NEW THINK TANK

C++ Tutorial 15

Posted by Derek Banas on May 10, 2018 in C Video Tutorial | 0 comments

In this part of my C++ tutorial I will answer a bunch of questions I have received. I'll talk about allocating memory with malloc(), Smart Pointers, Regular Pointers, Deallocating Memory, unique_ptr, Difference Between /n and endl, Polymorphic Templates and a whole bunch more.

Like always the heavily commented code follows the video below. For best results print it out and take notes as you watch.

If you like videos like this consider donating $1, or simply turn off Ad Blocking software. Either helps me to continue making free tutorials.

Code From the Video

```
// ---------- C++ Tutorial 15  ----------

// ---------- SMART POINTER EXAMPLE ----------

#include <cstdlib>
#include <iostream>
#include <string>
#include <vector>
#include <ctime>
#include <numeric>
#include <cmath>
#include <sstream>
#include <iterator>
#include <memory>
#include <stdio.h>

// A Smart pointer is a class that provides the
// power of pointers, but also handles the reallocation
// of memory when it is no longer required (The pointer
// is automatically destroyed)

// typedef creates an alias for a more complex type name
typedef std::vector<int32_t> intVec;

int main()
{
    /* MALLOC EXAMPLE
    // When you define a primitive type like int or
    // float you define exactly the amount of space
    // to set aside

    // If you need to define how much space to set aside
    // you could call malloc() and tell it how much
    // space to set aside and it returns the address to
```

```
    // that memory address

    int amtToStore;
    std::cout << "How many numbers do you want to store : ";
    std::cin >> amtToStore;

    // Create an int pointer and set aside enough space
    int * pNums;

    // Cast the pointer and define how much space to set aside
    pNums = (int *) malloc(amtToStore * sizeof(int));

    // Check if memory was allocated
    if(pNums != NULL){
        int i = 0;

        // Store values
        while(i < amtToStore){
            std::cout << "Enter a Number : ";
            std::cin >> pNums[i];
            i++;
        }
    }

    std::cout << "You entered these numbers\n";
    for(int i = 0; i < amtToStore; i++){
        std::cout << pNums[i] << "\n";
    }

    // Delete the pointer
    delete pNums;
    */

    // Smart Pointer Solution
    int amtToStore;
    std::cout << "How many numbers do you want to store : ";
    std::cin >> amtToStore;

    // This memory will be automatically reallocated
    std::unique_ptr<int[]> pNums(new int[amtToStore]);

    // unique_ptr can only have one owner
    // so this throws an error
    // std::unique_ptr<int[]> pNums2 = pNums;
    // I'll cover how to do this with shared_ptr
    // in a later tutorial

    if(pNums != NULL){
        int i = 0;

        // Store values
        while(i < amtToStore){
            std::cout << "Enter a Number : ";
            std::cin >> pNums[i];
            i++;
        }
    }

    std::cout << "You entered these numbers\n";
    for(int i = 0; i < amtToStore; i++){
        std::cout << pNums[i] << "\n";
    }

    return 0;
}
```

```cpp

// ---------- END SMART POINTER EXAMPLE ----------

// ---------- POLYMORPHIC TEMPLATES ----------

#include <cstdlib>
#include <iostream>
#include <string>
#include <vector>
#include <ctime>
#include <numeric>
#include <cmath>
#include <sstream>
#include <iterator>
#include <memory>

// Here I demonstrate how to use templates
// polymorphically

// Base class all pizzas inherit along with MakePizza
// which will be overridden
class Pizza{
public:
    virtual void MakePizza() = 0;
};

// The last templates that will be called
class NYStyleCrust {
public:
  std::string AddIngredient() {
    return "Crust so Thin You can See through it\n\n";
  }
};

class DeepDishCrust {
public:
  std::string AddIngredient() {
    return "Super Awesome Chicago Deep Dish Crust\n\n";
  }
};

// End of last templates called

// The middle templates called
template <typename T>
class LotsOfMeat: public T {
public:
  std::string AddIngredient() {
    return "Lots of Random Meat, " + T::AddIngredient();
  }
};

template <typename T>
class Vegan: public T {
public:
  std::string AddIngredient() {
    return "Vegan Cheese, Veggies, " + T::AddIngredient();
  }
};

// End of middle templates called

// We inherit from Pizza as well as the initial next template
template <typename T>
class MeatNYStyle: public T, public Pizza {
```

```
public:
  void MakePizza() { std::cout << "Meat NY Style Pizza : " <<
          T::AddIngredient(); }
};

template <typename T>
class VeganDeepDish: public T, public Pizza {
public:
  void MakePizza() { std::cout << "Vegan Deep Dish : " <<
          T::AddIngredient(); }
};

int main()
{
    // unique_ptr is a smart pointer that disposes of
    // a pointer when it is no longer in use
    std::vector<std::unique_ptr<Pizza>> pizzaOrders;

    // Generate Pizza types and place them at the end of the vector
    pizzaOrders.emplace_back(new MeatNYStyle<LotsOfMeat<NYStyleCrust>>());
    pizzaOrders.emplace_back(new VeganDeepDish<Vegan<DeepDishCrust>>());

    // Call the pizzas and execute the directions
    // for making them
    for(auto &pizza: pizzaOrders){
        pizza->MakePizza();
    }

    return 0;
}

// ---------- END POLYMORPHIC TEMPLATES ----------
```

Leave a Reply

Your email address will not be published.

Comment

Name

Email

Website

Submit Comment

Search

Search

Social Networks

Facebook

YouTube
Twitter
LinkedIn
G+

Buy me a Cup of Coffee
"Donations help me to keep the site running. One dollar is greatly appreciated." - (Pay Pal Secured)

Donate

My Facebook Page

Like Share 6K people like this. Be the first of your friends.

Archives

- October 2018
- September 2018
- August 2018
- July 2018
- June 2018
- May 2018
- April 2018
- March 2018
- February 2018
- January 2018
- December 2017
- November 2017
- October 2017
- September 2017
- August 2017
- July 2017
- June 2017
- May 2017
- April 2017
- March 2017
- February 2017
- January 2017
- December 2016
- November 2016
- October 2016
- September 2016
- August 2016
- July 2016
- June 2016
- May 2016
- April 2016
- March 2016
- February 2016
- January 2016
- December 2015
- November 2015
- October 2015
- September 2015
- August 2015

- July 2015
- June 2015
- May 2015
- April 2015
- March 2015
- February 2015
- January 2015
- December 2014
- November 2014
- October 2014
- September 2014
- August 2014
- July 2014
- June 2014
- May 2014
- April 2014
- March 2014
- February 2014
- January 2014
- December 2013
- November 2013
- October 2013
- September 2013
- August 2013
- July 2013
- June 2013
- May 2013
- April 2013
- March 2013
- February 2013
- January 2013
- December 2012
- November 2012
- October 2012
- September 2012
- August 2012
- July 2012
- June 2012
- May 2012
- April 2012
- March 2012
- February 2012
- January 2012
- December 2011
- November 2011
- October 2011
- September 2011
- August 2011
- July 2011
- June 2011
- May 2011
- April 2011
- March 2011
- February 2011

- January 2011
- December 2010
- November 2010
- October 2010
- September 2010
- August 2010
- July 2010
- June 2010
- May 2010
- April 2010
- March 2010
- February 2010
- January 2010
- December 2009

About the Author Google+