Estructura de Datos

Etructuras de datos fundamentales

Dr. Said Polanco Martagón[1], Dra. Karla E. Vázquez Ortiz[2]

Universidad Politécnica de Victoria

February 1, 2018

[1]E-mail address: spolancom@upv.edu.mx
[2]kvazquezo@upv.edu.mx

Outline

TIPOS DE DATOS

- Un tipo de dato es un conjunto de valores y operaciones asociados a esos valores.
- En lenguajes de programación existen una infinidad de tipos de datos entre los que destacan:
  - Primitivos
  - Compuestos
  - Agregados

TIPOS DE DATOS

Datos primitivos

- Son los tipos de datos más simples, también denominados datos atómicos, porque no se construyen a partir de otros tipos y son entidades únicas no descomponibles en otros.
- Ejemplo de tipo de datos atómicos:

Enteros	
valores	$-\infty, ..., -3, -2, -1, 0, 1, 2, 3, ..., +\infty$
operaciones	$\times, +, -, \%, /, ++, --, ...$
Punto flotante	
valores	$-., ..., -0.6, 0.0, 0.5, ..., 2.5,$
operaciones	$\times, +, -, \%, /, ...$
Caracter	
valores	$0, ..., 'A', 'B', ..., 'a', 'b', ...$
operaciones	$<, >, ...$
Lógicos	
valores	$VERDADERO, FALSO$
operaciones	$AND, or, NOT, ...$

TIPOS DE DATOS

Compuestos y agregados

- Los datos compuestos se pueden romper en subcampos que tengan significado. Un ejemplo sencillo es un número de teléfono 528341101021.
- Los tipos agregados son tipos de datos cuyos valores son colecciones de elementos de datos. Un tipo agregado se compone de tipos de datos previamente definitivos. Existen tres tipos agregados básicos: arrays, secuencias y registros.

TIPOS DE DATOS

Arrays

Es normalmente, una colección de datos de tamaño o longitud fija, cada uno de estos datos es accesible a través de los subíndices o índices correspondientes. Todos los elementos de un array deben ser del mismo tipo.

Array de enteros: [4, 6, 8, 35, 46, 810]

Cadena

Es un array cuyo tamaño puede variar en tiempo de ejecución. Por consiguiente, las secuencias son similares a arrays dinámicos o flexibles.

Cadena = "Universidad Politecnica de Victoria"

TIPOS DE DATOS

Registro

Un registro se puede considerar como un tipo o colección de datos de tamaño fijo en la que los campos pueden ser de diferentes tipos. A los campos de los registros se accede mediante identificadores.
En C++, el tipo registro se denomina estructura. Una estructura (struct) es un tipo definido por el usuario compuesto de otros tipos de variables. La sintaxis básica es:

```
struct nombreEstructura
   {
        tipo nombreVar;
        tipo nombreVar;
         ...
   };
```

TIPOS DE DATOS

Estructura de datos

Es un agregación de tipo de datos compuestos y atómicos en un conjunto con relaciones bien definidas. Una estructura significa un conjunto de reglas que contienen los datos en conjunto.

Outline

ARRAYS

Un array (lista o tabla) o arreglos es una secuencia de objetos del mismo tipo. Los objetos se llaman elementos y se numeran consecutivamente partiendo de 0, esta numeración se denomina índice. Los elementos almacenados en el array pueden ser de cualquier tipo de dato de C++, incluyendo clases definidas por el usuario.

ARRAYS

Declaración de un array

Un array se declara de modo similar a otros tipos de datos, excepto que se debe indicar al compilador el tamaño o longitud del array. La sintaxis para declarar un array de una dimensión determinada es:

```
tipo nombreArray[numeroDeElementos];
```

ARRAYS

Inicialización de un array

Cuando se inicializa un array, el tamaño del array se puede determinar automáticamente por las constantes de inicialización. Estas constantes se separan por comas y se encierran entre llaves, como en los siguientes ejemplos:

```
 int numeros[6] = {10, 20, 30, 40, 50, 60};
int n[] = {3, 4, 5} //Declara un array de 3 elementos
char c[] = {'L','u','i','s'}; //Declara un array de 4 elementos
char s[] = "Mona Lisa";
```

ARRAYS

Ejemplo

```
#include <iostream >
using namespace std;
const int NUM = 8;
int main()
{
 int nums[NUM];
 int total = 0;
 for (int i = 0; i < NUM; i++)
 {
 cout << "Por favor, introduzca el número ";
 cin >> nums[i];
 total += nums[i];
 }
 cout << "El total de la suma de números es" << total << endl;
 return 0;
}
```

Outline

ARRAYS MULTIDIMENSIONALES

Los arrays o arreglos multidimensionales son aquellos que tienen más de una dimensión y, en consecuencia, más de un índice. Los arrays más usuales son los unidimensionales y los de dos dimensiones, conocidos también por el nombre de tablas o matrices. Algunos ejemplos de declaración son:

```
char Pantalla[25][80];
int equipos[4][30];
double matriz[4][2];
```

ARRAYS MULTIDIMENSIONALES

Inicialización de un array

Los arrays multidimensionales se pueden inicializar, al igual que los de una dimensión, cuando se declaran. La inicialización consta de una lista de constantes separadas por comas y encerradas entre llaves, como en las siguientes definiciones:

```
int tabla[2][3] = {51, 52, 53, 54, 55, 56};
int tabla[2][3]= { {51, 52, 53},
   {54, 55, 56} };
int tabla[2][3]= { {51, 52, 53}, {54, 55, 56} };
int tabla[2][3]= {
    {51, 52, 53},
    {54, 55, 56}
    };
int tabla[3][4]= {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
    };
```

ARRAYS MULTIDIMENSIONALES

Acceso a los elementos de los arrays bidimensionales

En un arreglo bidimensional se accede a los elementos especificando la fila y la columna.
inserción de elementos:

```
<nombre array>[índice fila][índice columna]= valor elemento;
```

extracción de elementos:

```
<variable> = <nombre array> [índice fila] [índice columna];
```

ARRAYS MULTIDIMENSIONALES

Ejemplo

```
const int N = 2;
const int M = 4;
float discos[N] [M];
int fila, col;
for (fila = 0; fila < N; fila++){
  for (col = 0; col < M; col++){
    cin >> discos[fila][col];
  }
}
// Visualizar la tabla
for (fila = 0; fila < N; fila++){
  cout << "Precio fila " << fila << " : ";
  for (col = 0; col < M; col++){
    cout << discos[fila][col] << " ";
  }
  cout << endl;
}
```

ARRAYS DE MAS DE DOS DIMENSIONES

Un array tridimensional se puede considerar como un conjunto de arrays bidimensionales combinados juntos para formar, en profundidad, una tercera dimensión. El cubo se construye con filas (dimensión vertical), columnas (dimensión horizontal) y planos (dimensión en pro fundidad). Por consiguiente, un elemento dado se localiza especificando su plano, fila y columna. Una definición de un array tridimensional es:

```
int nomArray[3][15][10];
```

ARRAYS DE MAS DE DOS DIMENSIONES

Un ejemplo típico de un array de tres dimensiones es el modelo libro, en el que cada página de un libro es un array bidimensional construido por filas y columnas. Supongamos que el libro tiene 500 páginas, cada página tiene cuarenta y cinco líneas y cada linea consta de ochenta caracteres, por lo tanto, existirán quinientos planos y el número de elementos será $500 \times 80 \times 45 = 1.800.000$.

Outline

CADENAS

- Una cadena es un tipo de dato compuesto, un array de caracteres (char), terminado por un carácter nulo ('\0'), NULL.
- Cuando la cadena 'ABC' aparece dentro de un programa se verá como si se almacenarán cuatro elementos: 'A', 'B', 'C' y '\0'.
- El número total de caracteres de un cadena es siempre igual a la longitud de la cadena más uno

CADENAS

Inicialización de variables de cadena

La inicialización de una variable cadena en su declaración se realiza mediante una secuencia de caracteres encerrados entre comillas ("). Ejemplo:

```
char texto[81] = "Esto es una cadena";
char cadenatest[] = "¿Cuál es la longitud de esta cadena?";
char* ptrCadena = "Porcentajes de oliva";
```

y fuera de su declaración:

```
char buff[121];
strcpy(buff, "Aniversario del Quijote");
```

CADENAS

Lectura de cadenas

La lectura usual de datos es con el objeto cin y el operador >> . Por ejemplo:

```
char nombre[30];     // Define array de caracteres
cin >> nombre;       // Lee la cadena Nombre
cout << nombre;
```

El problema surge cuando se requiere introducir mas de un nombre, ya que el comando cin termina la operación de lectura cuando encuentra un espacio en blanco

CADENAS

Lectura de cadenas

El método recomendado será utilizar una función denominada getline(), en unión con cin , en lugar del operador >>. Ejemplo

```
#include <iostream>
using namespace std;
int main()
{
char nombre[80];
cout << "Introduzca su nombre ";
cin.getline(nombre, sizeof(nombre));
cout << "Hola " << nombre << " ¿cómo está usted?" << endl;
return 0;
}
```

CADENAS

Ejercicio

Un texto de n líneas tiene ciertos caracteres que se consideran comodines. Hay dos comodines, el # y el ? El primero indica que se ha de sustituir por la fecha actual, en formato día (nn) de mes (nombre) año (aaaa), por ejemplo 21 de abril 2001. El otro comodín indica que se debe reemplazar por un nombre. Escribir un programa que lea las líneas del texto y cree un array de cadenas, cada elemento referencia a una cadena que es el resultado de realizar las sustituciones indicadas. La fecha y el nombre se ha de obtener del flujo de entrada.

Outline

string.h

La biblioteca de cadena string.h contiene las funciones de manipulación de cadenas utilizadas más frecuentemente. El uso de las funciones de cadena tienen una variable parámetro declarada similar a:

```
char *s1
```

string.h

Función	Cabecera de la función y prototipo
memcpy()	$void * memcpy(void * s1, constvoid * s2, size_t n);$ Reemplaza los primeros n bytes de $*s1$ con los primeros n bytes de $*s2$. Devuelve $s1$.
strcat()	$char * strcat(char * destino, constchar * fuente);$ Añade la cadena fuente al final de destino, concatena.
strchr()	$char * strchr(char * s1, intch);$ Devuelve un puntero a la primera ocurrencia de ch en $s1$. Devuelve NULL si ch no está en $s1$.
strcmp()	$intstrcmp(constchar * s1, constchar * s2);$ Compara alfabéticamente la cadena $s1$ a $s2$ y devuelve: 0 si $s1 = s2$ < 0 si $s1 < s2$ > 0 si $s1 > s2$

Tarea

Investigar: El resto de las funciones string

Outline

La clase string

Muchas implementaciones de C++, como DEV-C++, especializan un contenedor para el tipo char. Esta especialización se le da el nombre string . De esta forma el usuario puede utilizar clase string para manejar cadenas. Por ejemplo:

```
string mipueblo = "Victoria";
string rotulo;
rotulo = "Lista de estudiantes\n";
```

La clase string

Variables string

Al ser string una clase se pueden definir variables, punteros, arrays... de tipo string como de cualquier otras clase. La inicialización se hace llamando a los respectivos constructores:

```
string vacia;
string tema("Marionetas de la vida");
```

Tambien se puede inicilizar con un literal:

```
string texto = "Esto es una cadena";
```

La clase string

Concatenación

El operador + permite aplicarse sobre objetos string para dar como resultado otro objeto string que es la unión o concatenación de ambas. Por ejemplo:

```
string c1 = "Ángela";
string c2 = "Paloma";
string c3 = c1 + c2; // genera una nueva cadena: AngelaPaloma
string cd ("clásica");
cd = "Música" + cd; // genera la cadena Musicaclasica
```

La clase string

Comparación de cadenas

La clase string redefine los operadores relacionales con el fin de comparar cadenas alfabéticamente. Estos métodos comparan los caracteres de dos cadenas utilizando el código numérico de su representación.

```
string c1 = "Universo Jamaicano";
string c2 = "Universo Visual";
if (c1 < c2)
cout << c1 << ", es alfabéticamente menor que " << c2 << endl;
else if (c1 > c2)
cout << c1 << ", es alfabéticamente mayor que " << c2 << endl;
cout << "Entrada de cadenas, termina con FIN" << endl;
// bucle condicional, termina con una cadena clave
while (c1 != "FIN")
{
cin >> c1;
cout << "Cadena leída: " << c1 << endl;
}
```

Outline

Estructuras

Una estructura (struct) es un tipo de dato definido por el usuario que puede encapsular uno o más tipos existentes. La sintaxis básica es:

```
struct nombreEstructura
{
tipo1 nombreVar1;
tipo2 nombreVar2;
// ...
};
```

Estructuras

Una estructura se utiliza cuando se tratan elementos que tienen múltiples propiedades. Por ejemplo, se desea escribir una aplicación que gestione los empleados de una empresa. Cada empleado tiene características diferentes: número de identificación del empleado, nombre, salario, edad, etc.

```
struct empleado
{
unsigned int id;
string nombre;
float salario;
int edad;
};
```

Estructuras

La sintaxis empleada para referenciar a los miembros individuales de una estructura utiliza el operador " . " para acceder a los mismos.

```
miVar.nombremiembro = valor;
```

En el caso de conocer los valores de los miembros de la estructura cuando se crea una variable nueva del tipo estructura, se pueden asignar simultáneamente dichos valores:

```
empleado e1 = {4044, "Marcos", 1499, 66};
```

Outline

Enumeraciones

Un tipo enumerado, es un tipo de dato cuyos valores se definen por una lista de constantes de tipo entero (int). Los tipos enumerados pueden definir sus propias secuencias de modo que se puedan declarar variables con valores en esas secuencias.

Enumeraciones

Tipo enumerado diasMes

```
 enum diasMes { DIAS_ENE = 31, DIAS_FEB = 28, DIAS_MAR = 31,
DIAS_APR = 30, DIAS_MAY = 31, DIAS_JUN = 30,
DIAS_JUL = 31, DIAS_AGO = 31, DIAS_SEP = 30,
DIAS_OCT = 31, DIAS_NOV = 30, DIAS_DIC = 31 };
```

Nota En un tipo enumerado si no se especifica ningún valor numérico, a los identificadores de la definición se asignan valores consecutivos, comenzando con 0.

Enumeraciones

Ejemplo

```
#include <iostream>
using namespace std;
int main()
{
enum color {BLANCO, AZUL, VERDE, ROJO};
color rotulador = ROJO;
int x;
cout << "\n El color es " << rotulador << endl;
cout << "Introduzca un valor: "; cin >> x;
rotulador = (color)x;
if (rotulador == ROJO)
cout << "El rotulador es rojo" << endl;
else if (rotulador == VERDE)
cout << "El rotulador es verde" << endl;
else if (rotulador == AZUL)
cout << "El rotulador es azul" << endl;
else if (rotulador == BLANCO)
cout << "El rotulador es blanco" << endl;
else
cout << "El color es indefinido" << endl;
return 0;
}
```

Outline

Argumentos

Los argumentos pasados en la línea de órdenes del DOS a un programa son recibidos por la función main(). Existen dos variables predefinidas dentro del lenguaje que reciben los argumentos que se pasan al ejecutar un programa.

	Tipo	Labor que realiza
argc	entero	contiene el número de argumentos que se han introducido.
argv	array	array de punteros a caracteres.

La variable argc como mínimo valdrá 1, ya que el nombre del programa se toma como primer argumento, almacenado con argv[0], que es el primer elemento de la matriz. Cada elemento del array apunta a un argumento de la línea de órdenes. Todos los argumentos de la línea de ordenes son cadenas.

Argumentos

La variable argc como mínimo valdrá 1, ya que el nombre del programa se toma como primer argumento, almacenado con argv[0], que es el primer elemento de la matriz. Cada elemento del array apunta a un argumento de la línea de órdenes. Todos los argumentos de la línea de ordenes son cadenas. Ejemplo

```
#include <iostream>
using namespace std;
main(int argc, char *argv[])
{

if(argc<2) {
cout<<"Ha olvidado su nombre.\n";
exit(1);
}
cout<<"Hola "<< argv[1]<<endl);
}
```

Archivos

Operaciones de escritura en archivos

El archivo de cabecera fstream.h define las clases ifstream, ostream y fstream para operaciones de lectura, escritura y lectura/escritura en archivos respectivamente. Para trabajar con archivos debemos crear objetos de éstas clases de acuerdo a las operaciones que deseamos efectuar.

Empezamos con las operaciones de escritura, para lo cual básicamente declaramos un objeto de la clase ofstream, después utilizamos la función miembro open para abrir el archivo, escribimos en el archivo los datos que sean necesarios utilizando el operador de inserción y por último cerramos el archivo por medio de la función miembro close.

Archivos

```
 //*********************************************
//  Escritura.cpp
//  Demuestra la escritura básica en archivo
//*********************************************

#include <fstream.h>
int main()
{
    ofstream archivo;  // objeto de la clase ofstream

    archivo.open("datos.txt");

    archivo << "Primera línea de texto" << endl;
    archivo << "Segunda línea de texto" << endl;
    archivo << "Última línea de texto" << endl;

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

Archivos

En el programa se ha creado un objeto de la clase ofstream llamado archivo, posteriormente se utiliza la función miembro open para abrir el arcivo especificado en la cadena de texto que se encuentra dentro del paréntesis de la función. Podemos invocar a la función constructora de clase de tal manera que el archivo también se puede abrir utilizando la siguiente instrucción:

```
ofstream archivo("datos.txt");  // constructora de ofstream
```

Archivos

Vamos a crear un archivo mediante un objeto de la clase ofstream, y posteriormente lo leeremos mediante un objeto de la clase ifstream:

```
 #include <iostream>
#include <fstream>
using namespace std;

int main() {
   char cadena[128];
   ofstream fs("nombre.txt");  // Crea un archivo de salida
   fs << "Hola, mundo" << endl; // Enviamos una cadena al archivo de salida:
   fs.close(); // Cerrar el archivo,
   ifstream fe("nombre.txt");   // Abre un archivo de entrada
   // Leeremos mediante getline, si lo hiciéramos
   // mediante el operador << sólo leeríamos
   // parte de la cadena:
   fe.getline(cadena, 128);
   cout << cadena << endl;
   return 0;
}
```

Archivos

Veamos otro ejemplo sencillo, para ilustrar algunas limitaciones del operador >> para hacer lecturas, cuando no queremos perder caracteres. Supongamos que llamamos a este programa "streams.cpp", y que pretendemos que se autoimprima en pantalla:

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
   char cadena[128];
   ifstream fe("streams.cpp");

   while(!fe.eof()) {
      fe >> cadena;
      cout << cadena << endl;
   }
   fe.close();

   return 0;
}
```

Explicar el resultado obtenido

Archivos

El resultado quizá no sea el esperado. El motivo es que el operador >> interpreta los espacios, tabuladores y retornos de línea como separadores, y los elimina de la cadena de entrada. Se soluciona modificando la siguiente linea:

```
fe.getline(cadena, sizeof(cadena));
```

Archivos binarios

Para algunos sistemas operativos todos los archivos son binarios. En esencia esto es más correcto, puesto que un archivo de texto es un fichero binario con un rango limitado para los valores que puede almacenar.

```
 #include <iostream>
#include <fstream>
 using namespace std;
 int main () {
    fstream ficheroMp3;
    char marca[5];
    ficheroMp3.open ("Cancion.mp3", ios::in | ios::binary);
     if (ficheroMp3.is_open()) {  // Compruebo si he podido abrir
          ficheroMp3.seekg(-128, ios::end);  // Me coloco donde empieza el ID3
          ficheroMp3.read(marca, 3); // Leo 3 bytes
          if ((marca[0] != 'T') || (marca[1] != 'A') || (marca[2] != 'G')){  //
            cout << "Sin datos de artista" << endl;
            cout << marca[0]<< marca[1] << marca[2] <<  endl;}
        else
            cout << "Parece un MP3 con ID3 TAG" << endl;
          ficheroMp3.close(); // Finalmente, cierro
    }
        else cout << "Archivo inexistente" << endl; // Si no he podido abrir, a
    return 0;
```

Archivos

Modos de apertura de archivo

- **ios::app** Operaciones de añadidura.
- **ios::ate** Coloca el apuntador del archivo al final del mismo.
- **ios::in** Operaciones de lectura. Esta es la opción por defecto para objetos de la clase ifstream.
- **ios::out** Operaciones de escritura. Esta es la opción por defecto para objetos de la clase ofstream.
- **ios::nocreate** Si el archivo no existe se suspende la operación.
- **ios::noreplace** Crea un archivo, si existe uno con el mismo nombre la operación se suspende.
- **ios::trunc** Crea un archivo, si existe uno con el mismo nombre lo borra.
- **ios::binary** Operaciones binarias.