|  |
| --- |
| Lab 06 Methods for Making Data Structures |
|  |
| Exercise 1: Understanding the Problem of Dynamic Array (40 min) |
| A problem of dynamic array is that there is plenty of copying involved especially when inserting or removing elements. If the array is huge and the elements are objects, resizing the array will invoke much object construction and destruction even if we just insert one element to the (full) array.  Write the definition of the four missing functions in the program. The sample output of the program is shown in Sample.Output.txt. Use Point.h and Point.cpp provided. Hint: Compile using **g++ Point.cpp main.cpp -o main.exe**  #include <iostream>  #include "Point.h"  using namespace std;  ostream& operator<< (ostream& os, Point& p)  {  os << "(" << p.getX() << "," << p.getY() << ")";  return os;  }  template <typename T>  void print (T \*array, int size)  {  if (size == 0)  {  cout << "Array is empty.\n";  return;  }  for (int i = 0; i < size; i++)  cout << array[i] << " ";  cout << endl;  }  template <typename T>  void addFront (T \*&array, int& size, T& elementToInsert)  {  // Add code  }  template <typename T>  void addEnd (T \*&array, int& size, T& elementToInsert)  {  // Add code  }  template <typename T>  void removeFront (T \*&array, int& size)  {  // Add code  }  template <typename T>  void removeEnd (T \*&array, int& size)  {  // Add code  }  int main()  {  int seed = 0; // seed for automatic value of x and y.  int size = 0; // array size.  Point \*points = NULL; // dynamic array.  int choice; // user choice.  Point newPoint;  do {  print (points, size);  cout << "Choice:\n"  << "1: Add new point at the front\n"  << "2: Add new point at the end\n"  << "3: Remove point at the front\n"  << "4: Remove point at the end\n"  << "Others: Exit\n";  cin >> choice;  switch (choice) {  case 1 : newPoint = Point(++seed);  addFront (points, size, newPoint); break;  case 2 : newPoint = Point(++seed);  addEnd (points, size, newPoint); break;  case 3 : removeFront (points, size); break;  case 4 : removeEnd (points, size); break;  }  } while (choice >= 1 && choice <= 4);  print (points, size);  if (size > 0)  delete [] points;  } |
| Exercise 2: Implementation of basic operations involving linked noes (40 min) |
| Rewrite the program in Exercise 1 to use linked nodes (without using a LinkedList class). Note that a Node struct is used and the print function has been updated.  #include <iostream>  #include "Point.h"  using namespace std;  ostream& operator<< (ostream& os, Point& p)  {  os << "(" << p.getX() << "," << p.getY() << ")";  return os;  }  template <typename T>  struct Node  {  T info;  Node<T> \*next;  };  template <typename T>  void print (Node<T> \*start)  {  if (start == nullptr)  {  cout << "List is empty.\n"; // cout a meesage if the list is empty.  return;  }  Node<T> \*ptr = start;  while (ptr != nullptr) // while not at the end of list  {  cout << ptr->info << " ";  ptr = ptr->next; // move to next node.  }  cout << endl;  }  template <typename T>  void addFront(Node<T> \*&start, T& newPoint)  {  // Add code  }  template <typename T>  void addEnd(Node<T> \*&start, T& newPoint)  {  // Add code  }  template <typename T>  void removeFront(Node<T> \*&start)  {  // Add code  }  template <typename T>  void removeEnd(Node<T> \*&start)  {  // Add code  }  int main()  {  int seed = 0; // seed for automatic value of x and y.  Node<Point> \*start = nullptr;  int choice; // user choice.  Point newPoint;  do  {  print(start);  cout << "Choice:\n"  << "1: Add new point at the front\n"  << "2: Add new point at the end\n"  << "3: Remove point at the front\n"  << "4: Remove point at the end\n"  << "Others: Exit\n";  cin >> choice;  switch (choice)  {  case 1 :  newPoint = Point(++seed);  addFront(start, newPoint);  break;  case 2 :  newPoint = Point(++seed);  addEnd(start, newPoint);  break;  case 3 :  removeFront(start);  break;  case 4 :  removeEnd(start);  break;  }  } while (choice >= 1 && choice <= 4);  print(start);  } |