46 - Dynamic Arrays in C++ (std::vector)

quarta-feira, 12 de março de 2025 07:14

- Template array, the data type the container contains, is up to you to decide
- Don't need to use templates to use, just need to rovide the type
- Class called std::vector
  - Should be called array list, not vector... but ok
  - It's a set that doesn't enforce the type
- Unlike array, this can actually resize itself
  - Create the array and put element into it
  - We can start witout knowing how many elements are in there
- We usually get creating our own types
- Make a vector that allocates 10 elements, and we violate this size
  - It copy all elements taht is already there, and put in another place in memory, larged
  - But the thng is... it grows and copy things a lot
- We nees a way to grow, that is the motivation with vectors
  - When reached certain aamount of data, groww

```
671  struct vertex46
672  {
673      float x, y, z;
674  };
675
676  std::ostream& operator<<(std::ostream& stream, const vertex46& vertex)
677  {
678      stream << vertex.x << ", " << vertex.y << ", " << vertex.z;
679      return stream;
680  }
681
682  void Function46(const std::vector<vertex46>& vertice)
683  {
684      std::cout << "Always pass vectors by referece!!!!!!!! Const reference if not going to change it" << std::endl;
685  }
686
687  int main()
688  {
689
690      // vertex46 vertices46 = new vertex46[5];
691      // we can access array 0 to 4, and if we try hier, we get an error
692      // vertex46[0] ~ vertex46[4]
693      // vertex46[5] // error
694
695      // Include a vector
696      std::vector<vertex46> vertex46_2;
697      // Is more optimal to store objects than memory
698          // to peform operations i mean
699      // If store ponters, it's cheper to resize, because t coies only integers and not objects
700      vertex46_2.push_back({1, 2, 3});
701      vertex46_2.push_back({4, 5, 6});
702
703      // get the size
704      for (int i = 0; i < (int)vertex46_2.size(); i++)
705      {
706          std::cout << vertex46_2[i] << std::endl;
707      }
708
709      // avoid copy at any cost, so lets convert to const reference
710      for (const vertex46& v : vertex46_2)
711      {
712          std::cout << v << std::endl;
713      }
714
715      // clear the element
716      // vertex46_2.clear();
717      // To erase an element
718      vertex46_2.erase(vertex46_2.begin() + 1);
719
720      for (const vertex46& v : vertex46_2)
721      {
722          std::cout << v << std::endl;
```

```
719
720     for (const vertex46& v : vertex46_2)
721     {
722         std::cout << v << std::endl;
723     }
724
725     Function46(vertex46_2);
726
727
```