Object-Oriented Programming: Exam Survival Kit

June 11, 2010

1 Input and Output

The preferred way of printing in C^{++} is through the <<-operator.

```
std::cout << 500 << "\n";
   // Prints the number 500 followed by a newline.
std::cout << ( a + b + c ) << "\n";
   // Prints the sum of a,b, and c, followed by a newline.
std::cout << "abcdefghijklmnopqrstuvwxyz\n";
   // Prints a character string.
```

It is important to remember that a `std::vector` cannot be printed through <<. One has to print the elements one by one, or define a proper <<-operator.
The preferred way of reading input is the >>-operator:

```
std::cin >> i;
   // Reads the variable i.
```

The >>-operator knows about the type of i, and it will make sure that the input is converted into the type of i in the proper way.
In addition, there exist the functions `putchar( )` and `getchar( )` that write and read a single character. They are inherited from *C*, and you should use them only when you want to do your own conversions.
The following program prints all the characters that it reads to its output:

```
int main( int argc, char * argv [ ] )
{

   int c = getchar( );
      // Reads the character c from std::cin. If there is no
      // character left, the result will be EOF.
   while( c != EOF )
   {
      putchar(c);
         // Writes the character into std::cout.

      c = getchar( );
         // Get new character from std::cin.
   }
   return 0;
}
```

2 Function Definitions

You need to understand how function definitions work in C^{++}. A function definition has the following form:

```
T  name( A1 a1, ..., An an )
{
   ...
}
```

T is the *return type* of the function. In case you want the function to return nothing, it is possible to have *T* is `void`.

A function call has form `name( t1, ..., tn),` where each `ti` must have type `Ai`.

If the function returns a type that is not `void`, you can do something with the result, e.g. store it in a variable, print it, or use it in further computation.

If you want the function to return an integer, you can write `integer func ( ... ) .` If you want the function to return a vector of integers, you can write `vector< int > myfunc ( ... )`

The sequence `A1 a1, ..., An an` specifies the *parameters* of the function. `A1` is the type of the first parameter, `a1` is the name of the first parameter.

`An` is the type of the last parameter, `an` is the name of the last parameter. A function with two parameters, both int, and returning an int, could be defined as follows:

```
int f( int i1, int i2 )
{
}
```

A function returning nothing, with one parameter, which is a vector of int, can be defined as follows:

```
void g( std::vector< int > x )
```

A function that returns a vector of unsigned int, and which requires are vector of unsigned int can be defined as follows:

```
std::vector< unsigned int > ff( std::vector< unsigned int > v )
```

In the body of the function, (that is the part between { and }), you should try to compute the result. The `return` statement can be used for returning the result, when it is computed. The following function computes $n!$:

```
unsigned int fact( unsigned int n )
{
   unsigned int f = 1;
   for( unsigned int i = 1; i < n; ++ i )
      f = f * i;
   return f;
}
```

3 Vectors

A vector is a finite sequence of objects of the same type. A vector of elements of type X is declared as follows:

```
std::vector< int > v1;            // Vector of integers.
std::vector< double > v2;         // Vector of doubles.
std::vector< unsigned int > v3;   // Vector of unsigned integers.
std::vector< std::vector< int > > v3;
   // Vector of vectors of integers.
```

At the moment of its creation, a vector has length 0. The length of a vector is obtained by the `size` method:

```
v1. size( )
   // Returns the size of v1.
```

The `push_back( )` method appends an object to the end of a vector:

```
v. push_back(i);
   // Appends i to the end of the vector. If i has type
   // std::vector< X >, then v must have type X.
```

The `pop_back( )` method removes an object from the end of a vector:

```
v. pop_back( );
```

Removing the last element from an empty vector is not a good idea.
Elements of a vector can be accessed through the `[ ]`-method.

```
std::cout << v[i] << "\n";
v[i] = 3;
   // It must be the case that i < v. size( );
   // i can have type int or unsigned int. Double is not possible.
   // If v has type std::vector<X>, then v[i] has type X.
```

As can be seen from the example, the `[ ]`-method can be used both for reading and for writing.

Note that vectors cannot be printed, unless you write a printing function by yourself.

4 Declarations

Variables must be declared before use. A declaration has form

```
T t;
```

Here T is the type of the variable and t is the name of the variable. Here are a few examples:

```
unsigned int i;
unsigned int j;
double x;
std::vector< unsigned int > i;
```