostęp do pojedynczego bitu w tablicy. Robi się to poprzez zastosowanie obiektów niewidocznej dla programisty klasy pomocniczej, umiejącej odczytać i zapisać pojedynczy bit.

```
class tab_bit
{
    typedef unsigned long long 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 dla operatora indeksowania
protected:
    int dl; // liczba bitów
    slowo *tab; // tablica bitów
public:
    explicit tab_bit (int rozm);
    tab_bit (const tab_bit &tb);
    tab_bit & operator = (const tab_bit &tb);
    ~tab_bit ();
private:
    bool czytaj (int i) const;
    bool pisz (int i, bool b);
public:
    bool operator[] (int i) const;
    ref operator[] (int i);
    inline int rozmiar () const { return dl; }
};
```

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

Do kompletu podefiniuj operatory koniunkcji, alternatywy, różnicy symetrycznej w połączeniu z przypisaniem oraz operator negacji, które będą wykonywać działania na całych tablicach bitów. Nie zapomnij też o operatorach czytania ze strumienia wejściowego i pisania do strumienia wyjściowego. Wymienione operatory powinny się przyjaźnić z klasą `tab_bit`.

Definicję klasy umieść w pliku `tabbit.h`, a definicje funkcji składowych w pliku `tabbit.cpp`. Program główny w pliku `main.cpp` ma rzetelnie przetestować poprawność zdefiniowanych przez Ciebie operacji na tablicach bitowych i operacji na poszczególnych bitach tablicy.

Uzupełnienie.

Napisz program, który rzetelnie przetestuje klasę `tab_bit` (operacje na poszczególnych bitach tablicy oraz na całych tablicach bitów).

Uwaga 1.

W programie zgłaszaj błędy za pomocą wyjątków (instrukcja `throw string("komunikat o błędzie")`).

Uwaga 2.

Podziel program na pliki nagłówkowe i źródłowe. Definicję klasy umieść w pliku `tabbit.h`, a definicje funkcji składowych w pliku `tabbit.cpp`. Program testujący umieść w pliku `main.cpp`.