

int studentId00 = 9000; int studentId20 = 9020; int studentId40 = 9040; int studentId60 = 9060;  
int studentId01 = 9001; int studentId21 = 9021; int studentId41 = 9041; int studentId61 = 9061;  
int studentId02 = 9002; int studentId22 = 9022; int studentId42 = 9042; int studentId62 = 9062;  
int studentId03 = 9003; int studentId23 = 9023; int studentId43 = 9043; int studentId63 = 9063;  
int studentId04 = 9004; int studentId24 = 9024; int studentId44 = 9044; int studentId64 = 9064;  
int studentId05 = 9005; int studentId25 = 9025; int studentId45 = 9045; int studentId65 = 9065;  
int studentId06 = 9006; int studentId26 = 9026; int studentId46 = 9046; int studentId66 = 9066;  
int studentId07 = 9007; int studentId27 = 9027; int studentId47 = 9047; int studentId67 = 9067;  
int studentId08 = 9008; int studentId28 = 9028; int studentId48 = 9048; int studentId68 = 9068;  
int studentId09 = 9009; int studentId29 = 9029; int studentId49 = 9049; int studentId68 = 9069;  
int studentId10 = 9010; int studentId30 = 9030; int studentId50 = 9050; int studentId70 = 9070;  
int studentId11 = 9011; int studentId31 = 9031; int studentId51 = 9051; int studentId71 = 9071;  
int studentId12 = 9012; int studentId32 = 9032; int studentId52 = 9052; int studentId72 = 9072;  
int studentId13 = 9013; int studentId33 = 9033; int studentId53 = 9053; int studentId73 = 9073;  
int studentId14 = 9014; int studentId34 = 9034; int studentId54 = 9054; int studentId74 = 9074;  
int studentId15 = 9015; int studentId35 = 9035; int studentId55 = 9055; int studentId75 = 9075;  
int studentId16 = 9016; int studentId36 = 9036; int studentId56 = 9056; int studentId76 = 9076;  
int studentId17 = 9017; int studentId37 = 9037; int studentId57 = 9057; int studentId77 = 9077;  
int studentId18 = 9018; int studentId38 = 9038; int studentId58 = 9058; int studentId78 = 9078;  
int studentId19 = 9019; int studentId39 = 9039; int studentId59 = 9059; int studentId79 = 9079;

I figured out how to  
store 80 student  
numbers!!!!

# Vectors

Slides #13 – Chapter 8.3

CMPT 130 © Dr. B. Fraser

# Topics

---

- 1) How can we store many values at once?
- 2) How can we pass vectors to functions?
- 3) How can we copy/compare vectors?

# Vectors

## Part 1

How to store many values?

# Vector

- Vector Object:
  - 
  - Can dynamically grow and shrink, and report its size.

prices = 

	0	1	2	3
	20	5	2	8

# Vector example

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    // Create a vector of double
    vector<double> myFavNums;

    // Insert my favourite numbers
    myFavNums.push_back(42);
    myFavNums.push_back(-2.5);
    myFavNums.push_back(3.141590000);

    // Print out the three numbers
    cout << "Num 1: " << myFavNums.at(0) << endl;
    cout << "Num 2: " << myFavNums.at(1) << endl;
    cout << "Num 3: " << myFavNums.at(2) << endl;

    return 0;
}
```

Must include <vector> and name-space std.

When created, must specify type of values it will hold...

Add an element to the vector with:...

Use .at(*n*) to access element *n*. Ex:  
double k = data.at(*i*);

Num 1: 42  
Num 2: -2.5  
Num 3: 3.14159

# Vectors

- Vector is in the Standard Template Library (STL):
  - STL is programmer-created data types and algorithms (not part of 'core' C++).
  - It is a template class:  
It can be used to hold...
- Specify type of data to hold when creating vector:  
vector<int>                    ages;  
vector<double>                heights;  
vector<string>                names;  
vector<char>                   firstInitials;

# Initializing a Vector

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    // Option #1:..

    vector<int> prices;
    prices.push_back(20);
    prices.push_back(5);

    // Option #2:..

    vector<int> daysPerMonth {31, 28, 31, 30, 31, 30, 31,
                             31, 30, 31, 30, 31};

    // Display all values (explanation coming on next slide!)
    for (int i = 0; i < daysPerMonth.size(); i++) {
        cout << i << ": " << daysPerMonth.at(i) << endl;
    }
}
```

## For Loop Generates a Warning:

"Comparison between signed and unsigned"

Explanation: myVector.size() is unsigned.

Fix: for (unsigned int i = 0; i < myVector.size(); i++) {...}

# Vector Element Access

Vector object  
daysPerMonth

- Direct access to any element:
  - For N elements...

```
daysPerMonth.at(0) = 31;    // January  
Pronounced...
```

- Ex:

```
daysPerMonth.at(11) = 31;    // December  
int a = daysPerMonth.at(1);    // February  
int guess = daysPerMonth.at(i + 1); // Depends on i.  
cout << daysPerMonth.at(1);    // Outputs 28  
cin >> daysPerMonth.at(9);    // Read in oct.
```

	Idx	Val
Jan	0	31
Feb	1	28
Mar	2	31
Apr	3	30
May	4	31
Jun	5	30
Jul	6	31
Aug	7	31
Sep	8	30
Oct	9	31
Nov	10	30
Dec	11	31



# Vector Indices vs Values

- An element's value and its index are different:

```
vector<int> prices {1, 5, 12, 20};
```

```
prices =
```

0	1	2	3
1	5	12	20

- Add 2 elements:

```
int a = prices.at(1) + prices.at(2);//
```

- Add 2 indices:

```
int b =prices.at(1 + 2);           //
```

# Vector methods

## Function

`myVect.at(i)`

or use:

`myVect[i]`

`myVect.clear()`

`myVect.empty()`

`myVect.pop_back()`

`myVect.push_back(42)`

`myVect.size()`

## Description

Access the *i*'th element of `myVect` (where *i* is an integer)

Ex: `int val = myVect.at(i);`

Ex: `myVect.at(i) = 77;`

Removes all elements from the vector.

Returns true if the vector is empty, false otherwise.

Removes the last element from the vector.

Adds the number 42 to the end of the vector. The value must match vector type.

`.at(i)` vs `[i]` are similar; however `.at(i)` is safer (more later).

# Vector example: Hours worked

```
int main() {
    const int DAYS_PER_WEEK = 7;
    // Create the vector for hours per day.
    vector<float> hoursWorked;

    // Ask user for time worked.
    for (int i = 0; i < DAYS_PER_WEEK; i++) {
        cout << "Hrs worked on day #" << i << ": ";
        float hours = 0;
        cin >> hours;
        hoursWorked.push_back(hours);
    }

    // Calculate total hours
    cout << "Week summary:\n";
    float totalHours = 0;
    for (int i = 0; i < DAYS_PER_WEEK; i++) {
        cout << fixed << setprecision(2);
        cout << "\t " << i << " = " << setw(5) << hoursWorked.at(i) << " hours\n";
        totalHours += hoursWorked.at(i);
    }
    cout << "Total hours: " << totalHours << endl;
}
```

```
Hrs worked on day #0: 0
Hrs worked on day #1: 1.5
Hrs worked on day #2: 26.9
Hrs worked on day #3: 8.2
Hrs worked on day #4: 1.6
Hrs worked on day #5: 0
Hrs worked on day #6: 0
Week summary:
    0 = 0.00 hours
    1 = 1.50 hours
    2 = 26.90 hours
    3 = 8.20 hours
    4 = 1.60 hours
    5 = 0.00 hours
    6 = 0.00 hours
Total hours: 38.20
```

# Suggested Exercise

- Change the hoursWorked:
  - When reading in hours worked, display the day name (hint Vector!)
  - Stop reading in values when the user enters -1.
- Additional
  - Calculate max # hours on a day. Or, find the day which has max hours worked.

```
Hours worked on Sunday: 0
Hours worked on Monday: 8.2
Hours worked on Tuesday: 5
Hours worked on Wednesday: -4
Hours worked on Thursday: -1
Week summary:
    Sunday = 0.00 hours
    Monday = 8.20 hours
    Tuesday = 5.00 hours
    Wednesday = -4.00 hours
Total hours: 9.20
Max hours in single day: 8.20
```

# Review

---

- Write some code which creates a vector to hold characters and insert the first 2 letters of your name.
  
  
  
  
  
  
  
  
  
  
- Write a loop to output the contents of the above vector. Do not hardcode the size!

# Vectors

## Part 2

Passing vectors to functions  
without making a copy!

# Passing Vector to Function

```
#include <iostream>
#include <vector>
using namespace std;

void printVector(vector<int> data) {
    for (unsigned int i = 0; i < data.size(); i++) {
        cout << data.at(i) << ", ";
    }
    cout << endl;
}

void doubleVector(vector<int> data) {
    for (unsigned int i = 0; i < data.size(); i++) {
        data.at(i) = 2 * data.at(i);
    }
}

int main() {
    vector<int> salaries {10'000, 20'000, 15'000};

    cout << "Initial pay:      ";
    printVector(salaries);

    // Give a big raise!
    doubleVector(salaries);

    cout << "After big raises: ";
    printVector(salaries);
}
```

Program does not  
(yet) work!

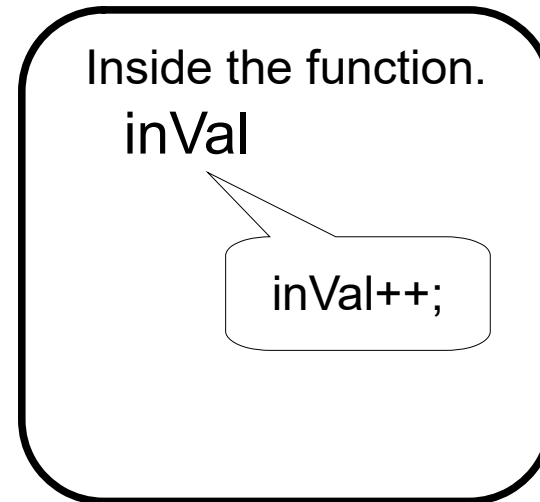
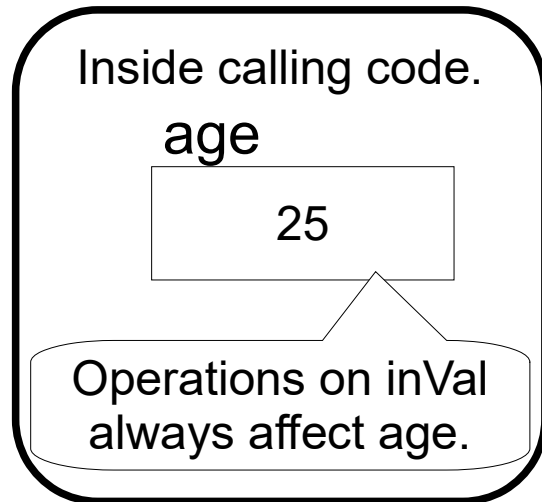
Output

```
Initial pay:      10000, 20000, 15000,
After big raises: 10000, 20000, 15000,
```

Can use ` to separate  
digit groups

# Explaining pass by reference

- Reference:
  - One variable is an alias of another variable...
- When using Pass-by-Reference:  
function's parameter refers to the actual argument.
  - Changing the parameter's value...





# Pass by reference

- To pass-by-reference, put an & between the parameter's type and name in the parameter list.
  - This makes the function's parameter an alias for the calling argument.

```
void growOlder(int &inVal) {  
    inVal++;  
}  
  
int main () {  
    int age = 25;  
    growOlder(age);  
    cout << "Age is:   " << age << endl;  
}
```

say: "inVal is a  
reference to an int."

Output

# Passing Vector to Function (Again!)

```
#include <iostream>
#include <vector>
using namespace std;

void doubleVector(vector<int> &data) {
    for (unsigned int i = 0; i < data.size(); i++) {
        data.at(i) = 2 * data.at(i);
    }
}

void printVector(vector<int> data) {
    for (int val : data) {
        cout << val << ", ";
    }
    cout << endl;
}

int main() {
    vector<int> salaries {10'000, 20'000, 15'000};

    cout << "Initial pay:      ";
    printVector(salaries);

    // Give a big raise!
    doubleVector(salaries);

    cout << "After big raises: ";
    printVector(salaries);
}
```

## Output

```
Initial pay:      10000, 20000, 15000,
After big raises: 20000, 40000, 30000,
```

# Uses for pass-by-reference

- Useful for passing back multiple values:  
// Return true if successfully read first and last names.  
// Otherwise, return false.  
bool readName(string &first, string &last);
- Cautions on Use:
  - Use pass-by-value as much as possible!
  - Use a return value to pass back a single value.
  - Arguments for pass-by-reference...

- Ex:

```
string a, b;  
readName(a, b);           // Good  
readName("Hello", "World"); // Compile Error.
```

# Passing elements

- Single elements of a vector can be passed to function...

```
void showChar(char ch) {
    cout << "Element: " << ch << endl;
}
void changeChar(char &ch) {
    char newVal = 'x';
    cout << "Changing " << ch << " to "
        << newVal << "." << endl;
    ch = newVal;
}
```

```
int main () {
    vector<char> greeting {'H', 'i', '!'};

    // Pass an element by value.
    showChar( greeting.at(0) );

    // Pass an element by reference.
    changeChar( greeting.at(0) );
    showChar( greeting.at(0) );

    ...
}
```

# Passing a whole vector

- You can pass a `vector` to a function using pass by value, or pass by reference.

```
void changeA( vector<int> data ) {  
    data.push_back(42);  
}
```

```
void changeB( vector<int> &data) {  
    data.push_back(1337);  
}
```

```
int main () {  
    // Create the vectors  
    vector<int> ages {10};  
  
    // Pass by value example  
    changeA(ages);  
  
    // Pass by reference example  
    changeB(ages);  
    ...  
}
```

# Working with Vectors

# Copy and Compare

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<double> grades {95.2, 56.1, 4.0, 88.5};

    // Copy an existing vector (element by element):
    vector<double> copy = grades;

    // Check if two have identical elements:
    if (grades == copy) {
        cout << "Same!" << endl;
    } else {
        cout << "Not the same!" << endl;
    }
}
```

- Vector “overloads”  
= and == to do..

- Makes it easy to  
work with!

Sample Output:  
Same!

# Out of Bounds

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<double> grades {95.2, 56.1, 4.0, 88.5};

    // [] lets you..
    grades[6] = 12.345;
    cout << "Testing out of bounds:" << endl;
    for (int i = 0; i < 10; i++) {
        cout << i << " = " << grades[i] << endl;
    }

    // Use grades.at(i) function instead of grades[i]
    cout << "Out of bounds execption: " << grades.at(10);
    cout << "Done!" << endl;
}
```

Testing out of bounds:

0 = 95.2

1 = 56.1

2 = 4

3 = 88.5

4 = 0

5 = 2.42092e-322

6 = 12.345

7 = 7.3067e-251

8 = 4.8671e-306

9 = 2.122e-314

Generates a runtime error (exception).

Why is this good?

terminate called after throwing an instance of 'std::out\_of\_range'  
what():  
vector::\_M\_range\_check



# Sample Program

- Write a complete C++ program which:
  - Reads in course percentages from the user (doubles) into a vector.
  - Has a function to compute pass/fail grades for each student (pass = 65% or more)
  - Display a table of results like:

#1	82.5%	P
#2	59.0%	F
...		
  - *Optional:* Before displaying, call a function which clamps all percentages to between [0%, 100%] (for example, a grade of 103% becomes 100%).

# Personal Review Questions

---

- Write a function which returns the largest value stored in a vector of integers.
  - Write a program to test it (different length vectors, positive and negative numbers).
- Write a function which returns the *index* of the largest value stored in a vector of integers.
  - Test as before.

# Summary

- C++ vectors store many items of the same type.
  - Can grow & shrink.
- Passing to functions
  - Pass by value: passes in a copy.
  - Pass by reference: passes in the real variable.
  - Can pass whole vector, or just elements.
- Working with Vectors
  - Copy and compare with = and ==
  - Out of bounds