```cpp
#include <iostream>
#include <string>
#include <list>

using namespace std;

template <typename T>
class vector_dinamico
{
private:
    list<T> fondo;

public:
    class iterador
    {
    private:
        typename list<T>::iterator it;

    public:
        iterador &operator++()
        {
            it++;
            return *this;
        };
        iterador &operator--()
        {
            it--;
            return *this;
        };
        bool operator==(const iterador &otro) const
        {
            return it == otro.it;
        };
        bool operator!=(const iterador &otro) const
        {
            return it != otro.it;
        };
        T &operator*()
        {
            return *it;
        };
        iterador &operator=(const iterador &otro)
        {
            this->it = otro.it;
            return *this;
        };

        friend class vector_dinamico<T>;
    };

    class const_iterador
    {
    private:
        typename list<T>::const_iterator it;

    public:
        const_iterador &operator++()
        {
            it++;
            return *this;
```

```
61          };
62
63          const_iterador &operator--()
64          {
65              it--;
66              return *this;
67          };
68
69          bool operator==(const const_iterador &otro) const
70          {
71              return it == otro.it;
72          };
73
74          const T &operator*() const
75          {
76              return *it;
77          };
78
79          bool operator!=(const const_iterador &otro) const
80          {
81              return it != otro.it;
82          };
83
84          friend class vector_dinamico<T>;
85      };
86      vector_dinamico() = default;
87      const T &back() const;
88      T &back();
89      int size() const;
90      void push_back(const T &);
91      void pop_back();
92      const T &operator[](int) const;
93      T &operator[](int);
94      iterador erase(iterador);
95      bool empty() const;
96      iterador insert(iterador, const T &);
97      void clear();
98      void resize(size_t n, T val=T());
99
100     iterador begin()
101     {
102         iterador i;
103         i.it = fondo.begin();
104
105         return i;
106     };
107
108     iterador end()
109     {
110         iterador i;
111         i.it = fondo.end();
112
113         return i;
114     };
115
116     const_iterador cbegin() const
117     {
118         const_iterador i;
119         i.it = fondo.cbegin();
120
```

```
121         return i;
122     };
123 
124     const_iterador cend() const
125     {
126         const_iterador i;
127         i.it = fondo.cend();
128 
129         return i;
130     };
131 };
132 
133 template <typename T>
134 ostream &operator<<(ostream &f, const vector_dinamico<T> &vec)
135 {
136     typename vector_dinamico<T>::const_iterador it;
137     for(it=vec.cbegin();it!=vec.cend();++it)
138     {
139         f << (*it) << " ";
140     }
141     f << endl;
142     return f;
143 }
144 
145 template <typename T>
146 const T &vector_dinamico<T>::back() const
147 {
148     return this->fondo.back();
149 }
150 
151 template <typename T>
152 T &vector_dinamico<T>::back()
153 {
154     return this->fondo.back();
155 }
156 
157 template <typename T>
158 int vector_dinamico<T>::size() const
159 {
160     return this->fondo.size();
161 }
162 
163 template <typename T>
164 void vector_dinamico<T>::push_back(const T &x)
165 {
166     this->fondo.push_back(x);
167 }
168 
169 template <typename T>
170 void vector_dinamico<T>::pop_back()
171 {
172     this->fondo.pop_back();
173 }
174 
175 template <typename T>
176 const T &vector_dinamico<T>::operator[](int pos) const
177 {
178     typename list<T>::const_iterator it=this->fondo.begin();
179     int i=0;
180     while(i<pos)
```

```
    {
        it++;
        i++;
    }
    return *it;
}

template <typename T>
T& vector_dinamico<T>::operator[](int pos)
{
    typename list<T>::iterator it=this->fondo.begin();
    int i=0;
    while(i<pos)
    {
        it++;
        i++;
    }
    return *it;
}

template <typename T>
typename vector_dinamico<T>::iterador vector_dinamico<T>::erase(iterador it)
{
    typename list<T>::iterator aux;
    aux=this->fondo.erase(it.it);
    iterador ret;
    ret.it=aux;
    return ret;
}

template <typename T>
bool vector_dinamico<T>::empty() const
{
    return this->fondo.empty();
}

template <typename T>
typename vector_dinamico<T>::iterador vector_dinamico<T>::insert(iterador it, const T
&nuevo)
{
    typename list<T>::iterator aux;
    aux=it.it;
    aux=this->fondo.insert(aux,nuevo);
    iterador ret;
    ret.it=aux;
    ++ret;
    return ret;
}

template <typename T>
void vector_dinamico<T>::clear()
{
    this->fondo.clear();
}

template <typename T>
void vector_dinamico<T>::resize(size_t n, T val)
{
    this->fondo.resize(n,val);
}
```

```
240
241 int main(){
242     vector_dinamico<int> prueba1;
243     prueba1.push_back(8);
244     vector_dinamico<int>::iterador it;
245     it=prueba1.begin();
246     ++it;
247     for(int i=0;i<5;i++)
248     {
249         it=prueba1.insert(it,i);
250     }
251     cout << "PRIMERA PRUEBA DE INSERT ->" << prueba1;
252     prueba1.pop_back();
253     cout << "PRUEBA DE POP_BACK ->" << prueba1;
254     prueba1.clear();
255     cout << "PRUEBA DE CLEAR ->"<< prueba1;
256     prueba1.resize(10);
257     for(int i=0;i<10;i++)
258     {
259         prueba1[i]=i;
260     }
261     cout << "INTRODUCCION CON []->" << prueba1;
262     it=prueba1.begin();
263     for(int i=90;i<95;i++)
264     {
265         it=prueba1.insert(it,i);
266         ++it; //PARA INTRODUCIR SALTEADO
267     }
268     cout << "SEGUNDA PRUEBA DE INSERT ->" <<  prueba1;
269     it=prueba1.begin();
270     for(it;it!=prueba1.end();)
271     {
272         it=prueba1.erase(it);
273     }
274     cout << "PRUEBA DE ERASE ->"<< prueba1 ;
275 }
```