char \* strcpy ( char \* destination, const char \* source );

For empty strings

char \* strcat ( char \* destination, const char \* source );

For concatenate strings

int puts ( const char \* str );

Show a string in the standard output

char \* strncpy ( char \* destination, const char \* source, size\_t num );

Copy an specific number of characters from source to destination

char \* strstr ( char \* str1, const char \* str2 );

Return a pointer to the first occurence of str2 in str1

int strcmp ( const char \* str1, const char \* str2 );

Compare two strings, if are equal = 0, if are lower str1 = -1, if are lower str2 = 1  
 stricmp compare without sensivity of case

+=, %=, -=, \*= Asignación a la misma variable

& La dirección de

\* (expresión) La cosa apuntada por  
 (declaración) Apuntar a

pIntVar es del tipo \*int, \*pIntVar es del tipo int

new/delete son funciones usadas para Heap, variables creadas en funciones que serán utilizadas en otros ámbitos. Después del delete es recomendable volver el puntero a cero

Depurador:

* % g++ arch.cpp -o arch
* % arch
* % g++ -g arch.cpp -o arch (archivo para depurador)
* % gdb arch (depurador)
* (gdb) list x,y (lines of code from x to y)
* (gdb) run
* (gdb) break x (stop on x line)
* (gdb) next
* (gdb) stop
* (gdb) quit

Class – private as default

Structure – public as default

Friend (add the prototype of the function)– access from extern functions to protected members from the class

Constructor is a function with the same name as the class, they function is initialize in 0 the members. They don’t have void, because of this, they only function for initialize

:: son referencias a clases en específico

: lo que sigue son llamadas al constructor de los miembros de datos de la clase actual. Las variables const o referencia en una función deben ser creadas con :

Static members requiere an specific sintaxis to work, an inicialization in fact  
 type class :: static-member = 0; (general inicialization)  
 The static members need to be global.

Herencia tiene como sintaxis   
 class NameOfClass…  
 class NameOfSubclass : public NameOfClass …

Polimorfismo: necesita la palabra *virtual* antes del prototipo de función miembro polimórfica, además, la función polimórfica debe incluir como argumento la clase más general. Técnicamente sólo es necesario declarar la función miembro polimórfica en la clase base, la “virtualidad” es llevada a las subclases automáticamente. Los argumentos de las funciones deben encajar para poder usar el polimorfismo, en caso de no ser así, convertirá los argumentos para que encajen con la primera función miembro declarada polimorfa.

* En caso de especificar referencia, aunque sea polimorfa la función, se usará la que fue especificada.
* Las funciones virtuales no pueden ser alineadas, deben ser declaradas en la clase y expandidas posteriormente.
* Funciones estáticas y constructores no pueden ser virtuales. Sin embargo, los destructores sí deben ser declarados estáticos en el caso de las herencias (salvo el caso de que no tenga funciones miembro virtuales, no tenga muchos miembros de datos y se pretendan crear muchos objetos de esta clase).
* Una función virtual pura se declara para subclases, tiene la siguiente sintaxis  
  virtual /Tipo de dato a regresar\ Nombre de la función (Argumentos) = 0;  
  Esto indica que el programador no intenta definir esta función. Esto corresponde a clases abstractas cuyo único fin es dar lugar a subclases concretas; como resultado de ello, no podemos crear objetos de clases abstractas.  
  Una clase deja de ser abstracta cuando sobrepasa todas funciones puras para definirlas de manera concreta.

We can create an operations for our classes, we only need to define it with the next sintaxis:   
 In the class: friend class operator/sign\(arguments)  
 Out of the class: /argument to return\ operator/sign\(arguments){ complete definition }  
 We have an special exception with the operator ++:  
 For the pre form ++x, we have operator++(Class name)  
 For the post form x++, we have operator++(Class name, int), with a zero as second.

In the same way, we can use operators of form, this is, convert an specific class in other type, (type)class.  
 operator type(){ definition }

Or we can convert different types into our class  
 declaration in our class: class(type to convert)  
 definition out of the class: class::class(type to convert){ definition }

If we create an specific convertitor for our classes to the basic types in C++, we don’t need to create an operators for our classes, because C++ will be converting our class into the basic forms and operate with them, for convert at the end in the oposite way.

The assignment operator (=) need to be a member function instead of a non-member function, obviously, it doesn’t be an static function. But, the assignment operator (+=, -=) doesn’t have this problem.

To make a specific output of the data, we have libraries to manipulate the output of the data. An example of this is the function precision.  
 cout.precision(n) in iostream to out a specific number of decimals in a float  
 cout << setprecision(n) in iostream and iomanip to ajust the decimals

Manipulators to control data flux output

* Dec / flags(10): root in 10
* Hex / flags(16): root in 16
* Oct / flags(8): root in 8
* Setfill(c) / fill(c): fill the spaces with the character c
* Setprecision(c) / precision(c): precision of output to c
* Setw(n) / width(n): long of the field to n characters

With the manipulation functions in iostream, we can ask for the current state of the manipulators, like: int value = cout.precision(); and this return us the current precision value.

Try, catch, throw: use this for manipulate the mistakes on the code, we can use to evade think on all the possible mistakes on the code.

In line with this, we can create our specific exceptions as classes

Class Exception{

Public:

Exception( char\* pMsg, char\* pFile, int nLine){

Strncpy(msg, pMsg, sizeof msg);  
 msg[sizeof msg - 1] = ‘\0’;  
 strncpy(file, pFile, sizeof file);  
 file[sizeof file – 1] = ‘\0’;  
 lineNum = nLine;

}

Virtual void display(ostream& out){

Out << “Error <” << msg << “>\n” << “occurred onn line #” << lineNum   
 << “, file” << file << endl;

}

Protected:

Char msg[80];  
 char file[80];  
 int lineNum;

}

The throw shows like:  
 throw Exception(“Negative argument to factorial”, \_FILE-\_, \_LINE\_);

FILE y LINE son #define intrínsecos que son ajustados con el nombre del archive Fuente y el número de línea actual en ese archivo, respectivamente.

If we want, we can use subclass on exception to make an specific exceptions on our throws.

For multiple herency, we can use the sintaxis:  
 class /name of subclass\(arguments) : public /class to heredate\, public /another class to heredate\ { description }

But, in the pattern class, we need to introduce an specific arrengments to solve collision problems, we need an root patern unique to the collision problems, and a virtual name in the first heredate on the pattern class. This is necessary to use an unique type of everything on our code.