Лабораторная работа №4.

ДИНАМИЧЕСКИЕ ПЕРЕМЕННЫЕ.

1. Цели работы

– приобретение практических навыков разработки программ, использующих указатели и динамические переменные;

– знакомство с типовыми ошибками и сообщениями компилятора.

2. Особенности динамических переменных

Основной способ определения как простых, так и структурированных данных в программах на языке C++ заключается в определении некоторого типа данных TypeVar и объявлении переменных этого типа с помощью оператора объявления:

```
TypeVar    X1, X2, X3;
```

Объявления переменных обрабатываются компилятором, который в соответствии с типом каждой переменной отводит ей область памяти, достаточную для хранения данных типа `TypeVar`. Объекты, задаваемые описанным способом, называют *статическими* переменными. Такие переменные обладают следующими особенностями:

а) память для размещения статических переменных выделяет компилятор;

б) начинают существовать в программе с момента ввода ее в память ЭВМ, являются доступными в любой момент выполнения программы (подпрограммы) и перестают существовать после завершения работы программы;

в) используются в выражениях и операторах языка C++ путем упоминания имени;

г) не допускают изменения числа компонент структурированных переменных во время выполнения программы.

Однако в некоторых задачах, например, моделирование и автоматизация проектирования, часто приходится иметь дело с данными, число компонент которых заранее не известно и может изменяться в широком диапазоне. Для того, чтобы облегчить программирование обработки таких данных, в языке C++ имеются средства для создания и уничтожения переменных в ходе выполнения программ. Такие переменные называются *динамическими*.

Динамическая переменная не имеет фиксированного имени, доступ к ней осуществляется с помощью адреса ее расположения в физической памяти машины, который хранится в статической переменной, называемой *указателем.* Каждый указатель должен быть описан и, более того, в программе должен быть определен тип объекта, на который он ссылается. Однако, в отличие от статических переменных, которые во время выполнения программы связаны с фиксированной областью памяти, динамическая переменная может находиться в любом месте памяти, так как указатель, как и любая статическая переменная, может иметь произвольное значение адреса в разные моменты выполнения программы. Основные отличия динамических переменных от статических заключаются в следующем:

а) память для размещения переменной выделяется в динамической памяти (или в “куче” (Heap)) при выполнении программы (подпрограммы);

б) эти переменные могут существовать в течение некоторой части времени выполнения программы;

в) используются в выражениях и операторах языка C++ путем упоминания указателя, ссылающегося на переменную динамического вида;

г) допускают изменение числа компонент структурированных данных во время выполнения программы.

3. Определение типов указателей

Одной из особенностей динамических данных является то, что их компоненты не именуются, а определяются специальными переменными, называемыми указателями. Эти переменные

должны быть описаны в программе, использующей динамические данные, и для каждой такой переменной должен быть определен тип компонентов, на которые она может ссылаться.

Чтобы отличить указатели от других переменных, используемых в программе, в языке C++ введен специальный тип, который отличается от других стандартных типов тем, что для его обозначения используется не идентификатор, а специальный знак: `*`. Этот знак используется перед именем переменной, являющейся указателем, а перед символом * должен быть указан тип данных, на который может ссылаться указатель. Например, описание

```
typdef
   int *PItem;
```

говорит о том, что имя PItem закрепляется за типом указателя, который может ссылаться на данные типа int.

А описание

```
struct
   TItem {
         char Data;
         TItem * Next;
   };
```

определяет структуру, состоящую из двух полей Data и Next, второе из которых также является указателем на компонент типа TItem, т.е. на такую же структуру.

4. Создание динамических переменных

После описания типов указателей в программе можно определить имена переменных, называемых указателями, в виде обычного описания, например,

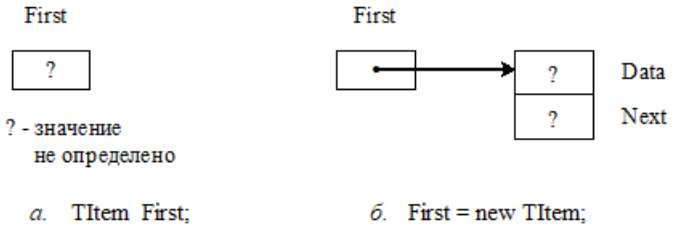
```
TItem *First;
```

где First – указатель типа TItem. При выполнении подобного оператора объявления, указателю (переменной First) транслятор отводит область памяти, достаточную для хранения адреса динамических данных. Однако значение самого указателя ещё не определено, т.к. компоненты данных, на которые он ссылается, ещё не построены.

Компоненты динамических данных строятся на этапе выполнения программы с помощью операции **`new TItem`**. Эта операция отводит память в динамической области, необходимую для размещения одного компонента данных типа `TItem`, и присваивает указателю адрес месторасположения построенного компонента в памяти. Если запрос на выделение памяти прошёл не удачно, то результатом выполнения оператора new будет нулевой указатель **`nullptr`**. Построенный описанным образом компонент данных может быть использован затем в операторах и выражениях программы.

Рассмотренную ситуацию можно пояснить с помощью схемы, изображенной на рис. 1.

На рис. 1а показана в виде прямоугольника область памяти, связанная с указателем `First`, значение которого пока ещё не определено. Схема, иллюстрирующая ситуацию после выполнения оператора `new TItem`, приведена на рис. 1б. Теперь указатель хранит адрес (число) динамической области, с которой он связан, и стрелка, исходящая из указателя `First`, «показывает» на какой элемент он ссылается.



а. TItem First; *б.* First = new TItem;

Рис. 1. Выделение памяти

Оператор `new` во время выполнения программы позволяет увеличивать размер памяти, используемый ею. При этом его многократное применение может привести к ситуации, когда все резервы памяти ЭВМ окажутся исчерпанными. Чтобы обеспечить рациональное использование основной памяти машины, в языке С++ предусмотрен оператор **`delete First`**, который производит действия, противоположные действиям оператора `new`. Он освобождает области памяти, определяемой указателем `First`. После его выполнения значение этого указателя становится неопределённым.

5. Операции над указателями и динамическими переменными

Переменные типа указатель в языке С++ разрешается использовать не только в операторах присваивания, но и в арифметических операциях, и операциях сравнения. Причем такой переменной может быть присвоено значение `nullptr`, которое используется для обозначения указателя, не ссылающегося (не имеющего ссылку) на динамическую переменную. Проще говоря, значение `nullptr` означает неинициализированный указатель.

*Замечание. В С++ указатель со значением 0 называется null-указателем (нулевым указателем). С++ гарантирует, что нулевой указатель никогда не указывает на допустимые данные, поэтому он часто используется в качестве признака неудачного завершения операций или функций, которые в противном случае должны возвращать корректные указатели. Иногда программисты употребляют конструкцию (void *) 0, чтобы подчеркнуть, что это именно указатель, а не числовое значение.*

Аналогом нулевого указателя в С является макрос NULL.

Такое многообразие вариантов представления указателя, который никуда не указывает, усложняет понимание кода и читателям, и компиляторам. В С++11 введено лучшее решение – ключевое слово nullptr, которое означает нулевой указатель:

str = ***nullptr****; // нотация нулевого указателя в С++11*

Используемое значение указателя можно определить либо с помощью оператора new, либо с помощью присваивания значения другого указателя. Возможность присваивания значений указателю позволяет использовать его для ссылок на различные компоненты данных во время выполнения программы.

В отличие от указателей переменные, на которые они ссылаются, могут быть использованы в любых конструкциях языка С++ аналогично другим статическим переменным. Для доступа к значению, на которое ссылается указатель (значению динамической переменной), в С++ используется специальный оператор *разыменовывания* «*» в виде: `* указатель`. Значением подобного выражения будет не адрес, хранящийся в указателе, а значение, хранящееся по этому адресу.

На рис. 2 приведена графическая схема (состояние статической и динамической областей памяти), поясняющая выполнение операции разыменовывания, присваивания указателю и динамической переменной. На схеме используются указатели `A` и `B`, динамические переменные `*A` и `*B` типа `int`:

```
int *A, *B;     // рис. 2а
A = new int, B = new int;  // рис. 2б
*A=13;  *B=75; // рис. 2в
*A=*B;         // рис. 2г
 A=B;          // рис. 2д
```

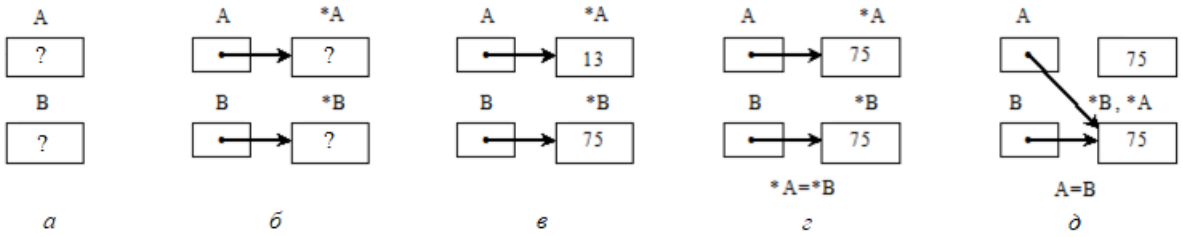


Рис. 2. Присваивание значения динамической переменной (г) и указателю (д)

Начальные значения динамических переменных указаны на рис. 2в. Если выполняется присваивание динамической переменной `*A`, которое поясняется схемой на рис. 2г, то просто изменяется значение компонента, на который ссылается указатель `A`, сам компонент продолжает существовать и может быть использован при дальнейшем выполнении программы.

Но если выполнить присваивание значения указателю `A`, то возникает ситуация, показанная на рис. 2д, при котором один компонент, хранящий значение 75, оказывается потерянным и не может быть использован в дальнейшем. Память (динамическая) будет занята, но будет недоступна программе! Такое состояние называется «утечкой» памяти и является грубой логической ошибкой.

Переменные динамического типа могут быть как простого, так и сложного структурированного типа. Во втором случае обозначение указателя динамической переменной должно быть использовано в качестве части составного имени компонентов рассматриваемой переменной. Например, указатель `P` ссылается на компонент, представляющий собой запись с тремя полями `A`, `B` и `C`, то для обозначения второго поля записи следует использовать составное имя `(*P).B`.

6. Пример программы с динамическими переменными

Рассмотрим задачу построения формулы "**A*A**" из символов '*****' и '**A**'. Решение этой задачи с использованием динамических переменных может быть представлено в следующем виде:

```
// example24
#include <iostream>
using namespace std;
typedef char * P;
P S1, S2, S3;

int main()
{
   S1 = new char;
   S2 = new char;
   cout << " Введите два символа [*A]: ";
   cin >> *S1 >> *S2; // *S1='*', *S2='A'
   S3 = S2; // S3 указывает на 'A'
   S2 = S1; // S2 указывает на '*'
   S1 = S3; // S1 указывает на 'A', источник – S3,
            // так как S2 уже изменен и указывает на '*'!
   cout << " Результат = " << *S1 << *S2 << *S3;
   delete S2; // *
   delete S3; // A, или delete S1
   return 0;
}
```

В этой программе предполагается, что после ввода данных указатель `S1` ссылается на символ `'*'`, а указатель `S2` – на символ `'A'`. Новое значение указателям присваивается, начиная с последнего в строке объявления переменных. Перед выводом данных два указателя `S3` и `S1` ссылаются на одну и туже динамическую переменную, содержащую символ `'A'`. При анализе

этой программы необходимо изобразить указатели, динамические данные и их значения в виде схемы по аналогии с рис. 2, добавляя изменения в схеме после каждого оператора присваивания.

7. Рекомендации по работе с указателями

При работе с динамическими переменными следует внимательно следить за тем, чтобы значения указателей в программе были определены перед их использованием. Например, если значение указателя `R` не было определено с помощью процедуры `new` или присваивания ему значения другого указателя, либо равно `nullptr`, то при компиляции эта логическая ошибка не обнаружится, поскольку в языке C++ допустимы любые значения указателя. Такая ошибка может проявиться при выполнении программы.

Наиболее трудным моментом при работе с динамическими данными является контроль и обнаружение ошибок при обращении к переменным с помощью указателя. Подобные ошибки не выявляются транслятором, поскольку значения указателей определяются во время выполнения программы. Например, если переменные `P1` и `P2` описаны в программе как указатели одного типа, то транслятор считает правильным оператор присваивания вида: `*P1=*P2`, не учитывая того, что значения указателей могут быть не определены. Пусть в исходной программе пропущен оператор, определяющий значение указателя `P2`, например, оператор `new P2`. В этом случае трансляция будет успешной, а использование указателя с неопределённым значением может привести к непредсказуемым результатам и аварийному останову программы по причине нарушения защиты памяти во время её выполнения, причём не обязательно при обращении к переменной `*P1`.

8. Индивидуальные задания

Составить программу построения и вывода на терминал формулы из заданной последовательности символов длиной K. Количество указателей для отображения формулы определяется ее длиной, причем первые K указателей должны быть использованы для ввода последовательности символов, присутствующих в формуле. В процессе построения формулы все её символы должны храниться в одном экземпляре. При выводе формулы порядок следования указателей в операторе вывода должен соответствовать порядку их объявления. Последовательности символов и формулы, которые строятся из этих последовательностей, приведены в табл.1. Предусмотреть ввод необходимых символов с клавиатуры, а также их контрольный вывод. Согласно принципам дружественного интерфейса предусмотреть вывод итоговой формулы.

9. Подготовка к работе

1. Изучить описание работы.
2. Разобраться со способами описания и работы с динамическими переменными, назначением операторов `new` и `delete`.
3. Проанализировать работу примера `example24`. По примеру рис. 2 привести графическую схему выполнение операций присваивания указателей на каждом шаге.
4. Изобразить итоговую структуру данных своего задания (значения указателей и динамических переменных, используемых для вывода формулы) (табл.1).
5. Составить программу, реализующую построение структуры данных, составленной в п.4.
6. Подготовить в письменной форме заготовку отчета по работе.

10. Порядок выполнения работы

1. Выполнить подготовку к работе.
2. Ввести, оттранслировать и выполнить программу для заданного варианта задания. Выполнить программу для другой осмысленной последовательности входных символов, приводящей к построению синтаксически верной с точки зрения C++ формулы.
3. Добавить в программу освобождение памяти, занятой динамическими переменными.

Таблица 1

Номер	Последовательность символов	Формула
1	A,B,C,+,*,(,)	A*(C+B)*C+A
2	A,B,C,D,+,),(	(A+B)+(C+D)+C
3	X,Y,*,-,(,)	(X*X-Y*Y)*X-Y
4	K,1,-,+,*,(,)	(K-1)*(K+1)*11
5	M,N,2,*,+	M*M+2*M*N+N*N
6	Y,X,0,1,.,*,+	0.011+X*X+Y
7	X,Y,+,-,/,(,)	(X+Y)/(X-Y)
8	X,Y,Z,-,*,/	-X*Y/Z*X/Y*Z
9	X,1,2,!,+,*,/	12+!X+X*X/2
10	A,B,C,P,-,*,(,)	(P-A)*(P-B)*(P-C)
11	A,D,|,&,~,(,)	(A|D)&&(~A|D)
12	A,B,C,D,~,&,^	~A^B&^C&&^D
13	X,Y,Z,~,|,(,)	~(~(~Z|Y)|X)
14	X,Y,Z,~,&,(,),|	(~X&Y)|Z&X&Y
15	A,B,C,|,~,&	~~A&~B|~C&A
16	1,0,b,L,&	0b110L&0b010
17	1,0,b,^,~,(,),&	0x11&(0b10^~0b0)
18	A,B,C,&,|,^	A&B&C|A^C^B
19	1,0,b,(,),&,^,~	(0b111&~0b001)^0b1
20	X,Y,Z,|,&,~,(,)	(~X|Z|Y)&(X|Y)
21	X,Y,>,<,=,&,-,(,)	(Y<=X)&&(Y>=-X)
22	X,Y,0,1,>,<,&	X>0&&Y>0&&X<1
23	A,B,C,*,+,1	A*B*C+A*B+1
24	A,B,|,&,~	A|~B&~A|B
25	X,Y,|,&,~	X&Y|~X&~Y

11. Содержание отчета

1. Индивидуальное задание.
2. Материалы в соответствии с подготовкой к работе (п. 3 и 4 «Подготовки к работе»).
3. Произольная осмысленная последовательность входных символов для проверки работоспособности программы и результаты её работы.

12. Пример программы

Задание. Имея девять указателей на символы, ввести последовательность символов `AB!&^` и разместить их в динамической памяти с помощью первых пяти указателей. Затем изменить значения указателей так, чтобы при выводе содержимого динамической памяти через указатели в последовательности их объявления получить на экране формулу `A!^B&^A!B`. Предусмотреть ввод необходимых символов с клавиатуры, а также их контрольный вывод.

```
// test24;
int main()
{
  char *P1, *P2, *P3, *P4, *P5, *P6, *P7, *P8, *P9;
  cout << "\n Получаем формулу: A!^B&^A!B ";
  cout << "\n Введите 5 символов: AB!&^  ";
  // Выделяем память для размещения символов
  P1 = new char;
  P2 = new char;
  P3 = new char;
  P4 = new char;
  P5 = new char;
  // Помещаем символы в динамическую память
  cin >> *P1 >> *P2 >> *P3 >> *P4 >> *P5; // AB!&^
  // Контрольный вывод
  cout << "\nВы ввели: " << *P1 << *P2 << *P3 << *P4 << *P5;
  // Меняем значения указателей, начиная с последнего
  P9 = P2; // B
  P8 = P3; // !
  P7 = P1; // A
  P6 = P5; // ^
  P5 = P4; // &
  P4 = P9; // B
  P3 = P6; // ^
  P2 = P8; // !
  // P1 сохраняет свое значение A
  // Выводим полученную формулу с помощью указателей
  // в порядке их объявления
  cout <<"\n" << *P1 << *P2 << *P3 << *P4
       << *P5 << *P6 << *P7 << *P8 << *P9;
  // освобождаем динамическую память
  delete P7; // A
  delete P9; // B
  delete P8; // !
  delete P5; // &
  delete P6; // ^
  return 0;
}
```