Iconsolata

Introducción al c++ moderno

Gustavo Puche

11 de septiembre de 2020

Outline

Introducción

- Objetivos
- Motivación
- Bases del lenguaje
- Definición de tipos
- Modularidad
- Gestión de memoria
- Clases
- Operaciones esenciales
- Librería estándar
- Uso de lambdas
- Programación genérica (templates)
- Conclusiones

Objetivos

Dar las bases para la programación moderna en C++11 y superiores.

Motivación

- Gran demanda de proyectos en esta tecnología
- Lenguaje de alto rendimiento
- Lenguaje moderno y actualizado
- Programación funcional
- Programación genérica

Bases del Lenguaje

- Tipos básico
- Sentencias

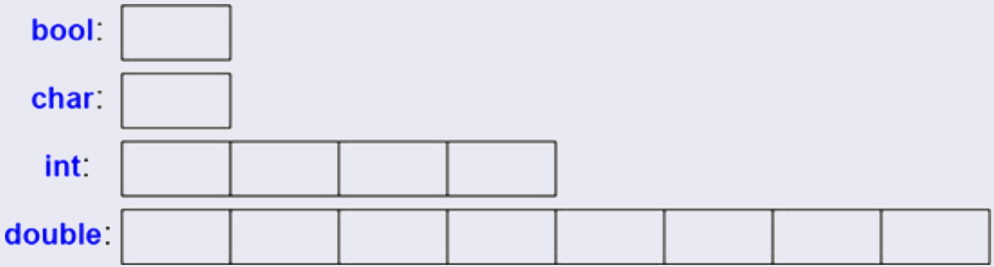
Tipos básicos

- Tipos Básico
- Punteros
- Referencias
- cons
- consexpr

Tipos básicos

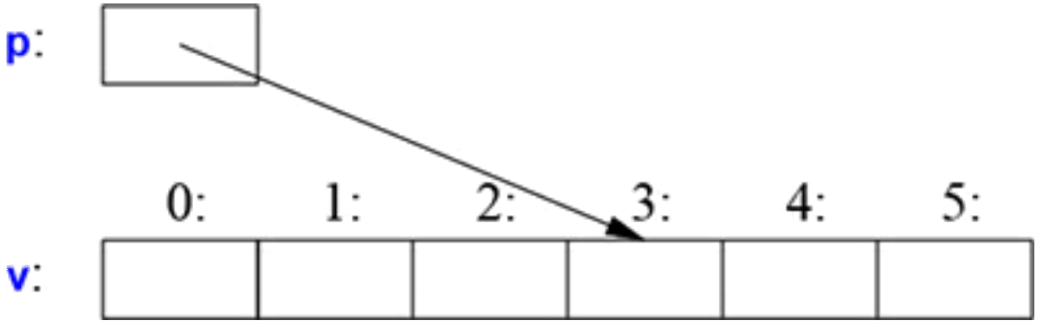
```
bool   // Boolean. Valores posibles true y false
char   // Carácter. Ejemplos: 'a', 'z', y '9'
int    // Entero. Ejemplos: -273, 42, y 1066
double // Número decimal. Ejemplos:-273.15, 3.14, y 6.626e-34
       // Enteros positivos. Ejemplos: 0, 1, y 999
```

Tamaño de los tipos en bytes



Tipos básicos

- Arrays, punteros y referencias



Tipos básicos

- Punteros
- Referencias

```
int num = 3;
int p1* = &num;

int v[10] = {0,1,2,3,4,5,6,7,8,9};

int p* = &v[3];
```

Tipos básicos

Las constantes en C++ son de 2 tipos:

const

- Prohíbe que se modifique una variables.
- Es usado generalmente en la definición de interfaces
- Puede ser calculado en tiempo de ejecución.

constexpr

- Calculado en tiempo de compilación.
- El compilador evalúa la expresión y la sustituye por su valor.

cons y consexpr

```
constexpr double square(double x) { return x*x;}
double sum(const vector<double>&); // sum no puede modificar su argumento.

vector<double> v{1.2,3.4,4.5}; // v no es una constante.

const double s1 = sum(v); // Correcto.
constexpr double s2 = sum(v) // Error: sum(v) no es una expresión constante.
```

Sentencias

- Bloques
  - if
  - switch
- Bucles
  - while
  - for

Ejemplo de bloque de código

```
void BubbleSort(int *A, int n, bool (*fptr)(int,int))
{
  int i,j,temp;
  for (i = 0; i < n; i++)
    for (j = 0; j < n - 1; j++)
    {
      if (fptr(A[j],A[j+1])) // true if SWAP is needed.
      {
        temp = A[j];
        A[j] = A[j+1];
        A[j+1]= temp;
      }
    }
}
```

Definición de tipos

- Estructuras
- Clases
- Uniones
- Enumeraciones

Definición de tipos

Estructuras

- Las estructuras mantienen separados los datos de las operaciones.
- Por defecto los miembros de una estructura son de acceso público.

Clases

- Las clases agrupan los datos y las operaciones sobre estos.
- Esto permite proteger los datos y las operaciones de los usuarios.
- Por defecto los miembros de una clase son de acceso privado.

Estructuras

```
struct Image
{
  int     nrows, ncols; // Number of rows and columns.
  int**   pixels;       // Pointer to image pixels.
};
```

Clases

```
class Image
{
 public:
  Image();
  Image(int n, int m, int g); // Fills pixels n x m with g value
  Image(const Image& I);      // Copy Constructor
  ~Image() ;                  // Destructor

  const Image& operator= (const Image&); // Assign operator

  int  getNrows();  // Rows
  int  getNcols();  // Columns
  int  getGray(int x, int y) const;  // Gray pixel value (x,y).
  void setGray(int x, int y, int g); // Set pixel value (x,y).

  [...]
};
```

Clases

```
class Image
{
  [...]

 private:
  int nrows, ncols; // Number of rows and columns.
  int **pixels;     // Pointer to image pixels.

  void reserveMemory();        // Reserves memory.
  void copy (const Image& I); // Copy I image over current image.
  void freeMemory();           // Frees memory.
};
```

Uniones

Unión

Es una estructura en la que sus elementos ocupan la misma posición de memoria.
Solo se puede escoger uno de los elementos a la vez.

Uniones

```
union Pixel
{
  RGB p;
  int gray;
};
struct Image
{
  int       nrows, ncols; // Number of rows and columns.
  Pixel**   pixels;       // Pointer to image pixels.
  Image();
};
[...]
Image im;
im.pixels[5][5].p.r = 180;
im.pixels[5][5].p.g = 50;
im.pixels[5][5].p.b = 190;
```

Enumeraciones

Enum

Permite definir un conjunto de valores que enumeran una cualidad.
Por ejemplo podemos definir color como rojo, verde y azul.

```
enum color {rojo,verde, azul};
```

Enumeraciones

```
enum ImageType {grayscale, color};
[...]
int main()
{
  Image im{color};

  if (im.type == color)
  {
    im.pixels[5][5].p.r = 180;
    im.pixels[5][5].p.g = 50;
    im.pixels[5][5].p.b = 190;
  }
  else
  {
    im.pixels[5][5].gray = 250;
  }
}
```

Modularidad

Programa C++

Formado por diferentes partes separadas entre sí.

- Interfaz
- Implementación

Modularidad

Interfaces

Se suelen definir en los archivos de cabecera con extensión `.h`

Representan las declaraciones que contienen todo lo necesario para usar una función o tipo.

```
class Image
{
  int nrows, ncols; // Number of rows and columns.
  int **pixels;     // Pointer to image pixels.

  void reserveMemory();        // Reserves memory.
  void copy (const Image& I); // Copy I image over current image.
  void freeMemory();           // Frees memory.

 public:
  Image();
  Image(int n, int m, int g); // Fills pixels n x m with g value
  Image(const Image& I);       // Copy Constructor
  ~Image();                    // Destructor

  const Image& operator= (const Image&); // Assign operator

  int  getNrows();  // Rows
  int  getNcols();  // Columns
```

Modularidad

Implementación

Se definen en uno o varios ficheros con extensión .cpp
Representan las definiciones de los tipos o funciones.

Implementación

```
#include <imageInterface.h>
using namespace std;

Image::Image(int rows, int cols, int g)
{
  nrows = rows;
  ncols = cols;

  pixels = new int*[rows];
  for (int i = 0; i < rows; i++)
  {
    pixels[i] = new int[cols];

    for (int j = 0; j < cols; j++)
    {
      pixels[i][j] = g;
    }
```

Modularidad

Compilación separada

- Los fichero .cpp se compilan por separado generando fichero objeto (.o o .obj).
- Esta compilación separada permite modificar una clase sin necesidad de recompilar el programa entero.

Compilación separada

```cpp
// imageModule.cpp

#include <imageInterface.h>

using namespace std;

int main()
{
  Image im{10,10,200};

  cout << "imageImplementation is finnished." << endl;
}
```

Compilación separada

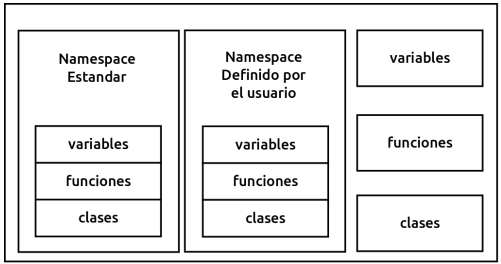
¡Atención!

- Un cambio en un fichero de implementación .cpp implicará recompilar el fichero y realizar el enlazado.
- Un cambio en un fichero de interfaz .h implicará recompilar todos los ficheros .cpp que la incluyen.

Modularidad

Namespaces

- Ofrecen un nivel superior de organización al de las clases y las enumeraciones.
- Permiten evitar conflictos de nombres.



Namespaces

```
namespace gray
{
  class Image
  {
    int nrows, ncols; // Number of rows and columns.
    int **pixels;     // Pointer to image pixels.

    void reserveMemory();        // Reserves memory.
    void copy (const Image& I); // Copy I image over current image.
    void freeMemory();           // Frees memory.

  public:
    Image();
    Image(int n, int m, int g); // Fills pixels n x m with g value
}
```

Namespaces

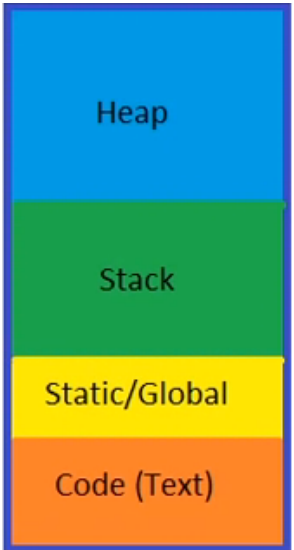
```
namespace rgb
{
  struct RGB
  {
    int r,g,b;
  };

  class Image
  {
    int nrows, ncols; // Number of rows and columns.
    RGB **pixels;     // Pointer to image pixels.

    void reserveMemory();        // Reserves memory.
    void copy (const Image& I); // Copy I image over current image.
    void freeMemory();           // Frees memory.

  public:
```

Namespaces

```
gray::Image::Image(int rows, int cols, int g)
{
 [...]
}
rgb::Image::Image(int rows, int cols, int r, int g, int b)
{
 [...]
}
```

Gestión de memoria

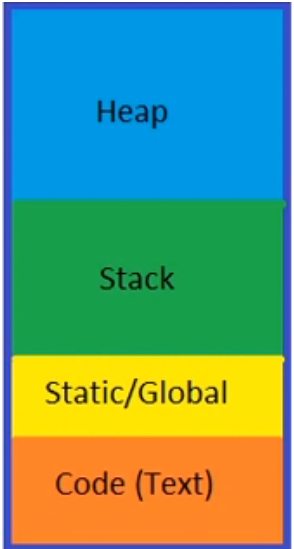
- C++ hereda de C un control fino de la memoria dinámica usada por el programa.
- El sistema operativo divide la memoria de programa en bloques.



Gestión de memoria

Memoria de programa

- Code (Text): Área para el código máquina del programa.
- Static/Global: Área para las variables globales del programa.
- Stack: Área para las variables locales y la pila de programa.
  - El tamaño de esta área es fijo, si se supera se produce el error `Stack Overflow`.
- Heap (Free Store): Área de memoria dinámica.
  - El tamaño de esta área es virtualmente la memoria del equipo.



Gestión de memoria dinámica

Herencia de C

- malloc
- free
- calloc
- realloc

C++

- new
- delete

Gestión de memoria dinámica

```
Pixel**   pixels;       // Pointer to image pixels.

pixels = new Pixel*[rows];
for (int i=0;i<10;i++)
{
  pixels[i] = new Pixel[cols];
}

int * copy;

copy = (int*)malloc(im.nrows*im.ncols*sizeof(int));
```

Clases

Definición

Tipo definido por el usario para representar un concepto.

Principales cualidades

- Agrupan los datos y las operaciones sobre estos.
- Proteger los datos y las operaciones de usuarios maintencionados.
  - Por defecto, sus miembros son de acceso privado.
- Permiten sobrecargar las operaciones para diferente dato de entrada.
- Permiten derivar typos más complejos que heredan sus características básicas.

Clases

Tipos de clases

- Clases concretas
- Clases abstractas
- Clases en las jerarquía de clases

Clases concretas (Tipos concretos)

- En C++ todo son clases por ser un lenguaje orientado a objetos.
- Las clases concretas son similires a los tipos básicos suministrados por las librerías básicas de C++.
- Su representación es parte de su definición, lo que permite que las implementaciones sean eficientes en tiempo y espacio.

Características

- Instancia sus datos en la memoria estática (stack).
- No usa punteros.
- Inicializa automáticamente los objetos.
- Copia y muebe objetos.

```cpp
// pixel.h
#include<stdio.h>
#include <iostream>

using namespace std;

namespace rgb
{
  class Pixel
  {
    int r,g,b; // Pixel components.
  public:
    Pixel(){
      r = g = b = 0;
    }

    Pixel(int r, int g, int b){ // Fills pixel with rgb value.
      this->r = r;
      this->g = g;
```

Clases Abstractas (Tipos Abstractos)

- En C++ todo son clases por ser un lenguaje orientado a objetos.
- Las clases abstractas o tipos abstractos aislan completamente el usuario de los detalles de implementación.
- Las clases abstractas definen interfaces.

Características

- No puede contener variables de clase.
- Instancia sus datos en la memoria dinámica (free store).
- Está compuesta por métodos abstractos.
  - Una clase deribada debe implementar los métodos abstractos de su clase padre.
- No se puede instanciar un objeto de un tipo abstracto.

```cpp
class AbstractPixel
{
public:
  virtual void set(int value) = 0; // Pure virtual.
};

class AbstractImage
{
public:
  virtual AbstractPixel& getPixel(int x, int y) = 0; // Pure virtual.
  virtual void setPixel(int x, int y, const AbstractPixel& pixel) = 0; // Pure
  ↪ virtual.
  virtual int getRows() = 0; // Pure virtual.
  virtual int getCols() = 0; // Pure virtual.
};

class AbstractImageOperations
{
public:
```

Jerarquía de Clases

Las clases pueden tomar funcionalidades de otras clase por medio de la herencia.

Tipos de Herencia

- Simple
  - Se hereda la funcionalidad de una única clase.
- Múltiple
  - Se heredan las funcionalidades de múltiples clases.

```
class Pixel : public AbstractPixel
{
  int r,g,b; // Pixel components.
public:
  Pixel(){
    r = g = b = 0;
  }

  Pixel(int r, int g, int b){ // Fills pixel with rgb value.
    this->r = r;
    this->g = g;
    this->b = b;
  }

  void set(int value) override{ // Implement AbstractPixel interface.
    r = g = b = value;
  }

  void set(int r, int g, int b){
```

```
class Image : public AbstractImage, public AbstractImageOperations
{
  int rows, cols;
  Pixel **pixels;
public:
  Image();
  Image(int n, int m, int r, int g, int b); // Fills pixels n x m with rgb value
  Image(const Image& I);       // Copy Constructor
  ~Image();                    // Destructor

  const Image& operator= (const Image&); // Assign operator

  int  getNrows();  // Rows
  int  getNcols();  // Columns

  Image operator+ (const Image& I); //Add Images
  Image operator- (const Image& I); //Sub Images
  Image operator! ();   // Invierte la Image
```

Funciones Virtuales y Virtuales Puras.

Virtuales

- Permite el poliformismo
- Se definen en la clase base y se pueden redefinir en las clases derivadas.

Virtuales puras

- Se declaran en la clase base y se marcan con $= 0$.
- Es obligatoria la definición en las clases derivadas.

```
class AbstractImage
{
public:
  virtual AbstractPixel& getPixel(int x, int y) = 0; // Pure virtual.
  virtual void setPixel(int x, int y, const AbstractPixel& pixel) = 0; // Pure
  ↪  virtual.
  virtual int getRows() = 0; // Pure virtual.
```

Operaciones esenciales

Librería estándar

Uso de lambdas

Programación genérica

Conclusiones

Proyecto

Editor de imágenes

- Incrustar imagen en otra
- Fundir imagen con otra
- Eliminar objetos de una imagen
- Extraer un trozo de una imagen

Other stuff

Gracias a Gustavo Puche

for the first viable Beamer setup in Org

Other stuff

Gracias a Gustavo Puche

for the first viable Beamer setup in Org

Gracias a alguien más

for contributing to the discussion

Frame 2 (where we will not use columns)

Request

Please test this stuff!

Círculo