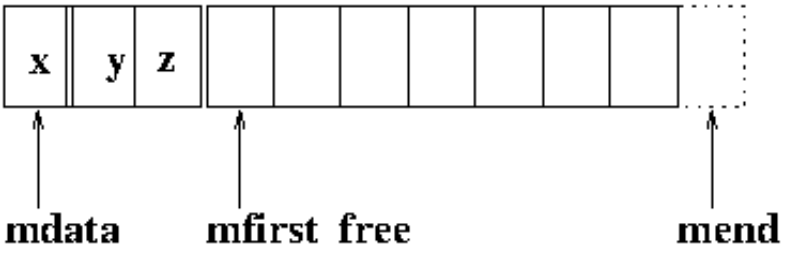
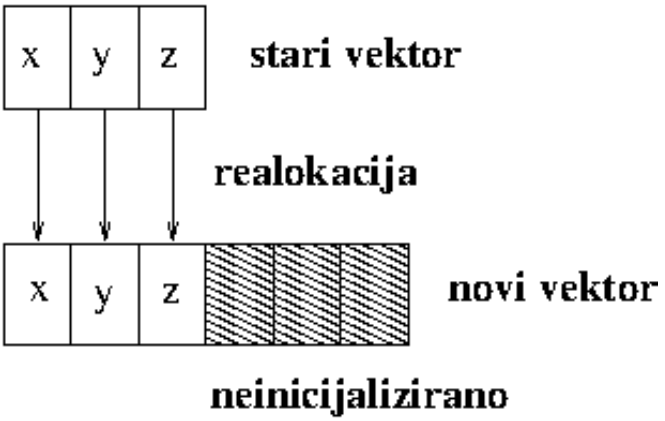
Vektor s realokacijom

Treba implementirati klasu `Vect` tako da dozvoljava metodu `push_back` i realokaciju vektora kada nema mjesta za novi element. Vektor će imati 3 pokazivača: `mdata` - pokazivač na prvi element polja; `mfirst_free` - pokazivač na prvi slobodni element u polju i `mend` - pokazivač na jedno mjesto iza zadnjeg elementa u polju.



Kada nema dovoljno prostora da smjestimo još jedan element vršimo realokaciju. Realociranom vektoru ćemo dati dvostruku duljinu trenutnog vektora (da ne realociramo suviše često). Pri realokaciji stoga jedan dio prostora ispunjavamo starim elementima, a drugi ostavljamo *neinicijaliziranim*. To je zadatak za **alokator**.



Napomena: Budući da funkcija `realocate` koristi alokator i svo ostalo alociranje (i dealociranje) memorije mora ići preko alokatora. To znači da konstruktor, konstruktor kopije i operator pridruživanja moraju alocirati memoriju pomoću alokatora. Stoga ćemo napisati jednu pomoćnu rutinu koja alocira memoriju pomoću alokatora i jednu koja dealocira memoriju, također kroz alokator (alokacija i dealokacija moraju biti *konzistentni*).

Klasa `Vect` mora držati jedan alokator (`std::allocator<double>`) koji ćemo implementirati kao **statičku varijablu**. Sva se alokacija i dealokacija dešava pomoću tog alokatora. Implementacijska odluka je da se dealokacija zatvori u jednu rutinu `free()` koja se može koristiti u destruktoru, operatorima pridruživanja i `reallocate()` funkciji.

Sučelje klase

Sučelje klase i određeni implementacijski detalji su dani ovdje:

```
#ifndef __VECT_REALLOC_H_IS_INCLUDDED__
#define __VECT_REALLOC_H_IS_INCLUDDED__

#include <iostream>
#include <string>

// Klasa Vect implemetira vektor dinamički alociranih double-ova kojima je
```

```
pridružen jedan string
// (ime varijable). Klasa vrši realokaciju kada više nema dovoljno mjesta za novi
element.
// Implementira metodu push_back. Implementirana je potpuna kontrola kopiranja te
neke uobičajene
// metode.
class Vect
{
    public:
        // konstruktor i delegirajuće verzije
        Vect(size_t n, double v, std::string const & ime); // Ctor
        Vect(size_t n, double v) : Vect(n, v, "") {}       // Ctor
        explicit Vect(size_t n) : Vect(n, 0.0, "") {}      // Ctor
        explicit Vect(std::string const & ime) : Vect(0, 0.0, ime) {} // Ctor

        // Kontrola kopiranja
        Vect(const Vect& v);                               // CCtor
        Vect(Vect && v) noexcept;                          // MCtor

        Vect& operator=(const Vect& v);                    // OP
        Vect& operator=(Vect && v) noexcept;               // MOP

        ~Vect(){std::cout << "Dtor\n";  free(); }

        // dohvat elemenata
        double const& operator[](size_t i) const { return mdata[i]; }
        double&       operator[](size_t i)       { return mdata[i]; }

        // push_back kao u std::vector. Eventualno izaziva realokaciju čitavog
vektora
        void push_back(double);

        // info rutine
        size_t  size()     const {return mfirst_free - mdata; }
        size_t  capacity() const {return mend - mdata; }

        // utility-rutine
        double norm2() const;
        Vect& scale(double alpha);
        void print(std::ostream& out) const;

        // Ime vektora je javna varijabla
        std::string  ime;   // ime varijable
    private:
        double *mdata;       // pokazivač na prvi element
        double *mfirst_free; // pokazivač na prvo slobodno mjesto
        double *mend;        // pokazivač na kraj alociranog prostora
                             // (jedno mjesto iza posljednjeg)

        // Vraća true ako imamo prostora za još jedan element, inaće vraća false.
        bool has_space() const {  return ( size() == capacity() ) ?  false : true;
}

        // Oslobodi zauzeti prostor (pomoću alokatora)
        void free();
        // realociraj na novu lokaciju. Povećaj dimenziju vektora 2 puta; ako je
        // vektor prazan dodaj jedan element.
        void reallocate();

        // alokator ne mora imati pristup varijablama članicama klase pa ga činimo
        // statičkim. Ne zaboraviti definirati ga u izvornoj datoteci.
        static std::allocator<double> alloc;

};
#endif
```

Zadatak je dovršiti implementaciju i pri tome označiti sve metode kontrole kopiranja. Kao pomoć ovdje su moguće implementacije nekih metoda:

```
// Konstruktor
// Tek ako je zadano n>0 alociramo memoriju
Vect::Vect(size_t n, double v, std::string const & vec_name) : ime(vec_name),
mdata(nullptr),

mfirst_free(nullptr), mend(nullptr)
{
            std::cerr << "Ctor"<<std::endl;
            if(n > 0) {
                 mdata = alloc.allocate(n);
                 for(size_t i=0; i<n; ++i) alloc.construct(mdata+i, v);
                 mfirst_free = mend = mdata+n;
            }
}

void  Vect::print(std::ostream& out) const
{
    out<< ime << ": (" << size() << "," << capacity()<<") ";
    for(size_t i=0; i < size(); ++i) {
        out << mdata[i];
        if(i+1 < size()) out << ",";
    }
}

void Vect::push_back(double x){
    if( !has_space() )
        reallocate();
    alloc.construct(mfirst_free++, x);
}
```

Konačno, sljedeći glavni program mora **ispravno raditi** s vašom implementacijom.

```
Vect foo(Vect x){
    x.scale(2.0);

    return x;
}

int main()
{
    using std::cout;
    using std::endl;

    Vect a("a");  // prazan vektor

    cout <<"Prazan vektor "; a.print(cout); cout << endl;

    a.push_back(1.0);
    cout<< "Nakon 1. push_back "; a.print(cout); cout << endl;
    a.push_back(2.0);
    cout<< "Nakon 2. push_back "; a.print(cout); cout << endl;
    a.push_back(3.0);
    cout<< "Nakon 3. push_back "; a.print(cout); cout << endl;
    a.push_back(4.0);
    cout<< "Nakon 4. push_back "; a.print(cout); cout << endl;
    a.push_back(5.0);
    cout<< "Nakon 5. push_back "; a.print(cout); cout << endl;

    Vect b(a);
    b.ime ="b";
    b.print(cout); cout << " = "; a.print(cout); cout << endl;
```

```
    Vect c(6, 11.0, "c");
    c.print(cout); cout << endl;

    b = c;
    b.print(cout); cout << "  = "; c.print(cout); cout << endl;

    c = std::move(a);
    c.print(cout); cout << " = "; a.print(cout); cout << endl;  // a opljačkan

    std::cout << "Vect d=foo(c);\n";
    Vect d = foo(c);

    std::cout << "d=foo(c);\n";
    d = foo(b);

    return 0;
}
```

Last Modified: *February 26, 2015*

<>