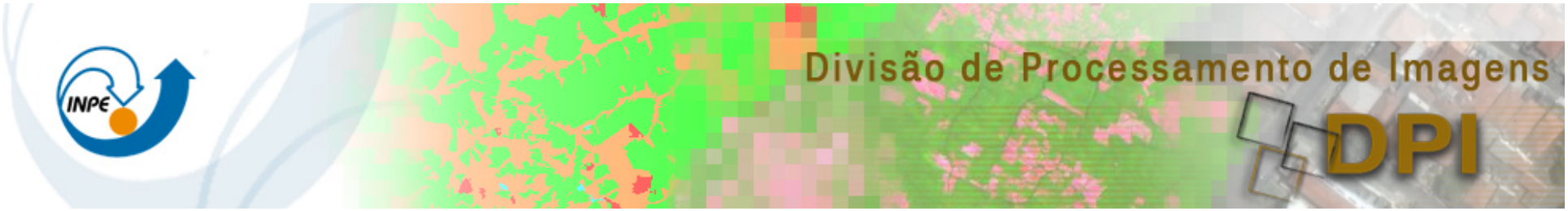


CAP- 390-1 Fundamentos de Programação Estruturada

Lubia Vinhas

Class 3 – More computation

- Each programming language feature exists to express a fundamental idea
  - For example
    - + : addition
    - * : multiplication
    - if (expression) statement else statement ; selection
    - while (expression) statement ; iteration
    - f(x); function/operation
    - …
- We combine language features to create programs

Expressions

```
// compute area:
int length = 20;        // the simplest expression: a literal (here, 20)
                        // (here used to initialize a variable)
int width = 40;
int area = length*width;            // a multiplication
int average = (length+width)/2; // addition and division
```

The usual rules of precedence apply:

a*b+c/d means **(a*b)+(c/d)** and not **a*(b+c)/d**.

If in doubt, parenthesize. If complicated, parenthesize.
Don't write "absurdly complicated" expressions:

a*b+c/d*(e-f/g)/h+7 **//** *too complicated*

Choose meaningful names.

Expressions

- Expressions are made out of operators and operands
  - Operators specify what is to be done
  - Operands specify the data for the operators to work with

- Boolean type: **bool** (**true** and **false**)
  - Equality operators: **= =** (equal), **!=** (not equal)
  - Logical operators: **&&** (and), **||** (or), **!** (not)
  - Relational operators: **<** (less than), **>** (greater than), **<=**, **>=**
- Character type: **char** (e.g., **'a'**, **'7'**, and **'@'**)
- Integer types: **short, int, long**
  - arithmetic operators: **+, -, *, /, %** (remainder)
- Floating-point types: e.g., **float, double** (e.g., **12.45** and **1.234e3**)
  - arithmetic operators: **+, -, *, /**

Concise operators

- For many binary operators, there are (roughly) equivalent more concise operators
  - For example
    - **a += c** means **a = a+c**
    - **a *= scale** means **a = a*scale**
    - **++a** means **a += 1**
      or **a = a+1**
  - "Concise operators" are generally better to use (clearer, express an idea more directly)

Statements

- A statement is
  - an expression followed by a semicolon, or
  - a declaration, or
  - a "control statement" that determines the flow of control

- For example
  - **a = b;**
  - **double d2 = 2.5;**
  - **if (x == 2) y = 4;**
  - **while (cin >> number) numbers.push_back(number);**
  - **int average = (length+width)/2;**
  - **return x;**

Selection

- Sometimes we must select between alternatives
- For example, suppose we want to identify the larger of two values. We can do this with an **if** statement

```
if (a<b)
    max = b;
else
    max = a;
```

- The syntax is

```
if (condition)
    statement-1     // if the condition is true, do statement-1
else
    statement-2     // if not, do statement-2
```

Iteration

- The world's first "real program" running on a stored-program computer (David Wheeler, Cambridge, May 6, 1949)

```
// calculate and print a table of squares 0-99:
int main()
{
        int i = 0;
        while (i<100) {
                cout << i << '\t' << square(i) << '\n';
                ++i ;        // increment i
        }
}
```

Iteration (for loop)

- Another iteration form: the **for** loop
- You can collect all the control information in one place, at the top, where it's easy to see

```
for (int i = 0; i<100; ++i)
{
    cout << i << '\t' << square(i) << '\n';
}
```

That is,

for (initialize**;** condition **;** increment **)**
controlled statement

Functions

- A call of the function **square()**

```
int square(int x)
{
    return x*x;
}
```

- We define a function when we want to separate a computation because it
  - is logically separate
  - makes the program text clearer (by naming the computation)
  - is useful in more than one place in our program
  - eases testing, distribution of labor, and maintenance

Another example

- Here is a function that compares the two values and returns the larger value.

```
int max(int a, int b)  // this function takes 2 parameters
{
        if (a<b)
                return b;
        else
                return a;
}

int x = max(7, 9);              // x becomes 9
int y = max(19, -27);           // y becomes 19
int z = max(20, 20);            // z becomes 20
```

Data for Iteration - Vector

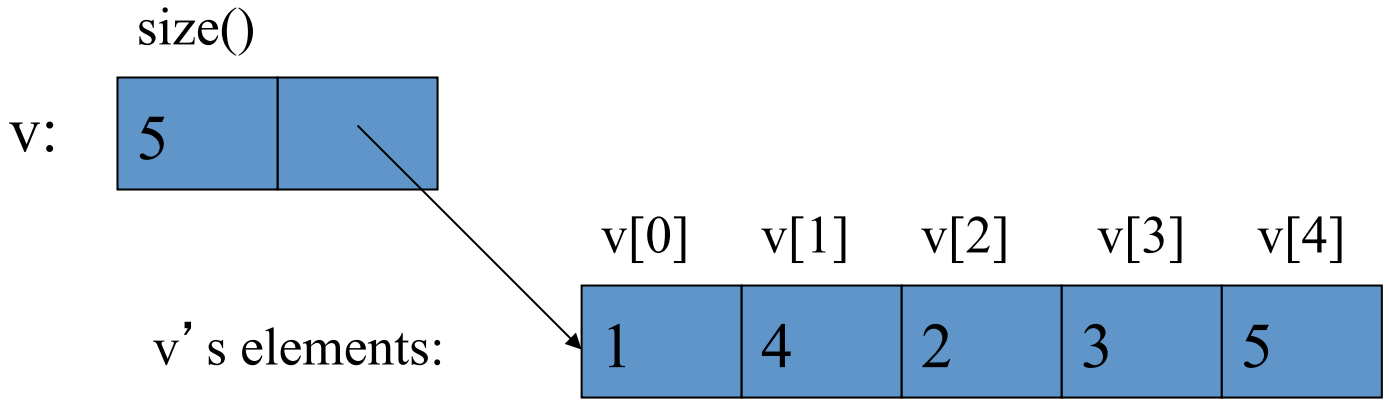
- To do just about anything of interest, we need a collection of data to work on. We can store this data in a **vector**. For example:

```
// read some temperatures into a vector:
int main()
{
        vector<double> temps;       // declare a vector of type double to store
                                    // temperatures – like 62.4
        double temp;        // a variable for a single temperature value
        while (cin>>temp)       // cin reads a value and stores it in temp
                temps.push_back(temp);  // store the value of temp in the vector
        // ... do something ...
}
// cin>>temp  will return true until we reach the end of file or encounter
// something that isn't a double: like the word "end"
```

Vector

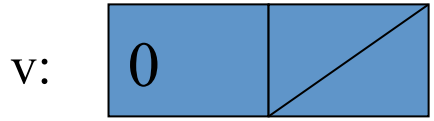
- Vector is the most useful standard library data type
  - a **vector<T>** holds an sequence of values of type **T**
  - Think of a vector this way

    A vector named **v** contains 5 elements: {1, 4, 2, 3, 5}

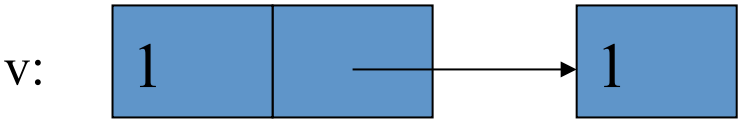


Vector

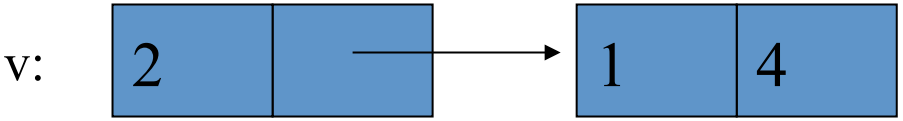
vector<int> v; // start off empty



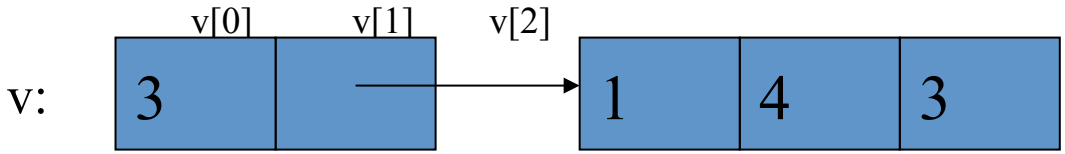
v.push_back(1); // add an element with the value **1**



v.push_back(4); // add an element with the value **4** at end (“the back”)



v.push_back(3); // add an element with the value **3** at end (“the back”)



Vectors

```
// compute mean (average) and median temperatures:
int main()
{
    vector<double> temps;     // temperatures in Fahrenheit, e.g. 64.6
    double temp;
    while (cin>>temp)  temps.push_back(temp); // read and put into vector
    double sum = 0;
    for (int i = 0; i< temps.size(); ++i) sum += temps[i];
                                  // sums temperatures

    cout << "Mean temperature: " << sum/temps.size() << endl;
    sort(temps.begin(),temps.end());
    cout << "Median temperature: " << temps[temps.size()/2] << endl;
}
```

Combining Language Features

- You can write many new programs by combining language features, built-in types, and user-defined types in new and interesting ways.
  - So far, we have
    - Variables and literals of types **bool, char, int, double**
    - **vector, push_back(), [ ]** (subscripting)
    - **!=, ==, =, +, -, +=, <, &&, ||, !**
    - **max( ), sort( ), cin>>, cout<<**
    - **if, for, while**

Example – word list

```
vector<string> words;
string s;
while (cin>>s && s != "quit")            // && means AND
      words.push_back(s);

sort(words.begin(), words.end()); // sort the words we read

for (int i=0; i<words.size(); ++i)
      cout<<words[i]<< "\n";

/*
   read a bunch of strings into a vector of strings, sort
   them into lexicographical order (alphabetical order),
   and print the strings from the vector to see what we have.
*/
```

More exercises..

- Re-write the function to get rid of duplicate words