Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

Лабораторная работа 1

По дисциплине «Структуры данных»

На тему «Перечисление. Объединение. STL контейнеры.

Массив. Дек. Одно/двух связный список. Вектор»

Выполнила:

Студентка 1 курса 2 группы

Глухова Д.В.

Преподаватель: Север А.С.

2023, Минск

**Перечисление** (или ***«перечисляемый тип»***) — это тип данных, где любое значение (или ***«перечислитель»***) определяется как символьная константа. Объявить перечисление можно с помощью **ключевого слова** **enum**. объявление перечислений не требует выделения памяти. Только когда переменная перечисляемого типа определена, только тогда выделяется память для этой переменной.

Каждый перечислитель отделяется запятой, а само перечисление заканчивается точкой с запятой. Каждой константе по умолчанию будет присваиваться числовое значение начиная с нуля. Перечисления - это отдельный тип, поэтому мы не можем присвоить переменной напрямую числовое значение. Если нас не устраивают значения по умолчанию для констант, то мы можем явным образом задать значения. Перечисления могут использоваться, когда у нас есть ряд логически связанных констант, которые естественно лучше определить в одном общем типе данных.

Пример:

|  |
| --- |
| // 3. Написать программу, которая, в зависимости от выбора пользователя, в консоли должна выводить название планеты Солнечной системы, с использованием перечислений.  #include <iostream>  using namespace std;  enum planets {  mercury = 1,  venus,  earth,  mars,  jupiter,  saturn,  uran  };  int main()  {  setlocale(LC\_ALL, "rus");  int number;  cout << "какая по расположению планета от солнца?(1 - меркурий, 2 - венера, 3 - земля, 4 - марс, 5 - юпитер, 6 - сатурн, 7 - уран)";  cin >> number;  switch (number)  {  case planets::mercury:  cout << "меркурий";  break;  case planets::venus:  cout << "венера";  break;  case planets::earth:  cout << "земля";  break;  case planets::mars:  cout << "марс";  break;  case planets::jupiter:  cout << "юпитер";  break;  case planets::saturn:  cout << "сатурн";  break;  case planets::uran:  cout << "уран";  break;  }  } |

**Контейнерный класс** (***«класс-контейнер»***) — [**класс**](https://ravesli.com/urok-113-klassy-obekty-i-metody-klassov/) для хранения и организации объектов определенного типа данных (пользовательских или фундаментальных). Наиболее часто используемым контейнером является [**массив**](https://ravesli.com/urok-74-massivy-chast-1/). Большинство программистов используют контейнерные классы массивы: **[std::array](https://ravesli.com/urok-94-vvedenie-v-std-array/" \t "_blank)** или **[std::vector](https://ravesli.com/urok-95-vvedenie-v-std-vector-vektory/" \t "_blank)** из-за их преимуществ. Контейнерные классы-массивы имеют возможность динамического изменения своего размера, когда элементы добавляются или удаляются. Это не только делает их более удобными, чем обычные массивы, но и безопаснее.

**Массив** — структура данных, хранящая набор значений (элементов массива), идентифицируемых по индексу или набору индексов, принимающих целые (или приводимые к целым) значения из некоторого заданного непрерывного диапазона. **std::array** — фиксированный массив, который не распадается в указатель при передаче в функцию. Он определяется в **заголовочном файле** array, внутри **пространства имен** std. Нельзя пропустить (не указывать) длину такого массива. Массив объект, управляющий последовательностью из элементов N типа Ty. Последовательность хранится как массив Ty в объекте array<Ty, N>. У этого типа есть конструктор по умолчанию array() и оператор присваивания по умолчанию operator=. Тип удовлетворяет требованиям для aggregate. Поэтому объекты типа array<Ty, N> можно инициализировать с помощью агрегатного инициализатора. Например, примененная к объекту директива.

Пример:

|  |
| --- |
| // 9. Написать программу, в которой нужно найти сумму минимального и максимального элементов массива, с выводом на экран.  #include <iostream>  #include <array>  using namespace std;  int main()  {  setlocale(LC\_ALL, "rus");  array<int, 10> arr = { };  int max = arr[0], min = arr[0];  for (int i = 0; i < 10; i++)  {  arr[i] = rand() % 20 - 5;  if (arr[i] > max) max = arr[i];  if (arr[i] < min) min = arr[i];  cout << arr[i] << " ";  }  cout << endl << "max = " << max << endl;  cout << "min = " << min;  } |

**Класс vector**— [**динамический массив**](https://ravesli.com/urok-86-dinamicheskie-massivy/), способный увеличиваться по мере необходимости для содержания всех своих элементов. Обеспечивает произвольный доступ к своим элементам через **оператор индексации** [ ], поддерживает добавление и удаление элементов. Позволяет избежать утечек памяти и облегчает работу программисту.

* pop\_back() — удалить последний элемент
* clear() — удалить все элементы вектора
* empty() — проверить вектор на пустоту

Пример:

|  |
| --- |
| // 7. Написать программу, в которой нужно найти максимальный элемент вектора и вывести на экран.  #include <iostream>  #include <vector>  #include <algorithm>  using namespace std;  int main()  {  srand((unsigned)time(NULL));  vector<int> vect;  for (int i = 0; i < 10; i++)  vect.push\_back(rand() % 20 - 5); // заполняем вектор рандомными числами  vector<int>::iterator beg = vect.begin(); // начало вектора  vector<int>::iterator end = vect.end(); // конец вектора  while (beg != end)  cout << \*(beg++) << ' ';  int max = \*max\_element(vect.begin(), vect.end()); // ищем минимальный элемент от начала до конца  int min = \*min\_element(vect.begin(), vect.end()); // ищем максимальный элемент от начала до конца  cout << endl << "min = " << min << endl;  cout << "max = " << max << endl;    return 0;  } |
|  |
| // 10. Написать программу, в которой нужно найти сумму всех элементов вектора и вывести ее на экран.  #include <iostream>  #include <vector>  #include <algorithm>  using namespace std;  int main()  {  srand((unsigned)time(NULL));  vector<int> vect;  for (int i = 0; i < 10; i++)  {  vect.push\_back(rand() % 20 - 5); // заполняем вектор рандомными числами  cout << vect.at(i) << " ";  }  int sum = 0;  for (int i = 0; i < 10; i++)  {  sum += vect.at(i);    }  cout << endl << "sum = " << sum;  return 0;  } |

**Класс deque** (или просто **«дек»**) — это двусторонняя очередь, реализованная в виде динамического массива, который может расти с обоих концов. Для использования данного контейнера нужно подключить заголовочный файл deque.

Для получения элементов очереди можно использовать операцию [] и ряд функций:

* [index]: получение элемента по индексу
* at(index): возращает элемент по индексу
* front(): возвращает первый элемент
* back(): возвращает последний элемент

Если мы будем обращаться с помощью операции индексирования по некорректному индексу, который выходит за границы контейнера, то результат будет неопредленным. Чтобы узнать размер очереди, можно использовать функцию size(). А функция empty() позволяет узнать, содержит ли очередь элементы. Функция resize() позволяет изменить размер очереди. Функция assign() позволяет заменить все элементы очереди определенным набором. Функция swap() обменивает значениями две очереди.

Пример:

|  |
| --- |
| // 8. Написать программу, в которой нужно найти минимальный элемент дэки и добавить его в начало, с выводом на экран .  #include <iostream>  #include <deque>  #include <algorithm>  using namespace std;  int main()  {  deque<int> num;  setlocale(LC\_ALL, "rus");  cout << "очередь: " << endl;  for (int i = 0; i < 10; i++)  {  num.push\_back(rand() % 20 - 5);  cout << num[i] << "\t";  }  cout << endl;  int max = \*max\_element(num.begin(), num.end()); // ищем максимальный элемент от начала до конца  int min = \*min\_element(num.begin(), num.end());  cout << "max = " << max << endl;  cout << "min = " << min;  num.push\_front(min);  cout << endl << "очередь с минимальным элементом в начале: " << endl;    for (int i = 0; i < num.size(); i++)  {  cout << num[i] << "\t";  }  return 0;  } |
|  |
| // 12. Написать программу, в которой нужно найти произведение всех элементов деки и вывести его на экран.  #include <iostream>  #include <deque>  #include <algorithm>  using namespace std;  int main()  {  deque<int> num;  setlocale(LC\_ALL, "rus");  cout << "очередь: " << endl;  int p = 1;  for (int i = 0; i < 10; i++)  {  num.push\_back(rand() % 20 - 5);  cout << num[i] << "\t";  p \*= num[i];  }  cout << endl << "произведение: " << p;  return 0;  } |

Односвязный список - это динамическая структура данных, состоящая из узлов. Каждый узел будет иметь какое-то значение и указатель на следующий узел. Начало списка — это головной элемент, звенья — это узлы, а конец списка определяется посредством NULL — специального узла. У односвязного списка есть преимущество — вставка и удаление узлов осуществляется довольно легко **в любом месте списка**. Однако список нельзя индексировать в качестве массива, а структура списка ограничивает доступ к узлам по индексу. Если нужно попасть на какой-нибудь узел односвязного списка, нужно пройти весь путь последовательно, начиная от головного элемента, заканчивая нужным узлом.

Чтобы удалить элемент из контейнера forward\_list можно использовать следующие функции:

* **clear()**: удаляет все элементы
* **pop\_front()**: удаляет первый элемент
* **erase\_after(p)**: удаляет элемент после элемента, на который указывает итератор p. Возвращает итератор на элемент после удаленного
* **erase\_after(begin, end)**: удаляет диапазон элементов, на начало и конец которого указывают соответственно итераторы begin и end. Возвращает итератор на элемент после последнего удаленного

Для добавления элементов в forward\_list применяются следующие функции:

* **push\_front(val)**: добавляет объект val в начало списка
* **emplace\_front(val)**: добавляет объект val в начало списка
* **emplace\_after(p, val)**: вставляет объект val после элемента, на который указывает итератор p. Возвращает итератор на вставленный элемент. Если p представляет итератор на позицию после конца списка, то результат неопределен.
* **insert\_after(p, val)**: вставляет объект val после элемента, на который указывает итератор p. Возвращает итератор на вставленный элемент.
* **insert\_after(p, n, val)**: вставляет n объектов val после элемента, на который указывает итератор p. Возвращает итератор на последний вставленный элемент.
  + **insert\_after(p, begin, end)**: вставляет после элемента, на который указывает итератор p, набор объектов из другого контейнера, начало и конец которого определяется итераторами begin и end. Возвращает итератор на последний вставленный элемент

Пример:

|  |
| --- |
| // 5. Написать программу, в которой в конец односвязного списка добавляется n элементов и вывести на экран.  #include <iostream>  #include <forward\_list>  using namespace std;  int main()  {  setlocale(LC\_ALL, "rus");  forward\_list<int> newlist = { 6, 2, 8, 4, 5 };  int size, num;  cout << "введите количество, которое хотите добавить = ";  cin >> size;  cout << "введите число, которое хотите добавить = ";  cin >> num;  cout << "первоначальный список: " << endl;  for (int n : newlist)  {  cout << n << "\t";  size++;  }  cout << endl << "конечный список: " << endl;  newlist.resize(size, num);  for (int n : newlist)  {  cout << n << "\t";  }  } |
|  |
| // 13. Написать программу, в которой нужно удалить все элементы односвязного списка и добавить n новых, с выводом на экран.  #include <iostream>  #include <forward\_list>  using namespace std;  int main()  {  setlocale(LC\_ALL, "rus");  srand((unsigned)time(NULL));  forward\_list<int> newlist;  int n;  for (int i = 0; i < 10; i++)  {  newlist.push\_front(rand() % 10);  }  cout << "первоначальный список: ";  for (int n : newlist)  {  cout << n << " ";  }  newlist.clear();    cout << endl << "введите n которое хотите добавить: " << endl;  cin >> n;  cout << "новый список: ";  for (int i = 0; i < n; i++)  {  newlist.push\_front(rand() % 10);  }  for (int n : newlist)  {  cout << n << " ";  }  return 0;  } |

**List** (или просто ***«список»***) — это двусвязный список, каждый элемент которого содержит 2 указателя: один указывает на следующий элемент списка, а другой — на предыдущий элемент списка. List предоставляет доступ только к началу и к концу списка — произвольный доступ запрещен. Если вы хотите найти значение где-то в середине, то вы должны начать с одного конца и перебирать каждый элемент списка до тех пор, пока не найдете то, что ищете. Преимуществом двусвязного списка является то, что добавление элементов происходит очень быстро, если вы, конечно, знаете, куда хотите добавлять. Обычно для перебора элементов двусвязного списка используются итераторы.

|  |
| --- |
| // 6. Написать программу, в которой в начало двусвязного списка добавляется n элементов и вывести на экран.  #include <iostream>  #include <list>  using namespace std;  int main()  {  setlocale(LC\_ALL, "rus");  list<int> newlist = { 6, 2, 8, 4, 5 };  int size, num;  cout << "введите количество, которое хотите добавить = ";  cin >> size;  cout << "введите число, которое хотите добавить = ";  cin >> num;  for (int i = 0; i < size; i++)  {  newlist.push\_front(num);  }  cout << endl << "конечный список: " << endl;    for (int n : newlist)  {    cout << n << "\t";  }  } |
|  |
| // 11. Написать программу, в которой удалить n элемент из двусвязного списка и добавить такой же в начало, с выводом на экран.  #include <list>  #include <iterator>  #include <iostream>  using namespace std;  int main() {  setlocale(LC\_ALL, "RUS");  list <int> ilist;  list<int>::iterator pos\_begin, i, elem;  cout << "список: " << endl;  for (int j = 0; j < 10; j++) {  ilist.push\_back(j);  }  for (i = ilist.begin(); i != ilist.end(); i++)  cout << \*i << " ";  int n, t = 1;  cout << endl << "n = ";  cin >> n;  pos\_begin = ilist.begin();  n++;    while(t != n)  {  ++pos\_begin;  ++t;    }  elem = pos\_begin;  cout << "удаленный элемент = " << \*elem << endl;  ilist.push\_front(\*elem);  ilist.erase(pos\_begin);    cout << "список с удаленным элементом и добавленным в начало: " << endl;  for (i = ilist.begin(); i != ilist.end(); i++)  cout << \*i << " ";  return 0;  } |