C++

Clases y Objetos

Informática Aplicada

DSI-EIE

Abstracciones y tipos de datos

- En un programa C podemos usar variables para representar abstracciones:
  - Tipos “fundamentales”:
    - números naturales (`int`)
    - caracteres (`char`)
    - reales (`float`, `double`)
    - direcciones de memoria: punteros, etc.
  - Tipos “compuestos”: arreglos, estructuras y uniones
- Cada tipo representa un concepto único e inmodificable

Los Tipos de Datos (ii)

- Los datos son manipulados por algoritmos y funciones y el acceso a ellos está limitado al ámbito (variables locales, globales, etc.)
- Es un modelo simple y cómodo para representar información, pero a la vez limitado
- ¿Como represento una cadena de caracteres?

Ejemplo: representamos un sensor en un programa de control

```
struct Sensor {
   char *id;
   char *unidad;
   double medicion;
}
```

¿Cómo lo utilizamos?

```
void main(){
   struct Sensor s1;
   s1.id = “TEMPS001”;
   s1.unidad = “Celsius”;
   …
   printf(“La temperatura del sensor
   %s es de %f grados%s”,s1.id,
   s1.medicion,s1.unidad);
   }
```

Manipulación

- Los miembros de la estructura pueden ser leídos y/o modificados por cualquier instrucción que esté dentro del ámbito
- Incluso puede asignarse valores inconsistentes a los campos, en el ejemplo anterior:

```
s1.unidad = “verde”;
```

- Si cambio el diseño de la estructura afecto a todas las funciones que hacen uso de ella

Clases

- Un concepto nuevo nos permite representar abstracciones preservando su coherencia
- A la idea de tipo de dato compuesto se le agregan tres conceptos fundamentales:
  - Comportamiento
  - Estado interno
  - Encapsulamiento

Clases: Comportamiento

- En lugar de dejar que el programa acceda a la estructura interna, proveen una “función miembro” que brinda ese servicio
- O sea que dentro del nuevo tipo hay código

Nuevo tipo “Sensor”

```
class Sensor {
  char *id;
  char *unidad;
  double medicion;
public:
  double leerMedicion();
}
```

¿Cómo usamos el nuevo tipo?

- Puedo usarlo en C? **NO!**
- Necesito otro lenguaje de programación que soporte clases
- C++

¿Cómo usamos el nuevo tipo?

```
void main(){
  Sensor s1; // creamos una
             // variable del
             // tipo “Sensor”
  …
  printf(“La temperatura es %f”,
s1.leerMedicion());
  …
```

Clases: Estado Interno

- Ahora que las variables en un programa pueden tener "comportamiento" vamos a agregarle otra característica: la capacidad de "recordar" que les pasó durante la ejecución del programa

Sensor con estado

```
class Sensor {
  char *id;
  char *unidad;
  double mediciones[100];
public:
  double leerMedicion();
}
```

Sensor con estado

- En este caso haremos que cada vez que se lea una medición esta quede registrada en el arreglo “mediciones[]”, de forma tal que pueda saberse como evolucionó la temperatura
- Pregunta: ¿qué otro dato sería deseable registrar?
- Una posible respuesta: la marca temporal de cada medición

Sensor con estado y memoria

```
struct Medicion{
    double valor;
    time_t marcaDeTiempo;
};
class Sensor {
  char *id;
  char *unidad;
  struct Medicion[100] mediciones;
public:
  double leerMedicion();
}
```

Clases: Encapsulamiento

- Hemos extendido la funcionalidad de los tipos de datos
- Aún es posible que sus campos sean modificados incorrectamente por el programa que los usa
- Para evitarlo introducimos un nuevo concepto: el encapsulamiento

Encapsulamiento

```
class Sensor {
private:
   char *id;
   char *unidad;
   struct Medicion mediciones[100];
public:
   double leerMedicion();
}
```

Lo que sigue es inaccesible para el resto del programa

Clases y Objetos

- Como muestra el ejemplo anterior, hemos creado un nuevo “tipo de datos”, llamado Sensor
- Ahora en nuestros programas podremos usar variables del tipo “Sensor”
- El nuevo tipo de dato se llama “Clase”
- A una variable del tipo “Sensor” se le denomina “Objeto”

Clases y Objetos

- Cada objeto en el programa es una instancia de su clase
- La declaración "`Sensor s1`" indica que s1 es un objeto, es decir una instancia de la clase Sensor
- s1 comparte con todos las instancias de la clase Sensor su estructura interna y su comportamiento, pero su estado es individual

Clases y Objetos (ii)

- Un programa “orientado a objetos” tiene características particulares:
  - Está escrito en un lenguaje de programación “orientado a objetos”, como C++, Java o Python o ...
  - El funcionamiento del programa (comportamiento) es producto de la interacción entre objetos
  - Usa “tipos de datos extensibles”, es decir, puedo crear los tipos de datos que necesite

Clases y Objetos (iii)

- Además de crear nuevos tipos definiendo clases, puedo utilizar clases existentes, por ejemplo en C++:
  - `string` (cadenas de caracteres)
  - `date` (fecha/hora y funciones temporales)
  - `vector` (arreglos dinámicos)
  - etc.
- En el caso de C++, la librería estándar contiene cientos de clases “listas para usar” en la librería STL

Librería estándar C++

- http://www.cplusplus.com/reference/stl/
-

Objetos

- Como dijimos, una variable cuyo tipo es una clase se llama “objeto”
- Los objetos tienen un “ciclo de vida” en un programa:
  - Son “construidos”
  - Son utilizados
  - Se destruyen al finalizar su vida útil

Objetos (ii)

- En cada etapa del ciclo de vida de un objeto el programa que lo usa emplea recursos de la clase a la que pertenece:
  - Crear un objeto: constructores
  - Usar un objeto: funciones miembro
  - Destruir un objeto: destructores

Constructores

- Crean la estructura interna del objeto
- Establecen el estado inicial, es decir, el valor de las variables internas (campos) del objeto
- Tienen el mismo nombre que la clase, no tienen tipo de retorno y puede haber 0, 1 o mas constructores en una clase

Constructores

```
Sensor::Sensor( char *nombre, char *unid ){
   id = new char[strlen(nombre) + 1];
   strcpy(id, nombre);
   unidad = new char[strlen(unid) + 1];
   strcpy(unidad, unid);

   return;
}
```

Funciones Miembro

- Son piezas de código asociadas al objeto, que ofrecen servicios a otros objetos y al programa
- El tipo y cantidad de funciones miembro es ilimitado
- Las FM tienen acceso irrestricto a los campos del objeto

Funciones miembro: ejemplo

```
double Sensor::LeerMedicion( void ){
   return mediciones[last].val;
}
```

Destructores

- Son funciones especiales que se usan para liberar memoria ocupada por un objeto que no será utilizado
- Tienen el mismo nombre de la clase, precedido por un carácter tilde: ~

Destructores: Ejemplo

```
Sensor::~Sensor(void) {
  delete id;
  delete unidad;
}
```

Operadores

- Disponemos de una variante especial de funciones llamadas “operadores”
- Estas funciones tiene como nombre un patrón especial:
  - `operator+()`
  - `operator>>()`
  - `Operator—()`
- Redefinen el comportamiento de los operadores +, -, *, /,>>,..

Ejemplo con la clase *string*

```
string operator+( const string& lhs, const string& rhs );
```

```
#include <iostream>
#include <string>

void main ()
{
  std::string firstlevel ("com");
  std::string secondlevel ("cplusplus");
  std::string scheme ("http://");
  std::string hostname;
  std::string url;

  hostname = "www." + secondlevel + “.” + firstlevel;
  url = scheme + hostname;

  std::cout << url << '\n';
  return 0;
}
```

Conclusiones

- C++ nos permite crear nuevos tipos de variables que representen nuevas abstracciones
- Estos tipos poseen estado, comportamiento y encapsulamiento
- Un nuevo tipo tiene un conjunto de operaciones definidas
- Puede redefinirse para ellos la semántica de los operadores