1. Маніпулятори. Побудова та використання маніпуляторів з параметрами.

Создание функций-манипуляторов, имеющих аргументы, несколько сложнее, чем манипуляторов без аргументов. Одна из причин этого заключается в том, что в Borland С++ параметризованные манипуляторы используют классы-шаблоны. Эти классы создаются с использованием ключевого слова template. Если не представлять, как работа­ют классы-шаблоны, то нельзя полностью понять создание параметризованных манипуляторов.

Для создания параметризованного манипулятора необходимо в файл включить заголовочный файл iomanip.h. В этом файле определено несколько шаблонов. Мы будем использовать два из них — omanip и imanip. Первый используется для создания манипуляторов вывода с аргументами. Второй используется для создания параметризованных манипуляторов ввода. (Можно взглянуть на опре­деления этих классов в файле iomanip.h для того, чтобы посмотреть, как они реализованы.)

В общем случае, если необходимо создать манипулятор, имеющий аргументы, то следует со­здать две перегруженных функции-манипулятора. В одной необходимо определить два парамет­ра. В качестве первого параметра используется ссылка на поток, а вторым является параметр, передаваемый в функцию. Вторая версия манипулятора имеет один параметр — тот, который указывается при использовании манипулятора в выражениях ввода/вывода. Эта вторая версия генерирует вызов первой версии. Общая форма манипуляторов вывода с параметрами имеет вид:  
  
*ostream &имя\_манипулятора(ostream &поток, тип параметр)  
// код  
return поток;  
}  
// перегрузка  
omanip <тип> имя\_манипулятора(тип параметр) {  
return omanip <тип> (имя\_манипулятора, параметр);*

Здесь имя\_манипулятора является именем манипулятора, а тип указывает на тип параметра, ис­пользованного манипулятором. Поскольку omanip является классом-шаблоном, то тип также ста­новится типом данных, с которыми оперирует специфический объект omanip, возвращаемый ма­нипулятором.

Следующая программа создает параметризованный манипулятор вывода indent(), осуществля­ющий отступ на указанное число пробелов.

#include <iostream.h>  
#include <iomanip.h>  
// отступ на указанное число пробелов  
ostream &indent(ostream &stream, int length)  
{  
register int i;  
for (i=0; i<length; i++) cout << " ";  
return stream;  
}  
omanip<int> indent(int length)  
{  
return omanip<int>(indent, length);  
}  
int main()  
{  
cout << indent (10) << "This is a test\n";  
cout << indent (20) << "of the indent manipulator. \n";  
cout << indent (5) << "It works! \n";  
return 0;  
}

Как можно видеть, функция indent() перегружена в соответствии с тем, как это обсуждалось выше. Когда программа доходит до indent(10), выполняется вторая версия indent() со значением 10, переданным в параметре length. Эта версия затем исполняет первую версию с величиной 10, пере­данной снова в параметре length. Этот процесс повторяется всякий раз при вызове indent().

Тип параметра манипулятора находится под контролем программиста и определен типом, который задан для второго параметра функции-манипулятора. Тот же самый тип затем исполь­зуется при создании шаблона для перегруженной версии манипулятора. Например, следующая программа показывает, как передать значение типа double функции-манипулятору. Оно затем выводится с использованием формата доллары-центы.  
  
#include <iostream.h>  
#include <iomanip.h>  
ostream &dollars(ostream &stream, double amount)  
{  
stream.setf (ios::showpoint);  
stream << "$" << setw(10) << setprecision (2) << amount;  
return stream;  
}  
omanip <double> dollars(double amount) {  
return omanip<double> (dollars, amount);  
}  
int main()  
{  
cout << dollars (123.123456);  
cout << "\n" << dollars(10.0);  
cout << "\n" << dollars(1234.23);  
cout << "\n" << dollars(0.0);  
return 0;  
}

Манипуляторы для ввода данных также могут иметь параметр. В следующей программе созда­ется манипулятор getpass(), используемый для получения пароля. Он имеет аргумент, в котором указано число попыток, предоставляемых пользователю для корректного ввода пароля.  
  
// программа использует манипулятор для ввода пароля  
#include <iostream.h>  
#include <iomanip.h>  
#include <string.h>  
#include <stdlib.h>  
char \*password="IlikeC++";  
char pw[80];  
// ввод пароля  
istream &getpass(istream &stream, int tries)  
{  
do {  
cout << "Enter password: ";  
stream >> pw;  
if (!strcmp(password, pw) ) return stream;  
cout << "\a"; // bell  
tries--;  
} while (tries>0);  
cout << "All tries failed! \n";  
exit(1); // пароль не введен  
return stream;  
}  
imanip<int> getpass(int tries) {  
return imanip<int>(getpass, tries);  
}  
int main()  
{  
// три попытки на ввод пароля  
cin >> getpass (3);  
cout << "Login Complete! \n";  
return 0;  
}

Обратим внимание, что формат для манипуляторов ввода тот же самый, что и для манипуляторов вывода, за исключением двух моментов: используется поток ввода istream и указывается класс imanip.

Подобные манипуляторы, приспособленные к конкретному использованию, могут значительно упростить инструкции ввода/вывода и улучшить читаемость кода.