Outline

- About C/C++ language
- Getting started
- Data types (arrays, strings, pointers)
- Functions
- About this course
- Self test exercises

C/C++ vs. Java

<table>
<thead>
<tr>
<th>C/C++</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>For</td>
<td>systems programming</td>
</tr>
<tr>
<td>Advantage</td>
<td>economy of expression, compact code, flexibility</td>
</tr>
<tr>
<td>Disadvantage</td>
<td>too concise for human understanding, lack of run-time checks, flexibility</td>
</tr>
<tr>
<td>Conclusion</td>
<td>FAST, FLEXIBLE, BUT DANGEROUS</td>
</tr>
</tbody>
</table>

Legacy: Any C program is a legal C++ program.
C++ programming methodologies

- **procedural programming**: input -> processing -> output
- **object oriented programming**: encapsulate data and actions in objects.
  Objects are bundles of data that know how to do things to themselves.

Cat is a class.
Tommy, Felix are objects of type Cat
Tommy.Age = 4 ;
Tommy.walk();

- **generic programming**: make functions and classes, called templates, for any type of the arguments
  T max (T left, T right) // where T can be any type.

---

A simple C++ program (example1.cpp)

```cpp
#include <iostream>

int main() {
  int a, b, c;
  cout << "Give 3 integer numbers: ";
  cin >> a >> b >> c;
  int sum = a + b + c;
  cout << "The sum is: " << sum << endl;
  return 0;
}
```

**Explanation**

//example1.cpp

/* Reads 3 numbers from the keyboard, sums them and writes the sum on the screen. */
#include <iostream>
int main() {
  int a, b, c;
  cout << "Give 3 integer numbers: ";
  cin >> a >> b >> c;
  int sum = a + b + c;
  cout << "The sum is: " << sum << endl;
  return 0;
}

---

Outline

- **About C/C++ language**
- **Getting started**

Getting started

- **Under Unix**:
  - use `g++`, the GNU C/C++ compiler
  - Compile: `g++ example1.cpp`
  - Run: `a.out` (The executable is named by default `a.out`)
  - You can compile and specify another name for the executable: `g++ example1.cpp -o example1.exe`
  - Run: `example1.exe`

- **Under Windows**:
  - use Microsoft Visual C++: Compile - Build - Execute
  - Details can be found in the practical guide on the course website.
  - Reading: Book Prata, ch.1 Getting started

---

What happens to a C++ program?

- `g++` is a compiler that compiles your code.
- `make` is a build tool that builds your program.
- `gcc` is another compiler that compiles your code.
- `ld` is a linker that links your code.
- `objdump` is a disassembler that disassembles your code.
- `strip` is a strip tool that removes debugging information from your code.

---

Console I/O: `#include <iostream>`

- `cin` is an input stream usually connected to the keyboard.
  Example: `cin >> number;`
- `cout` is an output stream usually connected to the screen.
  Example: `cout << "The number is " << number;`
Flow control statements
- if - else
- switch – break
- while loop – break, continue
- do-while loop
- for loop
- goto – not recommended (not in Java!)
- Read more in book Prata ch. 5,6

Outline
- About C/C++ language
- Getting started
- Data types

Preface
- We discuss only C++ specific things.
  - The rest is like in Java
  - Java syntax derived from C++: It will look familiar

Arrays
- How to declare? int scores [5]; (no need to allocate with new like in Java)
- How to initialize? int scores [5] = {2, 5, 10, 5, 7} ;
- How to access elements? scores [0], scores [1],... scores [4].
  Warning ! C++ gives no error if index is out of range (Java does!)

Primitive types
- Integer types: short, int, long
  - Default signed (+ and -)
  - Can also be unsigned (only +)
- Characters: char
- Floating point numbers: float, double, long double
- Boolean type: bool

Compound types
enum color {red, orange, green};
struct PersonInfo{
  double height ;
  int age ;
  Date birthday ;
} ;
**C-Strings**

An ordinary array of characters terminated by ‘0’ (character with value of 0).

Examples:

```c
#include <cstring>

char name[10] = "abcd"; // 10 char array
char name[5] = "abc"; // 5 char array
```

**Warning:** Never destroy C! This is a very common mistake.

```
```

**C++ Strings**

`string` is a C++ class (like in Java), its simpler (and safer) to use than character array.

```
#include <string>

string str;  // string constructor
string str = "abc"; \ \ C++ string from C-string
```

The class `string` has some useful member functions:

- `str[]` - returns last element
- `str.size()` - returns size of string
- `str + str2` - concatenates 2 strings
- `str < str2` - compares two strings
- `str.find(string)` - finds a substring in string  
- `str.size()` - returns the length of string

**Type conversions**

- `atoi()` – `#include <stdlib.h>` int `atoi(const char *str);` converts `str` into an integer, and returns that integer. `str` should start with some sort of number, `atoi()` will stop reading from `str` as soon as a non-numerical character has been read. For example, `i = atoi("512.035");` results in `i` being set to 512.

- You can use (Standard C I/O) `sprintf()` to convert a number into a string.

- `atof()` – converts a C string to a floating point number

**Variables and addresses**

Each variable occupies a block of bytes in the memory. The size depends on its type and on the computer architecture.

For example:

```c
int age = 14;
double temperature = 38.5;
```

```c
cout << "the age value is " << age << " and its address is " << &age << endl;
```

```c
cout << "the temperature value is " << temperature << " and its address is " << &temperature << endl;
```

```
#include <iostream>;

using namespace std;

int main()
{
    int age = 14;
    double temperature = 38.5;
    cout << "the age value is " " and its address is " << &age << endl;
    cout << "the temperature value is " " and its address is " << &temperature << endl;
}
```

Output:

the age value is 14 and its address is 0012FF2C  
the temperature value is 38.5 and its address is 0012FF24
Hex to decimal

Usually the address is represented in hexadecimal notation (usually prefixed with “0x”)

\[ \text{cout} << \& \text{var} \] prints the address of \text{var} in hexadecimal

\[ \text{cout} << (\text{unsigned int}) (\& \text{var}) \] prints the same address in decimal notation

Note: This is valid for 32 bit systems. For 64 bit systems, you should use \( \text{(unsigned long long)} \).

### Pointer declaration

- **pointer** is a specific \( \text{C/C++} \) variable type.
- **A pointer** holds the memory address of a variable. The address is not the value!
- Declaration:
  - \( \text{int} \ \& \text{p}; \) // in \( \text{C/C++} \)
  - \( \text{int} \ \* \text{p}; \) // in \( \text{C++} \)

Means \( \* \text{p} \) is an integer

When the pointer is declared, the variable it points to does not exist yet.

### Pointer assignment

\[ \text{int} \ \* \text{p} = \& \text{number}; \]

\( \* \text{p} = 42; \)

\( \* \text{p} \) is the variable pointed by \( \text{p} \) number becomes 42
Pointers are not integers, but you can say p = p + 1, or p++

p++ increases its value with the nr. of bytes to which it points

```
int prices[3] = {100, 200, 300};
int *p;
p = prices;
```

Always initialize pointer before you apply the dereferencing operator *.

```
#include <iostream>
using namespace std;
int main()
{
    int *p;
    new int(25);
    *p = 25;
    cout << "The value at p: " << *p;
    return 0;
}
```

A dynamic variable is created and destroyed while the program is running (at run time) (think about a holiday)

```
int *p;
p = new int;
creates a new dynamic integer variable and sets p to point to this variable
*p = 25;
A non-dynamic variable is called automatic (binding at compile time)
```

Freestore (heap) is a special memory area reserved for dynamic variables.

Warning: C++ does not have garbage collection! (Java does.)

Take care: Recycle any freestore memory that is no longer needed using the delete operator. Otherwise, memory leaks!

```
delete p; // destroys the dynamic variable pointed by p.
```

After delete p, p becomes an undefined pointer variable: a dangling pointer.

Take care: before using *p again, be sure p points to something and is not a dangling pointer. Otherwise unpredictable effects.

For every time in your program that you call new there should be a call to delete.

Do not delete p twice. This is a common mistake!

After delete p and before trying *p, be sure p points to something
```
delete p; // free memory pointed to by p
p = NULL; // mark p as bogus pointer
```
Arrays are pointers

- `int a[10]` - `a` is an variable of type array with 10 elements
- But `a` is also a pointer to its first element `a[0]`
- If you say `cout << a` you will get the address of its first element.
- `a[5] == (a+5)[0] == *(a+5) != *(a)+5`

Dynamic arrays and pointers

A dynamic array: array is created when the program is running (=dynamic binding)

- create a dynamic array:
  ```
  double* a; // declare a pointer to the array
  a = new double[10]; // create a pointer that points to the first element of a block of 10 integers
  ```
- use it like an ordinary array: `a[0], a[1], etc.`
- delete all the array to return the memory to the freestore
  ```
  delete[] a; // always use [ ] to delete arrays!
  ```

Outline

- About C/C++ language
- Getting started
- Data types (arrays, strings, pointers)
- Functions

Functions

- Top down approach
- A function has:
  - a declaration = name + formal parameters (only header+comments about what it does)
  - a definition = name + formal parameters+body (how it does)
  - one or many calls = name + actual parameters (function at work)
- C++ requires that either the complete definition or at least the function declaration appears in the code before the function is called (see the following 2 examples).

Functions: example 1

using standard namespace;
```
int max ( int param1, int param2 )
{
  function definition
  param1, param2
  = formal parameters
  
  if ( param1 < param2 ) return param2;
  
  else return param1 ;
}
```

Functions: example 2

using standard namespace;
```
int max ( int param1, int param2 );
// function defintion returns the maximum of two integers
int main()
{
  int max_val, a, b;
  // function call
  max_val = max ( a, b );
  return 0 ;
}
```
Functions: plugging in parameters

- When the function is called, the actual parameters of a function are plugged in its formal parameters.
- C++ has 2 plug-in mechanisms:
  - Call by value: `int do Stuff (int a)`; the value of `a` is plugged in (copied)
  - Call by reference: `void do Stuff (int & a)`; the address of `a` is plugged in (copied)

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

void do Stuff(int by_value, int & by_ref)

int main()
{
    int sheila, bob;
    sheila = 1;
    bob = 2;
    do Stuff(sheila, bob);
    cout << "sheila after call" << sheila << endl;
    cout << "bob after call" << bob << endl;
    return 0;
}
```

Example continued...

```
void do Stuff (int by_value, int & by_ref) {
    by_value = 1000;
    cout << "by_value in function call is" << by_value << endl;
    by_ref = 2000;
    cout << "by_ref in function call is" << by_ref << endl;
}
```

Output:
```
by_value in function call is 1000
by_ref in function call is 2000
sheila after function call is 1
bob after function call is 2000
```

by-value vs by-reference example 2.1
```
#include <iostream> // for cout
#include <cmath>   // for sqrt
using namespace std;

// solves the quadratic equation ax^2 + bx + c = 0
// if roots are real the function returns the roots in root1 and root2 and if they are complex function returns false
bool quadsolve (float a, float b, float c, float & root1, float & root2)   //out by reference
{
    float disc;
    disc = b*b - 4*a*c;
    if (disc < 0.0) return false;
    else
    {
        root1 = (-b + sqrt(disc)) / (2 * a);
        root2 = (-b - sqrt(disc)) / (2 * a);
        return true;
    }
}
```

by-value vs by-reference example 2.2
```
int main()
{
    float c1 = 1.0 , c2 = 1.0, c3 = -6.0;
    float r1, r2;
    if (quadsolve (c1, c2, c3, r1, r2))   //here is the call
        cout << "Roots are " << r1 << " and " << r2 << endl;
    else
        cout << "Complex roots" << endl;
    return 1;
}
When to use call-by-reference?

- If you want the arguments to remain changed also after the function call (see example 1)
- If you want your function to return more than one value (see example 2)
- If the argument you want to pass to the function is a large structure (like an array)
  - To prevent the structure from being copied
  - Can use const keyword to prevent changes

Passing arrays as parameters

- An index variable can be formal parameter for a function: add (score[3]);
- Arrays can be formal parameters for functions: array parameters.
  - Array elements remain changed after the function call.
- During the call the function gets the address of the first element but not the array length (just like a pointer).
  - Another parameter to tell the length is always required.
  - Java does not require the length because arrays are objects, not pointers, in Java.

Array parameters

```c
void fill_array (int a[], int size) {
    cout << "Enter " << size << " numbers:\n";
    for (int i = 0 ; i < size ; i++)
        cin >> a[i];
}

function call: fill_array (score,5);
```

Or like this

```c
void fill_array (int *a, int size) {
    cout << "Enter " << size << " numbers:\n";
    for (int i = 0 ; i < size ; i++)
        cin >> *a[i];
}

function call: fill_array (score,5);
```

Outline

- About C/C++ language
- Getting started
- Data types (arrays, strings, pointers)
- Functions
- About this course
- Self test exercises

About this course

- The place of this course in the curriculum:
  - Introductory methods
  - Programming in C/C++
  - Design/Case
  - Advanced

- Why C++?
  - It is fast and widely used in companies
- What can you learn during this course?
  - Learn to work with objects
  - Learn and experiment with modern C++ features, like template libraries, graphical libraries, etc.
About this course

How much time should we spend on this course?
(2 ECTS = 2 x 28 = 56 hours)

Lecture 1. Procedural programming 1 (C/C++ basics) (2 h)
Lecture 2. Procedural programming 2 (I/O and linked lists) (2 h)
Lecture 3. Object oriented programming (2 h)
Lecture 4. Generic programming (2 h)

Total: 8 hours lectures
Assignment 1. Warm up (8 hours)
Assignment 2. AEX (16 hours)
Assignment 3. Company (16 hours)
Assignment 4. Tourism (8 hours)

Total: 48 hours practicals

How do we get our 2 ECTS? It is of course useful to attend the lectures. But really important to get your credit points is to deliver the 4 assignments on time.

Course website: www.few.vu.nl/~nsilvis/C++/2009

Assistance?
- Each student has an assistant assigned (see website).
- You can communicate with your assistant by email.

Books

- S. Prata, C++ Primer, SAMS, 2005
- W. Savitch, Problem solving with C++, Addison Wesley, 2005
- Leen Ammeraal, Basiscursus C++, Academic service, 1999
- Liberty Jones, Teach yourself C++ in 21 days, SAMS, 2005
- T. Budd, C++ for Java Programmers, Addison Wesley, 1999
- S. Lipmann, J. Lajoie, C++ Primer, AT&T, 1998

Other resources

- Practical guide on the C++ website
- Useful links to on-line tutorials, also on the website
- Announcements on the website

Self test exercises

1. What is the output of the following program lines?
   ```
   char a = 'b';
   char b = 'c';
   char c = a;
   cout << a << b << c << 'c';
   ```

2. What is the output of the following program lines?
   ```
   int x = 10;
   while (x > 0)
   {
     cout << x << endl;
     x = x - 3;
   }
   ```

3. What is the output of the following program lines?
   ```
   int x = 42;
   do
   {
     cout << x << endl;
     x = x - 3;
   } while (x > 0);
   ```

4. Write a complete C++ program to output the even numbers from 0 to 20 one per line. The program does nothing else.

5. Write a function to swap two doubles. Use this function in a program to determine and print the maximum element of an array.
Questions?