Cours de C++

Organisation d'un programme

Cécile Braunstein
cecile.braunstein@lip6.fr

Organizing program

Goal

For a large problem, having only one program is unmanageable.
We need to break the program into independent and named parts.

C++ Language

- Functions
- Data structures
- Combine data structure and functions $\Leftrightarrow$ class

Functions

Advantages

When we have a computation on several places :

- Reduce the total programming effort
- Makes it easier to change the computation

Definition

```
T function_name(T param)
{
  //function body
}
```

We have : return type / function name / parameters list / function body.

Parameters

Example

Computing student's grade

```
double grade(double midterm, double final, double
    homework)
{
  return 0.2 * midterm + 0.4 * final + 0.4 * homework;
}
```

Parameters list

Behaves like local variables to the function :

- Calling the function : create the variables
- Returning from the function : destroy the variables

Call by value

```
std::cout << "Your final grade is : " << setprecision(3)
          << grade(midterm,final,sum/count)
          << setprecision(prec) << std::endl;
```

Arguments

- Arguments can be a variable or an expression.
- Each argument is used to initialize the corresponding parameters
- The parameters take a copy of the value of the argument

Reference

Definition

```
vector<double> homework;
vector<double>& hw = homework;
```

`hw` is a reference to the object `homework` $\Leftrightarrow$ hw is a synonym of homework
Anything we do on `hw` is equivalent to do the same thing to `homework` and vice versa.

Constant

```
const vector<double>& chw = homework;
```

`chw` is a other name for `homework` but `const` promises that we will not change the value of `chw`.
`chw` is "read-only" variable.

Call by `const` reference

```
double median(vector<double> vec)

double grade(double midterm, double final,
             const vector<double>& hw)
{
  return grade(midterm,final,median(hw));
}
```

const

- Direct access to the argument
- No copy of the argument
- Promise we will not change the value

Call by reference (*lvalue*)

We want to have a function that returns to value at once.

```
istream& read_hw(istream&
    in, vector<double>& hw
    )
{
  if(in){
    hw.clear();

    double x;
    while(in >> x)
      hw.push_back(x);

    in.clear();
   }

return in;
}
```

Reference

- Not copy
- The function will modify `hw` and `in`

```
if(read_hw(cin,homework))
    {...}
```

lvalue

It's a value of a non-temporary object : a variable, a reference, a function that return a reference

Resume

Call by	const ref	value	ref
	`void f(const string &a)`	`void f(string a)`	`void f(string &a)`
modification of `a`	No	local	with side effect
accepted values	All	All	non-temporary
advantages	security no copy	simple	more general no copy

Structure

Definition

```
struct Student_info{
  string name;
double midterm;
vector<double> homework;
};
```

- Type that contains zero or more members.
- Each object of the structure type contains its own instance of its members.

Exceptions(1)

```
int main () {
  try
  {
    throw 20;
  }
  catch (int e)
  {
    std::cout <<
    "Exception Nr. "
    << e << std::endl;
  }
  return 0;
}
```

Behavior

- The executions stops in the part in which the `throw` appears.
- An exception object is thrown.
- Exception handler : `catch` place just after a `try`

Exceptions (2)

```
try {
  // code here
}
catch (int param) { cout
    << "int exception"; }
catch (char param) { cout
    << "char exception"; }
catch (...) { cout << "
    default exception"; }
```

catch

- We can have multiple `catch`
- Catch the parameters type
- `...` works as default case