Auxiliar 1: Geometría computacional

Introducción a objetos en C++

Profesora: Nancy Hitschfeld-Kahler
Auxiliar: Sergio Salinas

CC5502-1 – Geometría Computacional

March 27, 2023

ToC

Introducción a C++

Introducción a C++

- Desarrollado por Bjarne Stroustrup en 1983 como una extensión del lenguaje C.
- Lenguaje de bajo nivel con control de memoria necesarias para aprovechar la arquitectura de la GPU.
- Se recomienda usar el estándar de C++ moderno (C++ 11 en adelante).

Hello world

hello.cpp

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
}

//g++ hello.cpp -o hello
```

Tipos de variables en C++

```cpp
int entero = 10;
float flotante = 3.14;
double doble = 3.141592653589793;
char caracter = 'A';
bool booleano = true;
long largo = 1234567890;
long long muy_largo = 12345678901234567890;
unsigned int sin_signo = 42;
unsigned long largo_sin_signo = 9876543210;
```

Funciones en C++

```cpp
#include <iostream>
// Función que suma dos enteros y devuelve el resultado
int suma(int a, int b) {
    return a + b;
}
// Función que imprime el resultado de la suma
void imprimirResultado(int resultado) {
    std::cout << "El resultado de la suma es: " << resultado <<
↪   std::endl;
}

int main() {
    int num1 = 5;
    int num2 = 7;
    int resultado = suma(num1, num2);
    imprimirResultado(resultado);
    return 0;
}
```

Clases en C++

Introducción a Clase en C++

- En C++, un objeto es una instancia de una clase.
- Una clase es una estructura de datos que define un conjunto de atributos y métodos que operan sobre esos atributos.
- Los objetos son útiles para modelar entidades del mundo real y representar datos complejos.
- Los objetos se pueden crear dinámicamente en tiempo de ejecución usando punteros y el operador `new`.
- Los objetos se eliminan usando el operador `delete` para liberar la memoria asignada a ellos.

Declaración

rectangulo_ex1.cpp

```
class Rectangulo {
private:
    // atributos privados
    double ancho, alto;
public:
    // Constructor por defecto
    Rectangulo() {
        ancho = 0.0; alto = 0.0; }

    // Constructor con parámetros
    Rectangulo(double ancho, double alto) {
        this->ancho = ancho; this->alto = alto; }

    // método público para calcular el área
    double calcular_area(){
        return ancho * alto;}
};
```

Crear un objeto

rectangulo_ex1.cpp

```cpp
int main()
{
    //Crea un objeto en el stack
    Rectangulo fig1 = Rectangulo(1,2);
    fig1.calcular_area();

    //Crea un objecto en el heap
    Rectangulo *fig2 = new Rectangulo(4, 5);
    fig2->calcular_area();
    delete fig2;

    return 0;
}
```

Declaración y Definición de la Clase Rectangulo

rectangulo.hpp

```
// Declaración de la clase
class Rectangulo {
private:
  double ancho;
  double alto;

public:
  Rectangulo();
  Rectangulo(double an, double
↪  al);
  double calcular_area();
};
```

rectangulo.cpp

```
#include "rectangulo.hpp"

// Definición de la clase
Rectangulo::Rectangulo() {
  ancho = 0;
  alto = 0;
}

Rectangulo::Rectangulo(double an,
↪  double al) {
  ancho = an;
  alto = al;
}

double Rectangulo::calcular_area()
↪  {
  return ancho * alto; }
```

Constructores

Tipos de constructores y destructores

rectangulo_constructores.cpp

```
// Constructor por defecto
Rectangulo() {
    ancho = 0;  alto = 0;}

// Constructor con parámetros
Rectangulo(double ancho, double alto) {
    this->ancho = ancho;
    this->alto = alto; }

// Constructor copia
Rectangulo(const Rectangulo& r) {
    this->ancho = r.ancho;
    this->alto = r.alto; }

// Destructor
~Rectangulo() {
    std::cout << "Se ha destruido un rectangulo." << std::endl;
}
```

Tipos de constructores y destructores

```
class Polylla
{
private:

    Triangulation *mesh_input; // Halfedge triangulation
    Triangulation *mesh_output;

public:

    Polylla() {}; //Default constructor

    //Constructor with triangulation
    Polylla(Triangulation *input_mesh){
        this->mesh_input = input_mesh;
        construct_Polylla();
    }

    //Constructor from a OFF file
    Polylla(std::string off_file){

        this->mesh_input = new Triangulation(off_file);
        mesh_output = new Triangulation(*mesh_input);
        construct_Polylla();
    }

    //Constructor from a node_file, ele_file and neigh_file
    Polylla(std::string node_file, std::string ele_file, std::string neigh_file){
        this->mesh_input = new Triangulation(node_file, ele_file, neigh_file);
        //call copy constructor
        mesh_output = new Triangulation(*mesh_input);
        construct_Polylla();
    }


    //Constructor random data construictor
    Polylla(int size){
        this->mesh_input = new Triangulation(size);
        mesh_output = new Triangulation(*mesh_input);
        construct_Polylla();
    }

    ~Polylla() {
        //triangles.clear();
        max_edges.clear();
        frontier_edges.clear();
        seed_edges.clear();
        seed_bet_mark.clear();
        triangle_list.clear();
        delete mesh_input;
        delete mesh_output;
    }
```

Overloading

Overloading de operadores en C++

El **overloading de operadores** es una funcionalidad en C++ que permite a los desarrolladores definir una función que se comporta como un operador. Por ejemplo, para sumar dos números complejos, podemos sobrecargar el operador + de la siguiente manera:

```
class Complejo {
public:
    Complejo operator+(const Complejo& c) const {
        return Complejo(real + c.real, imag + c.imag);
    }
private:
    double real;
    double imag;
};
```

En este caso, estamos sobrecargando el operador + para que sume dos números complejos. El operador se define dentro de la clase `Complejo` y se utiliza el operador + para definir la función.

Ejemplo de Overloading

Complejo.h

```
class Complejo {
private:
    double real, imag;
public:
    Complejo(double r = 0, double i = 0) :
↪   real(r), imag(i) {}
    Complejo operator+(Complejo const &obj)
↪   {
        Complejo res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        return res;
    }
    void imprimir() {
        std::cout << real << " + " << imag
↪   << "i" << std::endl;
    }
};
```

main.cpp

```
int main() {
    Complejo a(3.0, 4.0);
    Complejo b(2.0, 3.0);
    Complejo c = a + b;
    c.imprimir();
    return 0;
}
```

Ejemplo de friend class en C++

```
class Complejo {
private:
    ...
public:
    friend std::ostream&
↪   operator<<(std::ostream& out,
↪   const Complejo& c);
    friend std::istream&
↪   operator>>(std::istream& in,
↪   Complejo& c);
};

std::ostream&
↪   operator<<(std::ostream& out,
↪   const Complejo& c) {
    out << c.real << " + " <<
↪   c.imag << "i";
    return out;
}
```

```
std::istream&
↪   operator>>(std::istream& in,
↪   Complejo& c) {
    cin >> c.real;
    cin >> c.imag;
    return in;
}

int main() {
    Complejo c1(2, 3), c2;
    std::cout << "c1 = " << c1 <<
↪   std::endl;
    std::cout << "Ingrese un
↪   número complejo: ";
    std::cin >> c2;
    std::cout << "c2 = " << c2 <<
↪   std::endl;
    return 0;
}
```

Templates

Templates en C++

Los templates son una característica de C++ que permiten escribir funciones y clases genéricas que pueden trabajar con diferentes tipos de datos sin tener que escribir múltiples versiones para cada tipo de dato.

maximo.hpp

```
template <typename T>
T maximo(T a, T b) {
return a > b ? a : b;
}

int a = 5, b = 10;
maximo(a, b);

double c = 3.14, d = 2.71;
maximo(c, d);
std::string s1 = "hola", s2 = "mundo";
maximo(s1, s2)
}
```

Templates

Declaración

```
template <typename T>
class Rectangulo {
private:
    T ancho;
    T alto;
public:
Rectangulo(T ancho, T alto)
{
    this->ancho = ancho;
    this->alto = alto;
}

T calcular_area() {
    return ancho * alto;
}
};
```

Llamada

```
int main(){
    Rectangulo<int> rect1(4, 5);
    rect1.calcular_area();
    Rectangulo<float> rect2(2.5, 3.5);
    rect2.calcular_area();
    return 0;
}
```

Biblioteca estándar de C++

Biblioteca estándar de C++

Conjunto de funciones, objetos y clases que proporcionan una amplia variedad de características y funcionalidades para el lenguaje C++. Por ejemplo:

- Entrada/salida: operaciones de entrada y salida, como leer o escribir en archivos o en la consola.
- Contenedores: estructuras de datos para almacenar y manipular colecciones de objetos, como vectores, listas, mapas, etc.
- Algoritmos: funciones para realizar operaciones comunes en contenedores, como ordenar, buscar, mezclar, etc.
- Tipos de datos: tipos de datos comunes, como cadenas de caracteres, booleanos, números, etc.
- Funciones matemáticas: funciones matemáticas comunes, como seno, coseno, exponencial, etc.

La Biblioteca estándar de C++ está disponible en cualquier compilador que cumpla con el estándar de C++. Para usarla, se debe incluir el archivo de cabecera correspondiente.

Bibliotecas de la librería estandar

- `<iostream>`: para entrada/salida de consola
- `<vector>`: para el uso de vectores dinámicos
- `<string>`: para el uso de cadenas de texto
- `<algorithm>`: para el uso de algoritmos de ordenación, búsqueda, etc.
- `<unordered_map>` y `<map>`: para el uso de mapas y diccionarios
- `<set>`: para el uso de conjuntos
- `<cmath>`: para el uso de funciones matemáticas como `sqrt()`, `cos()`, `sin()`, etc.
- `<chrono>`: para el uso de medidas de tiempo
- `<thread>`: para el uso de hilos de ejecución
- etc

Más en `https://en.cppreference.com/w/cpp/standard_library`

Ejemplo de string y métodos

- size()
- length()
- empty()
- clear()
- substr()
- find()
- replace()

string_example.cpp

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
  string str = "Hello, world!";
  cout<<"\n str = "<<str;
  cout<<"\n size = "<<str.size();
  cout<<"\n substring = "<<str.substr(0, 5);
  int pos = str.find("mundo"); //pos = 5
  if(str.empty())
  cout << "La cadena esta vacia";
  else cout<<"La cadena tiene"<<str.length()
  <<" caracteres";

  return 0;
}
```

Métodos de vector en C++

- size()
- push_back()
- pop_back()
- insert()
- erase()
- clear()
- reserve()
- empty()

vector_example.cpp

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v;
    v.push_back(10);
    v.push_back(20); //{10, 20}
    v.pop_back(); //20
    v.insert(v.begin() + 1, 30); //{10, 30, 20}
    v.erase(v.begin() + 1); //{10, 20}
    v.clear();
    v.reserve(100);
    if (v.empty())
        cout << "El vector está vacío" << endl;
    return 0;
}
```

Iteradores en un vector de C++

Un iterador es un objeto que se utiliza para recorrer una secuencia de elementos en un contenedor, como un vector.

- `begin()`: devuelve un iterador al primer elemento del vector.
- `end()`: devuelve un iterador al último elemento del vector.
- `rbegin()`: devuelve un iterador al último elemento del vector.
- `rend()`: devuelve un iterador al primer elemento del vector.

Ejemplo

```cpp
#include <iostream>
#include <vector>
int main() {
    std::vector<int> vec = {1,
↪   2, 3, 4, 5};
    // Recorrer el vector con
↪   un iterador
    for (auto it = vec.begin();
↪   it != vec.end(); ++it) {
        std::cout << *it << "
↪   ";
    }
    return 0;
}
```

Vectores para almacenar objetos

rectangulo_constructores.cpp

```
int main() {
    std::vector<Rectangulo> rectangulos;
    rectangulos.push_back(Rectangulo(2, 3));
    rectangulos.push_back(Rectangulo(4, 5));
    rectangulos.push_back(Rectangulo(6, 7));

    rectangulos.at(2);

    for (const Rectangulo& rect : rectangulos) {
        std::cout<<"Area del rectangulo:
↪   "<<rect.calcular_area()<<std::endl;
    }

    std::cout<< rectangulos.at(2).calcular_area() << std::endl;

    return 0;
}
```

La biblioteca algorithm de C++

- sort()
- find()
- replace()
- fill()
- max_element()
- min_element()
- reverse()
- unique()
- binary_search()
- entre otros...

Ejemplo.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <vector>

int main() {
    std::vector<int> v = { 3, 2, 5, 4, 1 };
    std::sort(v.begin(), v.end());
    std::cout << "Vector ordenado: ";
    for (const auto& elem : v) {
        std::cout << elem << " ";
    }
    std::cout << std::endl;
    int max = *std::max_element(v.begin(),
↪   v.end());
    std::cout << "Maximo valor del vector: " <<
↪   max << std::endl;
    return 0;
}
```

Lectura de archivos con stream en C++

datos.txt

```
123
456
789
```

```cpp
#include <iostream>
#include <fstream>

int main() {
std::ifstream archivo("datos.txt");
int num;

while (archivo >> num) {
    std::cout << num << std::endl;
}

archivo.close();
return 0;
}
```

Bibliotecas no estándar

Boost C++ Library

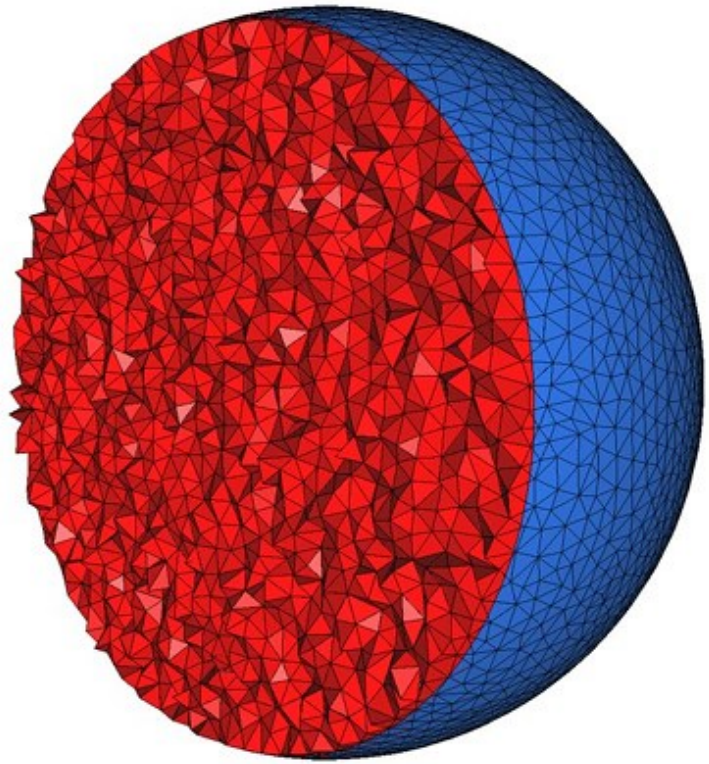
- La biblioteca Boost es una colección de bibliotecas de software libre que extiende las capacidades de C++.
- Boost proporciona muchas utilidades que no se encuentran en la Biblioteca Estándar de C++, cómo algoritmos geométricos

```cpp
#include <iostream>
#include <boost/geometry.hpp>
#include <boost/geometry/geometries/point_xy.hpp>
namespace bg = boost::geometry;

int main()
{
    typedef bg::model::d2::point_xy<double> point;
    point p1(1.0, 1.0), p2(4.0, 5.0);
    double distance = bg::distance(p1, p2);
    std::cout << "Distance between p1 and p2 is: " << distance <<
↪   std::endl;
    return 0;
}
```

La Biblioteca CGAL

- Proporciona algoritmos y estructuras de datos para problemas comunes en geometría computacional.
- Algunas características importantes incluyen:
  - Representación de mallas de polígonos 2D y 3D.
  - Algoritmos de triangulación, convex hull, intersecciones, etc.
  - Soporte para geometría algebraica, curvas paramétricas y superficies.
  - Herramientas para procesamiento y visualización de mallas.



Tests

Testing con Asserts en C++

En C++, se pueden hacer pruebas unitarias utilizando asserts.

- Un assert es una macro que verifica una expresión y termina el programa si la expresión es falsa.
- Los asserts son útiles para detectar errores lógicos en el programa, como valores inválidos de parámetros o errores de cálculo.

Para utilizar los asserts, se debe incluir la biblioteca `cassert` y luego utilizar la macro `assert` con la expresión a evaluar:

```
#include <cassert>

int dividir(int a, int b) {
    assert(b != 0);
    return a / b;
}
```

Si la expresión `b != 0` es falsa, el programa terminará en ese punto y mostrará un mensaje de error.

Compilación

Compilación

makefile

```
all: maximo complejo rectangulo assert_ex iteradores rectangulo_ex1
maximo:
	g++ maximo.cpp -o maximo
complejo:
	g++ complejo.cpp -o complejo
rectangulo:
	g++ rectangulo.cpp rectangulo.hpp -o rectangulo
rectangulo_ex1:
	g++ rectangulo_ex1.cpp -o rectangulo_ex1
assert_ex:
	g++ assert_ex.cpp -o assert_ex
iteradores:
	g++ iteradores.cpp -o iteradores
clean:
	rm -f maximo complejo rectangulo assert_ex1
```

CMake

CMake y Gtest serán explicado en la próxima auxiliar

```
cmake_minimum_required(VERSION 3.5)

project(Auxiliar)

add_executable(maximo maximo.cpp)
add_executable(complejo complejo.cpp)
add_executable(rectangulo rectangulo.cpp rectangulo.hpp)
add_executable(rectangulo_ex1 rectangulo_ex1.cpp)
add_executable(assert_ex assert_ex.cpp)
add_executable(iteradores iteradores.cpp)
```

Bibliografía

Bibliography

Stroustrup, B. (2018). A Tour of C++, Second Edition.