

## Laboration 2

### Menu system

#### Introduction

The purpose of this programming exercise is to show how virtual methods work in C++ and to show how objects can interact through abstract interfaces.

#### Preparations

Put your files in a separate directory and make sure that this directory can only be read by the members of your lab group. If you want the teaching assistants to be able to look at your files, give the AFS group `oopk_e00:assistent` read access for the directory.

Always load the course module before starting work:

```
module load oopk_e00
```

Copy the make file from the course directory

```
cp $OOPKHOME/Makefile .
```

(you need to edit this file before compiling).

#### The Problem

You are to write a few classes that make it easy to create menus in menu oriented programs. A menu may look like the following on screen:

```
--- Main menu ---
1. Read input data.
2. Write output data.
3. Process data.
4. Quit program.
Select option:
```

First, the menu's name is shown (here, it is "Main menu"), followed by a few numbered options. When the user enters a number after at the "Select option:" prompt and presses RETURN, the corresponding operation should take place. This is repeated until the option "Quit program" is selected.

Such a menu should be represented by an instance of the class `Menu` in your C++ program. The alternatives in the menu should be represented by subclasses of an abstract base class `MenuItem`. The `MenuItem` objects should contain the text to be written on their line of the menu (e.g., "Read input data") and perform the right operation when the user selects the corresponding number from the menu. For each operation, it is then possible to define a new sub-class of `MenuItem`, instantiate an object of this class, and associate the object with the menu. Of course, there should be no need to alter the code for `Menu`, which should be written once for all menus. The menu object should automatically number the options. The classes `Menu` and `MenuItem` could be used as follows to create the menu above:

```
int main() {
    Menu menu("Main menu");
    ReadItem read_item(menu, "Read input data.");
    WriteItem write_item(menu, "Write output data.");
    ProcessItem process_item(menu, "Process data.");
    QuitItem quit_item(menu, "Quit program.");

    menu.prompt();
    return 0;
}
```

(For a window oriented user interface, this code could produce a pop-up menu using another implementation of the classes `Menu` and `MenuItem`. In this exercise, we use the standard input and output instead for simplicity.)

`ReadItem`, `WriteItem`, `ProcessItem` och `QuitItem` are all sub-classes of `MenuItem` that perform dif-

ferent actions when activated from the menu (let them just print some text to the standard output to simulate this.)

`Menu::prompt()` should print out the menu and wait for user input, and then tell the selected `MenuItem` to perform the operation by calling a method (say, `MenuItem::invoke()`). It should then print the menu again.

Instances of the sub-class `QuitItem` need to have a way of telling menu that it is time to return from `Menu::prompt`. They can, for example, do this by returning a special value to menu, or call a method, say `Menu::quit`, in menu. Exactly how this is done will be hidden from the application programmer, so you can decide how to do this yourself.

If there are many menu options, there is a need for sub-menus. For example, one might want to move the alternatives "Read input data" och "Write output data" to a sub-menu called "File handling". Define such a sub-class, called `SubmenuItem`, that connects one `Menu` to another:

```
int main() {
    Menu submeny("File handling menu");
    ReadItem read_item(submeny, "Read input data.");
    WriteItem write_item(submeny, "Write output data.");
    QuitItem return_item(submeny, "Return to main menu.");

    Menu menu("Main menu");
    SubmenuItem submeny_item(menu, "File handling...", submeny);
    ProcessItem process_item(menu, "Process data.");
    QuitItem quit_item(menu, "Quit program.");

    menu.prompt();
    return 0;
}
```

På skärmen ska det då se ut på följande sätt. Notera att `QuitItem` får undermenyn att återvända till närmast högra menynivå. (alv> är promptern)

```
alv> ./a.out
--- Main menu ---
1. File handling...
2. Process data.
3. Quit program.
Select options: 1
--- File handling menu ---
1. Read input data.
2. Write output data.
3. Return to main menu.
Select option: 3
--- Huvudmeny ---
1. File handling...
2. Process data.
3. Quit program.
Select option: 3
alv>
```

## Hints

In the course directory, there are two functions that convert between strings and integers:

```
bool string2int(const string& str, int& i);
string int2string(int i);
```

These are declared in the file `<oopk/convert.hh>` that you need to include first. `string2int` returns `true` if `str` can be interpreted as an integer. In this case the result is places in the parameter `i`. Otherwise, `false` is returned, and the parameter `i` is not modified. `int2string` can not fail, and always returns the string resulting from the conversion.