**Динамические структуры данных: списки**

**Введение**

В предыдущих обзорах мы рассматривали программирование, связанное с обработкой только **статических данных. Статическими величинами** называются такие, память под которые выделяется во время компиляции и сохраняется в течение всей работы программы.

В языках программирования (Pascal, C, др.) существует и другой способ выделения памяти под данные, который называется **динамическим**. В этом случае память под величины отводится во время выполнения программы. Такие величины будем называть **динамическими**. Раздел оперативной памяти, распределяемый статически, называется **статической памятью**; динамически распределяемый раздел памяти называется **динамической памятью (динамически распределяемой памятью)**.

Использование динамических величин предоставляет программисту ряд дополнительных возможностей. Во-первых, подключение динамической памяти позволяет увеличить объем обрабатываемых данных. Во-вторых, если потребность в каких-то данных отпала до окончания программы, то занятую ими память можно освободить для другой информации. В-третьих, использование динамической памяти позволяет создавать структуры данных переменного размера.

Работа с динамическими величинами связана с использованием еще одного типа данных — **ссылочного типа**. Величины, имеющие ссылочный тип, называют **указателями**.

**Указатель** содержит адрес поля в динамической памяти, хранящего величину определенного типа. Сам указатель располагается в статической памяти.

**Адрес величины** — это номер первого байта поля памяти, в котором располагается величина. Размер поля однозначно определяется типом.

Далее будем более подробно обсуждать указатели и действия с ними в языке Pascal, примеры будем приводить на Pascal и C.

Величина ссылочного типа (указатель) описывается в разделе описания переменных следующим образом:

Var <идентификатор> : ^<имя типа>;

Вот примеры описания указателей:

Type Mas1 = Array[1..100] Of Integer;

Var P1 : ^Integer;

P2 : ^String;

Pm : ^Mas1;

Здесь P1 — указатель на динамическую величину целого типа; P2 — указатель на динамическую величину строкового типа; Pm — указатель на динамический массив, тип которого задан в разделе Type.

Сами динамические величины не требуют описания в программе, поскольку во время компиляции память под них не выделяется. Во время компиляции память выделяется только под статические величины. Указатели — это статические величины, поэтому они требуют описания.

Каким же образом происходит выделение памяти под динамическую величину? Память под динамическую величину, связанную с указателем, выделяется в результате выполнения стандартной процедуры NEW. Формат обращения к этой процедуре:

NEW(<указатель>);

Считается, что после выполнения этого оператора создана динамическая величина, имя которой имеет следующий вид:

<имя динамической величины> := <указатель>^

Пусть в программе, в которой имеется приведенное выше описание, присутствуют следующие операторы:

NEW(P1); NEW(P2); NEW(Pm);

После их выполнения в динамической памяти оказывается выделенным место под три величины (две скалярные и один массив), которые имеют идентификаторы:

P1^, P2^, Pm^

Например, обозначение P1^ можно расшифровать так: динамическая переменная, на которую ссылается указатель P1.

Дальнейшая работа с динамическими переменными происходит точно так же, как со статическими переменными соответствующих типов. Им можно присваивать значения, их можно использовать в качестве операндов в выражениях, параметров подпрограмм и пр. Например, если переменной P1^ нужно присвоить число 25, переменной P2^ присвоить значение символа "Write", а массив Pm^ заполнить по порядку целыми числами от 1 до 100, то это делается так:

P1^ := 25;

P2^ := 'Write';

For I := 1 To 100 Do Pm^[I] := I;

Кроме процедуры NEW значение указателя может определяться оператором присваивания:

<указатель> := <ссылочное выражение>;

В качестве ссылочного выражения можно использовать

указатель;

ссылочную функцию (т.е. функцию, значением которой является указатель);

константу Nil.

Nil — это зарезервированная константа, обозначающая пустую ссылку, т.е. ссылку, которая ни на что не указывает. При присваивании базовые типы указателя и ссылочного выражения должны быть одинаковы. Константу Nil можно присваивать указателю с любым базовым типом.

До присваивания значения ссылочной переменной (с помощью оператора присваивания или процедуры NEW) она является неопределенной.

Ввод и вывод указателей не допускается.

Рассмотрим пример. Пусть в программе описаны следующие указатели:

Var D, P : ^Integer;

K : ^Boolean;

Тогда допустимыми являются операторы присваивания

D := P; K := Nil;

поскольку соблюдается принцип соответствия типов. Оператор K := D ошибочен, т.к. базовые типы у правой и левой части разные.

Если динамическая величина теряет свой указатель, то она становится "мусором". В программировании под этим словом понимают информацию, которая занимает память, но уже не нужна.

Представьте себе, что в программе, в которой присутствуют описанные выше указатели, в разделе операторов записано следующее:

NEW(D); NEW(P);

{Выделено место в динамической памяти под две целые переменные. Указатели получили соответствующие значения}

D^ := 3; P^ := 5;

{Динамическим переменным присвоены значения}

P := D;

{Указатели P и D стали ссылаться на одну и ту же величину, равную 3}

WriteLn(P^, D^); {Дважды напечатается число 3}

Таким образом, динамическая величина, равная 5, потеряла свой указатель и стала недоступной. Однако место в памяти она занимает. Это и есть пример возникновения "мусора". На схеме показано, что произошло в результате выполнения оператора P := D.

В Паскале имеется стандартная процедура, позволяющая освобождать память от данных, потребность в которых отпала. Ее формат:

DISPOSE(<указатель>);

Например, если динамическая переменная P^ больше не нужна, то оператор

DISPOSE(P)

удалит ее из памяти. После этого значение указателя P становится неопределенным. Особенно существенным становится эффект экономии памяти при удалении больших массивов.

В версиях Турбо-Паскаля, работающих под операционной системой MS DOS, под данные одной программы выделяется 64 килобайта памяти (или, если быть точнее, 65520 байт). Это и есть статическая область памяти. При необходимости работать с большими массивами информации этого может оказаться мало. Размер динамической памяти — много больше (сотни килобайт). Поэтому использование динамической памяти позволяет существенно увеличить объем обрабатываемой информации.

Следует отчетливо понимать, что работа с динамическими данными замедляет выполнение программы, поскольку доступ к величине происходит в два шага: сначала ищется указатель, затем по нему — величина. Как это часто бывает, действует "закон сохранения неприятностей": выигрыш в памяти компенсируется проигрышем во времени.

Пример. Дан текстовый файл размером не более 64 Кб, содержащий действительные числа, по одному в каждой строке. Переписать содержимое файла в массив, разместив его в динамически распределяемой памяти. Вычислить среднее значение элементов массива. Очистить динамическую память. Создать целый массив размером 10000, заполнить его случайными целыми числами в диапазоне от –100 до 100 и вычислить его среднее значение.

{Язык Turbo Pascal}

Program Srednee;

Const NMax = 10000;

Type Diapazon = 1..NMax;

MasInt = Array[Diapazon] Of Integer;

MasReal = Array[Diapazon] Of Real;

Var PIint : ^MasInt; PReal : ^MasReal;

I, Midint : longInt; MidReal : Real; T : Text; S : string;

Begin

Write('Введите имя файла: '); ReadLn(S);

Assign(T, S); Reset(T); MidReal := 0; MidInt := 0;

Randomize;

NEW(PReal); {Выделение памяти под вещественный массив}

{Ввод и суммирование вещественного массива}

While Not Eof (T) Do

Begin ReadLn(T, PReal^[I]); MidReal := MidReal + PReal^[I] End;

DISPOSE(PReal); {Удаление вещественного массива}

NEW(PInt); {Выделение памяти под целый массив}

{Вычисление и суммирование целого массива}

For I := 1 To NMax Do

Begin PInt^[I] := -100 + Random(201); MidInt := MidInt + PInt^[I] End;

{Вывод средних значений}

WriteLn('среднее целое равно: ', MidInt Div NMax);

WriteLn('среднее вещественное равно: ', (MidReal / NMax) : 10 : 6)

End.

// Язык C++

#include < stdio.h >

#include < time.h >

#include < stdlib.h >

#include < iostream.h >

#define NMax 10000

typedef int MasInt;

typedef float MasReal;

MasInt \*PInt; MasReal \*PReal;

int I, n, MidInt; float MidReal; char S[255];

FILE \*t; char \*endptr;

void main()

{ cout << "Введите имя файла: "; cin >> S;

t=fopen(S, "r");

MidReal = 0; MidInt = 0;

randomize(); I=0;

/\*Выделение памяти под вещественный массив\*/

PReal = (MasReal\*) malloc (sizeof(MasReal));

/\*Ввод и суммирование вещественного массива\*/

while (!feof(t))

{fgets(S, 255, t); // вводим из файла строку

PReal[I] = strtod(S, &endptr); // преобразуем введенную строку в вещественное число

MidReal += PReal[I]; I++;}

n=I+1;

free (PReal); /\*Удаление вещественного массива\*/

PInt = (MasInt\*) malloc(sizeof(MasInt)); /\*Выделение памяти под целый массив\*/

/\* Вычисление и суммирование целого массива \*/

for (I=0; I < NMax; I++)

{ PInt[I] = -100 + random(201);

MidInt += PInt[I];}

/\*Вывод средних значений\*/

cout << "\nсреднее целое равно " << MidInt / double(NMax) << "\n";

cout << "среднее вещественное равно: " << MidReal / n << "\n";

fclose(t);

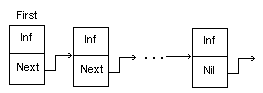
}

Списки

Обсудим вопрос о том, как в динамической памяти можно создать структуру данных переменного размера.

Разберем следующий пример. В процессе физического эксперимента многократно снимаются показания прибора (допустим, термометра) и записываются в компьютерную память для дальнейшей обработки. Заранее неизвестно, сколько будет произведено измерений.

Если для обработки таких данных не использовать внешнюю память (файлы), то разумно расположить их в динамической памяти. Во-первых, динамическая память позволяет хранить больший объем информации, чем статическая. А во-вторых, в динамической памяти эти числа можно организовать в связанный список, который не требует предварительного указания количества чисел, подобно массиву. Что же такое "связанный список"? Схематически он выглядит так:



Здесь Inf — информационная часть звена списка (величина любого простого или структурированного типа, кроме файлового), Next — указатель на следующее звено списка; First — указатель на заглавное звено списка.

Согласно определению, список располагается в динамически распределяемой памяти, в статической памяти хранится лишь указатель на заглавное звено. Структура, в отличие от массива, является действительно динамической: звенья создаются и удаляются по мере необходимости, в процессе выполнения программы.

Для объявления списка сделано исключение: указатель на звено списка объявляется раньше, чем само звено. В общем виде объявление выглядит так.

Type U = ^Zveno;

Zveno = Record Inf : BT; Next: U End;

Здесь BT — некоторый базовый тип элементов списка.

Если указатель ссылается только на следующее звено списка (как показано на рисунке и в объявленной выше структуре), то такой список называют **однонаправленным**, если на следующее и предыдущее звенья — **двунаправленным списком**. Если указатель в последнем звене установлен не в Nil, а ссылается на заглавное звено списка, то такой список называется **кольцевым**. Кольцевыми могут быть и однонаправленные, и двунаправленные списки.

Более подробно рассмотрим работу со связанными списками на примере однонаправленного некольцевого списка.

Выделим типовые операции над списками:

добавление звена в начало списка;

удаление звена из начала списка;

добавление звена в произвольное место списка, отличное от начала (например, после звена, указатель на которое задан);

удаление звена из произвольного места списка, отличного от начала (например, после звена, указатель на которое задан);

проверка, пуст ли список;

очистка списка;

печать списка.

Реализуем выделенный набор операций в виде модуля. Подключив этот модуль, можно решить большинство типовых задач на обработку списка. Пусть список объявлен так, как было описано выше. Первые четыре действия сначала реализуем отдельно, снабдив их иллюстрациями.

1. Добавление звена в начало списка

|  |  |
| --- | --- |
|  | {Процедура добавления звена в начало списка; в x содержится добавляемая информация}  Procedure V\_Nachalo(Var First : U; X : BT);  Var Vsp : U;  Begin  New(Vsp);  Vsp^.Inf := X;  Vsp^.Next := First; {То звено, что было заглавным, становится вторым по счёту}  First := Vsp; {Новое звено становится заглавным}  End; |

2. Удаление звена из начала списка

|  |  |
| --- | --- |
|  | {Процедура удаления звена из начала списка;  в x содержится информация из удалённого звена}  Procedure Iz\_Nachala(Var First : U; Var X : BT);  Var Vsp : U;  Begin  Vsp := First; {Забираем ссылку на текущее заглавное звено}  First := First^.Next; {То звено, что было вторым по счёту, становится заглавным}  X := Vsp^.Inf; {Забираем информацию из удаляемого звена}  Dispose(Vsp); {Уничтожаем звено}  End; |

3. Добавление звена в произвольное место списка, отличное от начала (после звена, указатель на которое задан)

|  |  |
| --- | --- |
|  | {Процедура добавления звена в список после звена,  на которое ссылается указатель Pred;  в x содержится информация для добавления}  Procedure V\_Spisok(Pred : U; X : BT);  Var Vsp : U;  Begin  New(Vsp); {Создаем пустое звено}  Vsp^.Inf := X; {Заносим информацию}  Vsp^.Next := Pred^.Next; {Теперь это звено ссылается на то,  что было следом за звеном Pred}  Pred^.Next := Vsp; {Теперь новое звено встало вслед за звеном Pred}  End; |

4. Удаление звена из произвольного места списка, отличного от начала (после звена, указатель на которое задан)

|  |  |
| --- | --- |
|  | {Процедура удаления звена из списка после звена,  на которое ссылается указатель Pred;  в x содержится информация из удалённого звена}  Procedure Iz\_Spiska(Pred : U; Var X : BT);  Var Vsp : U;  Begin  Vsp := Pred^.Next; {Забираем ссылку на удаляемое звено}  {Удаляем звено из списка, перенаправив ссылку на следующее  за ним звено}  Pred^.Next := Pred^.Next^.Next;  X := Vsp^.Inf; {Забираем информацию из удаляемого звена}  Dispose(Vsp); {Уничтожаем звено}  End; |

Приведём полный текст модуля.

|  |  |  |
| --- | --- | --- |
| {Язык Pascal}  Unit Spisok;  Interface  Type BT = LongInt;  U = ^Zveno;  Zveno = Record Inf : BT; Next: U End;  Procedure V\_Nachalo(Var First : U; X : BT);  Procedure Iz\_Nachala(Var First : U; Var X : BT);  Procedure V\_Spisok(Pred : U; X : BT);  Procedure Iz\_Spiska(Pred : U; Var X : BT);  Procedure Ochistka(Var First: U);  Function Pust(First : U) : Boolean;  Procedure Print(First : U);  Implementation  Procedure V\_Nachalo;  Var Vsp : U;  Begin  New(Vsp);  Vsp^.Inf := X;  Vsp^.Next := First;  First := Vsp;  End;  Procedure Iz\_Nachala;  Var Vsp : U;  Begin  Vsp := First;  First := First^.Next;  X := Vsp^.Inf;  Dispose(Vsp);  End;  Procedure V\_Spisok;  Var Vsp : U;  Begin  New(Vsp);  Vsp^.Inf := X;  Vsp^.Next := Pred^.Next;  Pred^.Next := Vsp;  End;  Procedure Iz\_Spiska;  Var Vsp : U;  Begin  Vsp := Pred^.Next;  Pred^.Next := Pred^.Next^.Next;  X := Vsp^.Inf;  Dispose(Vsp);  End;  Procedure Ochistka;  Var Vsp : BT;  Begin  While Not Pust(First) Do Iz\_Nachala(First, Vsp)  End;  Function Pust;  Begin  Pust := First = Nil  End;  Procedure Print;  Var Vsp : U;  Begin  Vsp := First;  While Vsp <> Nil Do  Begin  Write(Vsp^.Inf : 6);  Vsp := Vsp^.Next  End; WriteLn  End;  Begin  End. |  | // Язык С++  #include < iostream.h >  #include < conio.h >  #include < stdlib.h >  #include < time.h >  typedef long BT;  struct Zveno{  BT Inf;  Zveno \*Next; };  Zveno \*V\_Nachalo(Zveno \*First, BT X)  { Zveno \*Vsp;  Vsp = (Zveno \*) malloc(sizeof(Zveno));  Vsp->Inf=X; Vsp->Next=First; First=Vsp;  return First;  }  Zveno \*Iz\_Nachala(Zveno \*First)  { Zveno \*Vsp;  Vsp=First->Next;  free(First);  return Vsp;  }  Zveno \*V\_Spisok(Zveno \*Pred, BT X)  { Zveno \*Vsp;  Vsp = (Zveno \*) malloc(sizeof(Zveno));  Vsp->Inf=X;  Vsp->Next=Pred->Next;  Pred->Next=Vsp;  return Vsp;  }  BT Iz\_Spiska(Zveno \*Pred)  { BT X;  Zveno \*Vsp;  Vsp=Pred->Next;  Pred->Next=Pred->Next->Next;  X=Vsp->Inf;  free(Vsp);  return X;  }  void Print(Zveno \*First)  { Zveno \*Vsp;  Vsp=First;  while (Vsp)  {cout << Vsp->Inf << ' '; Vsp=Vsp->Next;}  cout << "\n";  }  int Pust(Zveno \*First)  {  return !First;  }  Zveno \*Ochistka(Zveno \*First)  {  while (!Pust(First)) First=Iz\_Nachala(First);  return First;  } |

Пример. Составить программу, которая на основе заданного списка формирует два других, помещая в первый из них положительные, а во второй — отрицательные элементы исходного списка.

При реализации алгоритма будем использовать подпрограммы разработанного модуля. Это существенно облегчает решение задачи.

{Программа на Turbo Pascal}

Program Ex\_sp\_1;

Uses Spisok;

Var S1, S2, S3, V1, V2, V3 : U; A : BT; I, N : Byte;

Begin

Randomize;

N := 1 + Random(20);

S1 := Nil; A := -100 + Random(201);

V\_Nachalo(S1, A); V1 := S1;

For I := 2 To N Do

Begin A := -100 + Random(201); V\_Spisok(V1, A); V1 := V1^.Next End;

WriteLn('Исходный список: '); Print(S1);

V1 := s1; S2 := Nil; S3 := Nil;

While V1 <> Nil Do

Begin

If V1^.Inf > 0

Then If S2 = Nil

Then Begin V\_Nachalo(S2, V1^.Inf); V2 := S2 End

Else Begin V\_Spisok(V2, V1^.Inf); V2 := V2^.Next End;

If V1^.Inf < 0

Then If S3 = Nil

Then Begin V\_Nachalo(s3, V1^.Inf); V3 := S3 End

Else Begin V\_Spisok(V3, V1^.Inf); V3 := V3^.Next End;

V1:= V1^.Next

End;

WriteLn('Результирующий список из положительных элементов: '); Print(S2);

WriteLn('Результирующий список из отрицательных элементов: '); Print(S3);

Ochistka(S1); Ochistka(S2); Ochistka(S3);

End.

// Программа на C++

#include "SPIS.CPP"

void main()

{Zveno \*S1, \*S2, \*S3, \*V1, \*V2, \*V3;

BT a; int i, n;

clrscr();

randomize();

S1=NULL;

// создаём первый элемент

a=-100+random(201);

S1=V\_Nachalo(S1, a);

n=1+random(20);

// формируем список произвольной длины и выводим на печать

V1=S1;

for (i=2; i<=n; i++)

{

a=-100+random(201);

V1=V\_Spisok(V1, a);

}

Print(S1);

V1 = S1; S2 = NULL; S3 = NULL;

while (V1)

{if (V1->Inf > 0)

if (!S2)

{S2=V\_Nachalo(S2, V1->Inf); V2 = S2;}

else {V\_Spisok(V2, V1->Inf); V2 = V2->Next;};

if (V1->Inf < 0)

if (!S3)

{S3=V\_Nachalo(S3, V1->Inf); V3 = S3;}

else {V\_Spisok(V3, V1->Inf); V3 = V3->Next;};

V1= V1->Next;}

cout << "Результирующий список из положительных элементов: \n";

Print(S2);

cout << "Результирующий список из отрицательных элементов: \n";

Print(S3);

S1=Ochistka(S1); S2=Ochistka(S2); S3=Ochistka(S3);

}

Контрольные вопросы и задания

Чем отличаются статические и динамические величины?

Какая память называется динамически распределяемой?

Что такое указатель?

Какие виды указателей вам известны?

Как определяется адрес переменной?

Приведите примеры объявления указателей.

Как выделить память под динамическую переменную? Как освободить память от динамической переменной?

Что такое "разыменование"?

Что в языке Pascal обозначает константа Nil (в языке C константа NULL)?

В каком случае возможно присваивание указателей?

Какие ситуации приводят к возникновению в динамически распределяемой памяти "мусора"?

Что понимают под "связанным списком"?

Как классифицируют связанные списки?

Какие основные действия над списками и компонентами списков обычно реализуют?

Как описывается список?

Двунаправленный список объявлен следующим образом:

Type BT = Byte;

U = ^Zveno;

Zveno = Record Inf : BT; Pred, Next: U End;

Здесь Pred, Next — соответственно указатели на предыдущее и последующее звенья списка. Разработать основные подпрограммы для обслуживания такого списка.