

Author: Dr. S. A. Norman.

Electronic copies of handouts for L02 and T02 can be found at
<http://www.enel.ucalgary.ca/People/Norman/encm339fall2005/>

Exercise 1: References.

Make a diagram for point one in the following program.

```
void c(const int& ts, int& m, int& s);
void d(int x, int& y, int& z);
```

```
int main()
{
    int j, k;
    d(2 * 60 + 17, j, k);
    return 0;
}
```

```
void c(const int& ts, int& m, int& s)
{
    m = ts / 60;
    s = ts % 60;

    // point one
}
```

```
void d(int x, int& y, int& z)
{
    c(x, y, z);
}
```

Exercise 2: Studying a simple class type

The following three files (one on this page and two on the other page) show a class that behaves somewhat like the `vector<int>` type used in ENGG 233. The `SimpleVector` type has very limited functionality—it's too simple for practical use—but it provides examples of many basic features of class types.

Predict the program output, and make memory diagrams for points one and two. (In the diagrams, you may leave out the string constants.)

```
// file: SimpleVector.h
#ifndef SIMPLEVECTOR_H
#define SIMPLEVECTOR_H

// Maximum number of elements available in a SimpleVector.
const int MAX_SIZE = 5;

class SimpleVector {
public:
    SimpleVector();
    SimpleVector(int init_size);
    SimpleVector(const int *a, int init_size);

    int size() const;
    int get_element(int index) const;
    bool is_identical_to(const SimpleVector& other) const;

    void set_element(int index, int new_value);
    void resize(int new_size);

private:
    int sizeM;
    int storageM[MAX_SIZE];
};
#endif
```

```

// file: SimpleVector.cpp
#include "SimpleVector.h"

SimpleVector::SimpleVector()
    : sizeM(0)
{ }

SimpleVector::SimpleVector(int init_size)
    : sizeM(init_size)
{ }

SimpleVector::SimpleVector(const int *a, int init_size)
    : sizeM(init_size)
{
    for (int i = 0; i < sizeM; i++)
        storageM[i] = a[i];
}

int SimpleVector::size() const
{
    return sizeM;
}

int SimpleVector::get_element(int index) const
{
    return storageM[index];
}

bool SimpleVector::is_identical_to(const SimpleVector& other) const
{
    // point one

    if (sizeM != other.sizeM)
        return false;
    for (int i = 0; i < sizeM; i++)
        if (storageM[i] != other.storageM[i])
            return false;
    return true;
}

void SimpleVector::set_element(int index, int new_value)

```

```

{
    storageM[index] = new_value;
}

void SimpleVector::resize(int new_size)
{
    sizeM = new_size;
}

```

```

// file: main.cpp
#include <iostream>
using namespace std;

#include "SimpleVector.h"

int main()
{
    bool b;
    int x[] = { 11, 22, 33, 44 };
    SimpleVector vec1(x, 4);
    SimpleVector vec2(3);
    cout << "elements of vec2:\n";
    for (int i = 0; i < vec2.size(); i++)
        cout << " " << vec2.get_element(i) << endl;
    vec2.set_element(2, 33);
    vec2.set_element(0, 11);
    vec2.set_element(1, 22);
    vec1.resize(3);
    b = vec1.is_identical_to(vec2);

    // point two

    cout << "elements of vec2:\n";
    for (int i = 0; i < vec2.size(); i++)
        cout << " " << vec2.get_element(i) << endl;
    return 0;
}

```