Revista CENIC Ciencias Biológicas, 22, 3, 1991

TIPOS GENERICOS EN TURBO PASCAL 6.0 Y TURBO C++ 1.0

Rafael Sánchez Aroche

Centro de Neurociencias, Centro Nacional de Investigaciones Científicas

ABSTRACT. This paper shows implementation of the genericity resource in Turbo Pascal (version 6.0) and Turbo C++ (version 1.0) languages, which is an important mechanism not supported by these languages and which is greatly useful for implementation of abstract data types.

RESUMEN. En el trabajo se exponen algunas instrumentaciones del recurso: definición de tipos genéricos, en los lenguajes Turbo Pascal (versión 6.0) y Turbo C++ (versión 1.0), importante mecanismo que no es soportado por estos lenguajes y que es de gran utilidad para la instrumentación de tipos de datos abstractos.

INTRODUCCION

La definición de tipos genéricos es un recurso que se brinda en algunos lenguajes de programación(1) y su necesidad es algo real, sobre todo cuando se trata de la instrumentación de tipos de datos abstractos (TDA), por ejemplo los tipos Pila, Lista, etc. Contar con esta posibilidad permite definir operaciones sobre un tipo abstracto (o sea, genérico), lo cual significa que no se conoce el tipo en el momento en que se define la operación, y posteriormente se puede hacer una correspondencia de este tipo genérico con alguno de los predefinidos por el sistema o por el usuario. En dependencia de la aplicación puede desearse hacer corresponder en un mismo programa un tipo genérico a diferentes tipos concretos.

Aunque ni en Turbo Pascal (hasta la versión 6.0),(2) ni en Turbo C++ (hasta la versión 1.0),(3) se brinda esta posibilidad de definición de tipos genéricos, a lo largo del artículo se verán algunas posibles instrumentaciones para su simulación en estos lenguajes.

Turbo Pascal

Primero presentaremos el caso en el que se desea hacer una sola evaluación del tipo genérico en todo el programa y después en el que se desea hacer varias, por ejemplo para un tipo lista, puede desearse tener lista de enteros, lista de reales, etc.

Caso particular

En Turbo Pascal (TP) es posible instrumentar este caso, de una manera bastante elegante, mediante el uso de directivas de compilación condicional. La idea consiste en definir un módulo (unit) que contenga el tipo abstracto T y las operaciones sobre él. Para evaluarlo a algún tipo concreto se utilizan directivas de compilación condicional que permitan asociar el tipo T con un tipo determinado. Por ejemplo, supongamos que se desea la instrumentación de algunas operaciones sobre los tipos numéricos (enteros y reales), entonces puede definirse un tipo genérico NUMERIC y todos los posibles valores para este tipo:

```
Type

 {$IFDEF Byte}
   NUMERIC = Byte;
 {$ELSE}
 {$IFDEF Word}
   NUMERIC = Word;
 {$ELSE}
 {$IFDEF Real}
   NUMERIC = Real;
 {$ELSE}
 {$IFDEF Double}
   NUMERIC = Double;
 {...}
 {$ENDIF} {$ENDIF} {$ENDIF} {$ENDIF}
```

Como se observa, esta vía de solucionar la definición de tipos genéricos tiene sus ventajas y desventajas. Al utilizar símbolos condicionales, la evaluación del tipo genérico (NUMERIC) a un tipo concreto, se resuelve en compilación y por tanto no se pierde la filosofía de fuerte chequeo de tipos defendida por TP, tampoco es necesario modificar el programa fuente para hacer dicha correspondencia; pero por otro lado es necesario definir todos los tipos concretos con los cuales se puede hacer corresponder el tipo genérico. Está claro que esto juega con la aplicación concreta a desarrollar, pues en muchos casos sólo se desea trabajar con determinado conjunto de tipos (conocidos). Debe tenerse en cuenta que no basta con definir símbolos condicionales, sino que debe recompilarse la unit para lograr la evaluación del tipo, por lo cual es necesario disponer del fuente de la unit, con todos los inconvenientes que esto trae.

En el siguiente ejemplo se define el TDA List Of NUMERIC como un objeto de Turbo Pascal. No es interés ahora la forma en que se instrumenta la lista, sino cómo es que su tipo base puede cambiar según el símbolo condicional definido.

```
Unit ADTList;

INTERFACE

  Type

  {$IFDEF Byte}
    NUMERIC = Byte;
  {$ELSE}
  {$IFDEF Word}
    NUMERIC = Word;
  {$ELSE}
  {$IFDEF Real}
    NUMERIC = Real;
  {$ELSE}
  {$IFDEF Double}
    NUMERIC = Double;
{ .... }
{$ENDIF} {$ENDIF} {$ENDIF} {$ENDIF}

Const
  MaxItems = 100;

Type

  List = Object
           Total:Word;
           Items:Array[1..MaxItems] Of Numeric;
           Procedure Init;
           Procedure Include( Value:Numeric );
           { ... }
         End;

IMPLEMENTATION

Procedure List.Init;
Begin
 Total : = 0
End;
```

```
Procedure List.Include;
Begin
 If Total = MaxItems Then
     writeln( 'Error: Demasiados elementos' )
 else
  begin
    inc(Total);
    Items[Total]:= Value;
  end
End;

End.
```

En un programa basta con definir el símbolo condicional asociado a un tipo concreto y recompilar el módulo ADTList para evaluar el tipo NUMERIC a dicho tipo.

```
Program Instance;

Uses AdtList;

{$DEFINE Real}

Var L:List;

Begin
  L.Init;
  L.Include(1.2);
End.
```

Caso general

En general para lograr la instrumentación de tipos genéricos se establece un compromiso con la filosofía de fuertes chequeos de tipos y la eficiencia.

Cuando se instrumenta un TDA con un tipo genérico, generalmente debe existir una primitiva (en ocasiones, llamada CREATE, INIT, etc.) la cual se le indique con qué tipo se desea asociar el tipo genérico. Así de alguna manera se llega a conocer el tamaño del tipo de dato con que se trabaja, y en dependencia del universo de la aplicación es utilizado o bien el tamaño o bien el tipo del dato. Posteriormente cada primitiva (que está relacionada con el tipo) podrá diferenciar mediante una instrucción condicional (CASE o IF) determinadas acciones en dependencia del tipo, o simplemente trabajar con la zona de memoria que ocupa el tipo, teniendo en cuenta su tamaño.

Con esta forma de instrumentación, se permite la evaluación de un tipo genérico a varios tipos concretos en un mismo programa. Además, se resuelve también la evaluación de un tipo genérico a cualquier tipo que pueda construir el usuario, aún después de la instrumentación de las operaciones sobre el tipo genérico; pero se afecta el fuerte chequeo de tipos o la eficiencia (o ambos inclusive).

Esta solución implica, en general, trabajo con memoria dinámica y traspaso de parámetros sin tipo, lo cual influye en la claridad y la eficiencia de la instrumentación de las operaciones sobre el tipo.

En el siguiente ejemplo se define el TDA List, y se puede hacer corresponder a cualquier tipo definido por el sistema o por el usuario, para ello basta indicar el tamaño que ocupan las variables de este tipo.

```
Unit ADTList;

INTERFACE

Type

Buff = Array[1..1] Of Byte;
List = Object
         Total     :Word;
         Items     :Buff;
         MaxItems  :Word;
         ItemSize  :Word;
         Procedure Init( ISize:Word; MaxI:Word );
         Procedure Include( Var Value );
         { ... }
       End;

IMPLEMENTATION

  Procedure List.Init;
  Begin
     getmem( Items, ISize*MaxI );
     Total := 0;
     MaxItems := MaxI;
     ItemSize := ISize;
  End;

  Procedure List.Include;
  Begin
     If Total = MaxItems Then
        writeln( 'Error: Demasiados elementos' )
     else
      begin
        inc(Total);
        move( Value,Items^[(Total-1)*ItemSize + 1], ItemSize );
      end
  End;

End.
```

El siguiente programa indica en el método Init, el tamaño del tipo real y con eso se evalúa el tipo genérico a un tipo concreto.

```
Program Instance;

Uses Adtlist;

Var L:List;
    R:Real;

Begin
 R:= 1.3;
 L.Init( SizeOf(REAL),100 );
 L.Include(R);
End.
```

TURBO C++

En esta sección se analizará sólo la instrumentación del caso general debido a que el caso particular se puede programar de manera similar a como se vio en Turbo Pascal.

En Turbo C++ (TC++) además de contar con el recurso de directivas de compilación condicional, se puede aprovechar el mecanismo de definición de macros.

Si se considera que un TDA es una clase de C++, entonces vinculando el mecanismo de definición de macros con el de la herencia, se puede lograr la definición de tipos genéricos de una manera más elegante que en TP.

Una posible vía es definir una clase en la que los operadores trabajen de la manera como se vio en TP. Para no perder el recurso de fuerte chequeo de tipos, se definen macros que al expandirse (por el preprocesador), se genere una herencia de esta clase; en la nueva clase (resultado de la herencia) se redefinen aquellos métodos que tengan que ver con el tipo abstracto, de forma que pueda ser chequeado en compilación el traspaso de parámetros a estos métodos. Para no perder en eficiencia, estos métodos pueden ser declarados INLINE.

En el siguiente ejemplo se define el TDA List y se puede hacer corresponder el tipo de la lista a algunos de los tipos predefinidos por TC++ o algún nuevo tipo creado por el usuario.

Primero se definen dos macros que son las que el usuario debe utilizar desde sus programas.

```
#include "adt.cpp"

#define instance_list(t) class list##t:public adtlist
{ public:
    list##t(void):adtlist( sizeof(t) ){};
    void add( t value ){ adtlist::add(&value); };
}

#define list(t) list##t
```

La macro instance_list, se utiliza para indicar el tipo al que se desea hacer corresponder el tipo genérico y debe ser invocada una sola vez por cada tipo. La macro list, permite crear un objeto de la nueva clase y puede ser invocada las veces que se desee. Nótese que lo mejor es haber podido combinar ambas macros en una sola, mediante el uso de directivas de compilación condicional, para diferenciar la primera vez que se invoca la macro instance_list (definiendo la clase), de las demás (referenciando la clase definida). Lamentablemente TC++ no permite la mezcla de las macros #define con las de compilación condicional (#if, #ifdef, #undef, etc.). La clase adtlist puede definirse como:

```
#include <stdio.h>
#include <mem.h>

class adtlist
{ protected:
    #define max 100
    typedef char buff[1];
    buff *items;
    int total;
    int itemsize;
  public:
    adtlist(int size)
     { total = 0; items = new buff[max*size];
       itemsize = size; };
    void add( void *ptrvalue );
};

void adtlist::add( void*ptrvalue )
{ if (total = = max)
    printf("Error: Demasiados elementos");
  else
  {
   movmem( ptrvalue, &(*items)[total + + *itemsize],itemsize );
  };
}
```

En el siguiente programa se utiliza el TDA list y se evalúan los tipo int y double.

```
#include "list.cpp"
instance_list(int);
instance_list(double);

list(int) l1;
list(double) l2;

main()
{
  l1.add(1);
  l2.add(2.3);
}
```

Esta forma de instrumentar tipos de datos abstractos con tipos genéricos es elegante, clara y no se compromete la eficiencia.

Un inconveniente presente es la notificación de errores por parte del compilador cuando se utilizan inadecuadamente estas macros, pues debido a la expansión de las macros surgen nuevos símbolos que pueden resultarles desconocidos al usuario. No obstante, queda abierta la posibilidad de que el usuario no utilice estas macros y defina explícitamente la herencia a partir de la clase AdtList para evaluar el tipo genérico a un tipo concreto.

CONCLUSIONES

La manera en que se ha expuesto la simulación del uso de tipos genéricos en estos lenguajes es una vía de aumentar la potencialidad de los mismos (aún cuando estos no disponen de este recurso), logrando el desarrollo de aplicaciones con este concepto tan importante. No obstante, dependiendo del universo de la aplicación a desarrollar, las propuestas anteriores pueden atentar contra la eficiencia, claridad, o contra el fuerte chequeo de tipos, e incluso para algunos casos, pueden resultar no aplicables.

BIBLIOGRAFIA

1. YOUNG, S. J.: *An introduction to Ada*, Second Edition, John & Sons, 1984.
2. *Turbo Pascal version 6.0 user's guide*, Borland Inc., 1990.
3. *Turbo C++ 1.0 user's guide*. Borland Inc., 1990.

SISTEMA DE INTRODUCCION CROMATOGRAFICA

Sistema automático para la introducción cromatográfica de desorción térmica-crioconcentración directa (D.T-C.C.D), diseñado en base a experiencias de laboratorio sobre procedimientos de introducción de sustancias extraídas y preconcentradas en polímeros porosos y por la técnica de desorción térmica controlada con el efecto de crioconcentración directa. El mismo es acoplable directamente a cualquier cromatógrafo de gases, permitiendo la introducción de microcantidades de muestras en columnas de relleno.

PRODUCIDO Y EXPORTADO POR:
PRODUCED AND EXPORTED BY:
CNIC