Kompüter proqramlaşdırması üzrə ara imtahan cavabları

1. C++ dilində OOP nədir

Obyekt yönümlü proqramlaşdırma (OOP) obyektlər ideyasına əsaslanan proqramlaşdırma paradiqmasıdır. Obyekt həm verilənləri, həm də həmin verilənlər üzərində işləmək üçün metodları ehtiva edən bir sinif nümunəsidir. C++ inkapsulyasiya, irsiyyət və polimorfizm kimi xüsusiyyətləri dəstəklədiyinə görə OOP üçün məşhur bir dildir.

C++ dilində obyektləri müəyyən etmək üçün siniflərdən istifadə olunur. Sinif məlumat üzvlərini və üzv funksiyalarını ehtiva edən istifadəçi tərəfindən müəyyən edilmiş məlumat növüdür. Məlumat üzvləri obyekt üçün verilənləri saxlayan dəyişənlərdir, üzv funksiyalar isə həmin verilənlər üzərində işləyən metodlardır.

İnkapsulyasiya OOP-da əsas anlayışdır ki, o, sinfin məlumat üzvlərinin və metodlarının icra detallarını kənar koddan gizlətməyi nəzərdə tutur. Bu, məlumat üzvlərini şəxsi etmək və onlara daxil olmaq və dəyişdirmək üçün ictimai üzv funksiyalarını təmin etməklə əldə edilir. Bu, obyektin məlumatlarının yalnız idarə olunan və proqnozlaşdırıla bilən şəkildə dəyişdirilməsini təmin etməyə kömək edir.

Vərəsəlik OOP-da əsas sinifin məlumat üzvlərini və metodlarını miras alan törəmə siniflərin yaradılmasına imkan verən digər mühüm konsepsiyadır. Bu, kodun təkrar istifadəsinə imkan verir və kodun təkrarlanmasının qarşısını alır. Polimorfizm C++ dilində virtual funksiyalar vasitəsilə də dəstəklənir ki, bu da müxtəlif siniflərin çoxsaylı obyektlərinin eyni sinifdən olduğu kimi rəftar edilməsinə imkan verir.

2.C++ dilində göstəricilərlə işləmək

C++ dilində göstəricilər digər dəyişənlərin yaddaş ünvanlarını saxlayan dəyişənlərdir. Onlar yaddaşı birbaşa manipulyasiya etmək üçün istifadə olunur və yaddaşı dinamik olaraq ayırmaq və boşaldmaq üçün istifadə edilə bilər.

C++-da göstəricilərlə işləmək yaddaşın idarə edilməsini başa düşməyi, həmçinin göstərici hesabını və istinadı ləğv etməyi tələb edir. Göstərici arifmetikası tam ədədləri yaddaşın müxtəlif yerlərinə köçürmək üçün göstəricilərdən əlavə və ya çıxmağı əhatə edir. Göstəriciyə istinadın ləğvi

göstəricinin göstərdiyi yaddaş ünvanında saxlanılan dəyərə daxil olmaq üçün * operatorunun istifadəsini nəzərdə tutur.

Yaddaş sızması və etibarsız göstərici istinadları kimi ümumi problemlərin qarşısını almaq üçün C++ dilində göstəricilərdən diqqətlə istifadə etmək vacibdir. Yaddaş sızması dinamik olaraq ayrılmış yaddaş düzgün bölüşdürülmədikdə, etibarsız göstərici istinadları isə artıq ayrılmış yaddaşa daxil olmaq üçün göstərici istifadə edildikdə baş verir.

3. Siyahı nədir və iş prinsipləri

C++-da siyahı siyahının əvvəlindən və ya sonundan daimi vaxt əlavələri və çıxarılmasına imkan verən ardıcıllıq konteyneridir. Siyahı ikiqat əlaqəli siyahı kimi həyata keçirilir, bu o deməkdir ki, siyahıdakı hər bir element siyahıdakı həm əvvəlki, həm də sonrakı elementə bir göstəriciyə malikdir.

Siyahı konteyneri siyahı ilə manipulyasiya etmək üçün bir sıra üzv funksiyaları təmin edir, o cümlədən daxil etmək, silmək, itələmək_ön, push_back, pop_front, pop_back və s. Bu üzv funksiyaları siyahıya elementləri əlavə etmək, silmək və daxil olmaq üçün istifadə edilə bilər.

C++ dilində siyahıdan istifadə nümunəsi:

```cpp
#include <iostream>
#include <list>

int main() {
    std::list<int> myList;

    myList.push_back(1);
    myList.push_back(2);
    myList.push_back(3);

    for (auto it = myList.begin(); it != myList.end(); ++it) {
        std::cout << *it << std::endl;
    }

    return 0;
}
```

Bu misalda std::list sinifindən istifadə edərək siyahı konteyneri yaradılır və elementlər push_back üzv funksiyasından istifadə edərək siyahıya əlavə edilir. For döngəsi iteratordan istifadə edərək siyahı üzərində təkrarlanır və hər bir element konsolda çap olunur.

4. Siyahı və Dəstlərin funksiyaları

Siyahı və dəst C++ dilində çox istifadə olunan iki konteyner sinifidir və onların hər birinin öz funksiyaları dəsti var.

Siyahının funksiyaları:

itələyin_ön : Siyahının əvvəlinə element əlavə edir.

geri itələmək : Siyahının sonuna element əlavə edir.

pop_front : Siyahıdan birinci elementi silir.

pop_back : Siyahıdan sonuncu elementi silir.

daxil edin: Siyahıda müəyyən edilmiş mövqeyə element daxil edir.

silmək: Siyahıda müəyyən edilmiş mövqedəki elementi silir.

ölçüsü: Siyahıdakı elementlərin sayını qaytarır.

ön : Siyahıdakı ilk elementi qaytarır.

geri : Siyahıdakı sonuncu elementi qaytarır.

Setin funksiyaları:

daxil edin :Çoxluğa element daxil edir.

silmək : Çoxluqdan elementi silir.

ölçü: Dəstdəki elementlərin sayını qaytarır.

tapmaq : Çoxluğu element üçün axtarır və tapılsa, ona iterator qaytarır.

saymaq : Çoxluqda müəyyən elementin baş vermə sayını qaytarır.

aydın: Setdən bütün elementləri silir.

Həm siyahı, həm də dəst C++-da məlumat kolleksiyalarını saxlamaq üçün faydalıdır. Siyahılar adətən konteynerin əvvəlindən və ya sonundan tez-tez daxil edilməsi və çıxarılması tələb olunduqda, dəstlər isə elementlərin unikallığı və sıralanması vacib olduqda istifadə olunur.

5. Konstruktorlar nədir və misal yazın

Konstruktorlar nədir və bir nümunə yazın:

Konstruktorlar C++ dilində obyekt yaradılan zaman çağırılan xüsusi üzv funksiyalardır. Onlar obyektin məlumat üzvlərini ilkin qiymətlərinə başlamaq üçün istifadə olunur. Konstruktorlar siniflə eyni ada malikdir və qaytarma növü yoxdur. Obyektin işə salınmasının müxtəlif yollarına icazə vermək üçün onlar həddən artıq yüklənə bilər.

Budur konstruktoru olan bir sinif nümunəsi:

```
#include <iostream>

class Person {
public:
    std::string name;
    int age;

    Person(std::string n, int a) {
        name = n;
        age = a;
    }
};

int main() {
    Person p("John", 25);
    std::cout << p.name << " is " << p.age << " years old." <<
        std::endl;
    return 0;
}
```

Bu misalda Person sinfində iki parametr, ad və yaş götürən və obyektin adını və yaş məlumatı üzvlərini işə salan konstruktor var. Əsas funksiyada konstruktordan istifadə etməklə yeni Person obyekti yaradılır və onun məlumat üzvlərinə daxil olur və konsolda çap olunur.

6.Daimi aqreqatlar,Daimi göstəricilər

C++-da sabit aqreqat const elan edilən məcmu növüdür (məsələn, massiv, struktur və ya birlik). Bu o deməkdir ki, aqreqatdakı elementlərin qiymətləri işə salındıqdan sonra dəyişdirilə bilməz. Misal üçün:

```
const int arr[] = {1, 2, 3};
```

Bu misalda arr massivi sabit məcmudur və onun elementlərinin qiymətləri dəyişdirilə bilməz.

Sabit göstərici const elan edilən göstəricidir, yəni göstəricinin dəyərini (yəni onun göstərdiyi yaddaş ünvanı) işə saldıqdan sonra dəyişdirmək mümkün deyil. Bununla belə, göstəricinin göstərdiyi obyektin dəyəri hələ də dəyişdirilə bilər. Misal üçün:

```
int x = 5;
const int* ptr = &x;
```

Bu nümunədə ptr x dəyişəninə işarə edən sabit göstəricidir. The ptr dəyəri (yəni, x-in yaddaş ünvanı) dəyişdirilə bilməz, lakin x dəyəri hələ də dəyişdirilə bilər.

7.C++ dilində spesifikatorlara giriş

C++ dilində giriş spesifikatorları sinif üzvlərinin giriş səviyyəsini təyin etmək üçün istifadə olunan açar sözlərdir. C++ dilində üç giriş spesifikatoru var: ictimai, özəl və qorunan.

İctimai üzvlərə sinfin obyektinə daxil ola bilən istənilən kodla daxil olmaq olar. Şəxsi üzvlərə yalnız üzv funksiyaları ilə daxil ola bilərsiniz

eyni sinif. Qorunan üzvlərə eyni sinfin üzv funksiyaları və ya törəmə siniflərin üzv funksiyaları vasitəsilə daxil olmaq olar.

Budur bir nümunə:

```
#include <iostream>

class Person {
public:
    std::string name;
    int age;

    void print() {
        std::cout << name << " is " << age << " years old."
            << std::endl;
    }

private:
    int salary;
};

int main() {
    Person p;
    p.name = "John";
    p.age = 25;
    p.print();
    // p.salary = 50000; // This line will cause a compile
        error
    return 0;
}
```

Bu misalda Person sinfində print() ictimai üzv funksiyası var sinfin obyektinə daxil ola bilən istənilən kodla daxil olmaq olar. The class da yalnız əldə edilə bilən şəxsi məlumat üzvü maaşına malikdir eyni sinfin üzv funksiyaları. Əsas funksiyada bir obyekt Şəxsiyyət sinfi yaradılır və onun ictimai məlumat üzvlərinə daxil olur və konsolda çap olunur. Şəxsi məlumat üzvünə daxil olmaq cəhdi əmək haqqı kompilyasiya xətası ilə nəticələnir.

8. Funksiyaların həddən artıq yüklənməsi

Funksiyaların həddən artıq yüklənməsi C++-da proqramçıya eyni adlı, lakin müxtəlif parametr növləri və ya parametrlərin nömrələri ilə birdən çox funksiyanı təyin etməyə imkan verən xüsusiyyətdir. Funksiya arqumentlərlə çağırıldıqda, kompilyator ötürülən arqumentlərin növlərinə və sayına əsasən çağırmaq üçün uyğun funksiyanı seçir.

Budur bir nümunə:

```cpp
#include <iostream>

int add(int a, int b) {
    return a + b;
}

double add(double a, double b) {
    return a + b;
}

int main() {
    std::cout << add(1, 2) << std::endl;
    std::cout << add(2.5, 3.5) << std::endl;
    return 0;
}
```

Bu nümunədə "əlavə et" adlı iki funksiya var, biri ikisini alır tam parametrlər və iki cüt parametr alan digəri. Nə n əlavə funksiyası tam ədədlərlə çağırılır, birinci əlavə funksiyası adlanır və ikiqatlarla çağırıldıqda ikinci əlavə funksiyası çağırılır.

9.C++ dilində strukturlar nədir və nümunə yazın

C++ dilində struktur, müxtəlif məlumat növlərinin əlaqəli məlumat üzvlərini bir vahiddə qruplaşdıran istifadəçi tərəfindən müəyyən edilmiş məlumat növüdür. Struktur sinfə bənzəyir, lakin onun üzvlərinə defolt ictimai girişi var.

Budur bir nümunə:

```cpp
#include <iostream>

struct Person {
    std::string name;
    int age;
};

int main() {
    Person p;
    p.name = "John";
    p.age = 25;
    std::cout << p.name << " is " << p.age << " years old." << std
        ::endl;
    return 0;
}
```

Bu nümunədə, "Şəxs" adlı struktur iki məlumat üzvü ilə müəyyən edilir, "ad" sətri və "yaş" tam ədədi. Əsas funksiyada bir obyekt Şəxs strukturu yaradılır və onun məlumat üzvlərinə daxil olur və çap olunur konsol.

10. C++ dilində OOP konsepsiyaları

Obyekt yönümlü proqramlaşdırma (OOP) problemləri həll etmək üçün bir-biri ilə qarşılıqlı əlaqədə olan təkrar istifadə edilə bilən və modul obyektlərə kodu təşkil etməyə yönəlmiş proqramlaşdırma paradiqmasıdır. C++ dilində əsas OOP anlayışları bunlardır:

- Siniflər və obyektlər: Siniflər obyektlər yaratmaq üçün planlardır və obyektlər öz unikal verilənləri və davranışları olan sinif nümunələridir.
- Enkapsulyasiya: İnkapsulyasiya, bir sinfin tətbiq detallarını istifadəçidən gizlətmək və onun məlumatlarına və davranışlarına daxil olmaq üçün ictimai interfeys təmin etmək təcrübəsidir.
- Varislik: Vərəsəlik mövcud sinifdən yeni bir sinif yaratma prosesidir, burada yeni sinif mövcud sinfin məlumatlarını və davranışlarını miras alır və öz məlumatlarını və davranışlarını əlavə edə və ya dəyişdirə bilər.
- Polimorfizm: Polimorfizm, daha çevik və modul kod yaratmağa imkan verən müxtəlif sinif obyektlərinin bir-birini əvəz edə bilmə qabiliyyətidir.

11. İnkapsulyasiya

İnkapsulyasiya, bir sinfin tətbiq detallarını istifadəçidən gizlətmək və onun məlumatlarına və davranışlarına daxil olmaq üçün ictimai interfeys təmin etmək təcrübəsinə istinad edən bir OOP konsepsiyasıdır. İnkapsulyasiya, istifadəçilərin ictimai interfeysdən keçmədən sinif məlumatlarına və davranışlarına birbaşa daxil olmaq və ya dəyişdirməkdən çəkindirməklə kodun daha yaxşı təşkilinə, modulluğuna və təhlükəsizliyinə imkan verir.

Budur bir nümunə:

```cpp
#include <iostream>

class BankAccount {
private:
    int accountNumber;
    double balance;

public:
    BankAccount(int accNum, double bal) {
        accountNumber = accNum;
        balance = bal;
    }

    void deposit(double amount) {
        balance += amount;
    }

    void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        }
    }

    double getBalance() {
        return balance;
    }
};

int main() {
```

12. Miras

Varislik obyekt yönümlü proqramlaşdırmanın əsas xüsusiyyətidir (OOP) ki, siniflərə bir valideyndən xassələri və metodları miras almağa imkan verir sinif. C++ dilində miras sintaksisdən istifadə etməklə əldə edilir **sinif DerivedClass : accessSpecifier BaseClass**, harada **DerivedClass** olan sinifdir varislik, **BaseClass** miras alınan ana sinifdir və **accessSpecifier** törəmə sinfin malik olduğu giriş səviyyəsini müəyyən edir baza sinfinin üzvləri (ictimai, qorunan və ya özəl). Miras tək, çox, iyerarxik və çoxsəviyyəli kimi bir neçə növ ola bilər miras. Budur miras nümunəsi:

```
class Animal {
   public:
      void eat() {
         cout << "Eating..." << endl;
      }
};

class Dog : public Animal {
   public:
      void bark() {
         cout << "Barking..." << endl;
      }
};

int main() {
   Dog myDog;
   myDog.eat();
   myDog.bark();
   return 0;
}
```

Yuxarıdakı misalda,**it**sinfindən yaranmışdır**Heyvan**ictimai mirasdan istifadə edən sinif. Bu o deməkdir ki, ictimaiyyət nümayəndələri**Heyvan** sinifə daxil olmaq mümkündür**it**sinif. The**it**sinifin də öz metodu var, **qabıq()**. Bir obyekt yaratdıqda**it**sinif və zəng edin**yemək()**və **qabıq()**üsulları, hər iki üsul icra edilir, çünki**it**sinfi miras alır **yemək()**üsulundan**Heyvan**sinif.

13.Polimorfizm

Polimorfizm OOP-da obyektlərin qabiliyyətinə aid olan bir anlayışdır bir çox formalar alır. C++ dilində polimorfizm ikidən istifadə etməklə əldə edilir üsullar: funksiyanın həddən artıq yüklənməsi və funksiyanın ləğvi. Funksiya həddindən artıq yükləmə eyni adlı, lakin fərqli bir çox funksiyaya imkan verir Parametrlərin eyni miqyasda mövcud olmasına baxmayaraq, funksiyanın ləğvi a törəmə sinfi öz tətbiqini təmin etmək üçün artıq bir metoddur ana sinfində müəyyən edilmişdir. Polimorfizm bizə kod yazmağa imkan verir müxtəlif sinif obyektləri ilə onların dəqiq tipini bilmədən işləmək. Funksiyadan istifadə edərək polimorfizm nümunəsi:

```
class Animal {
   public:
      virtual void makeSound() {
         cout << "The animal makes a sound..." << endl;
      }
};

class Dog : public Animal {
   public:
      void makeSound() {
         cout << "The dog barks..." << endl;
      }
};

class Cat : public Animal {
   public:
      void makeSound() {
         cout << "The cat meows..." << endl;
      }
};
```

```
int main() {
   Animal* myAnimal = new Animal();
   Animal* myDog = new Dog();
   Animal* myCat = new Cat();

   myAnimal->makeSound();
   myDog->makeSound();
   myCat->makeSound();

   delete myAnimal;
   delete myDog;
   delete myCat;
   return 0;
}
```

Yuxarıdakı misalda, **Heyvan** sinifin virtual metodu var **makeSound()**. The **it** və **pişik** sinifləri miras alır **Heyvan** və özlərinin həyata keçirilməsini təmin edirlər **makeSound()** üsul. obyektləri yaratdıqda **Heyvan**, **it**, və **pişik** dərsləri oxuyun və zəng edin **makeSound()** Bunların hər biri üzərində uyğun üsulla həyata keçirilməsi **makeSound()** edir növündən asılı olaraq hər bir obyekt üçün yerinə yetirilir

14. Funksiyalara işarələr

C++-da funksiyalar dəyişənlər kimi də qəbul edilə bilər və biz ötürmək üçün funksiyalara göstəricilərdən istifadə edə bilərik. digər funksiyalar üçün arqument kimi və ya onları massivlərdə saxlamaq üçün funksiyaları yerinə yetirir. Budur bir nümunə:

```
#include <iostream>

using namespace std;

int add(int a, int b) {
   return a + b;
}

int subtract(int a, int b) {
   return a - b;
}

int main() {
   int (*p)(int, int); // declare a pointer to a function

   p = add; // assign the address of the add function to p
   cout << p(2, 3) << endl; // call the add function using p

   p = subtract; // assign the address of the subtract function
       to p
   cout << p(5, 2) << endl; // call the subtract function using p

   return 0;
}
```

Yuxarıdakı nümunədə bir göstərici elan edirik **səh** i tam arqument alan funksiyaya və tam dəyər qaytarır. Sonra ünvanını təyin edirik **əlavə et()** funksiyası **səh** ünvanı zəng edin göstəricidən istifadə etməklə. Sonra ünvanını təyin edirik **çıxmaq()** funksiyası **səh** ünvanı zəng edin göstəricidən istifadə etməklə.

15.Scanf, printf, cin cout fərqi

scanf()və**printf()**C dilində giriş/çıxış funksiyalarıdır**cin**və**cout**C++ dilində giriş/çıxış obyektləridir. Onların əsas fərqi ondadır**scanf()**və**printf()**giriş və ya çıxışın məlumat tipini təyin etmək üçün format təyinedicilərindən istifadə edin**cin**və**cout**istifadə edin**<<**və **> >** məlumatları oxumaq və ya yazmaq üçün operatorlar.

Budur istifadə nümunəsi**scanf()**və**printf()**:

```c
#include <stdio.h>

int main() {
   int age;
   char name[20];

   printf("Enter your name: ");
   scanf("%s", name);
   printf("Enter your age: ");
   scanf("%d", &age);

   printf("Your name is %s and you are %d years old.\n", name, age
       );
   return 0;
}
```

Və burada eyni nümunə istifadə olunur **cin** və **cout**:

```c
#include <iostream>

using namespace std;

int main() {
   int age;
   string name;

   cout << "Enter your name: ";
   cin >> name;
   cout << "Enter your age: ";
   cin >> age;

   cout << "Your name is " << name << " and you are " << age << "
       years old." << endl;
   return 0;
}
```

İçində **scanf()** və **printf()** məsələn, istifadə edirik **%s** və **%d**-da oxumaq üçün format təyinediciləri müvafiq olaraq sətir və tam ədəd. Biz də istifadə edirik **&** ünvanını ötürmək üçün operator **yaş** üçün dəyişən **scanf()**. İçində **cin** və **cout** məsələn, istifadə edirik **> >**-də oxumaq üçün operator sətir və tam dəyərlər. Biz də istifadə edirik **endl** yeni sətir simvolunu çap etmək üçün manipulyator çıxışın sonu.

16. Konstruktorun həddən artıq yüklənməsi

> C++-da konstruktor hər hansı digər funksiya kimi həddən artıq yüklənə bilər. Bu o deməkdir ki, biz müxtəlif parametr siyahıları olan bir sinif üçün çoxlu konstruktor təyin edə bilərik. Həmin sinfin obyekti yaradıldıqda, ötürülən arqumentlər əsasında müvafiq konstruktor çağırılacaq.

Burada iki konstruktoru olan bir sinif nümunəsidir:

```
class Person {
   private:
      string name;
      int age;

   public:
      Person(string n, int a) {
         name = n;
         age = a;
      }

      Person() {
         name = "Unknown";
         age = 0;
      }

      void display() {
         cout << "Name: " << name << endl;
         cout << "Age: " << age << endl;
      }
```

Yuxarıdakı misalda biz iki konstruktor təyin edirik **Şəxs** sinif. İlk konstruktor iki parametr götürür, **n** və **a**, və başlatır **ad** və **yaş** bunlarla üzv dəyişənlər dəyərlər. İkinci konstruktor heç bir parametr qəbul etmir və başlanğıc verir **ad** və **yaş** üzv r standart dəyərləri olan dəyişənlər. Bundan sonra obyektləri yarada bilərik **Şəxs** ya istifadə edərək sinif konstruktor:

```
Person p1("John", 30);
p1.display(); // output: Name: John Age: 30

Person p2;
p2.display(); // output: Name: Unknown Age: 0
```

17.Həddindən artıq yüklənmiş konstruktorun seçilməsi

Bir sinif üçün bir neçə konstruktor olduqda, obyekt yaradılarkən ötürülən arqumentlər əsasında müvafiq konstruktor seçilir. Arqument növləri ilə konstruktorlardan birinin parametr növləri arasında dəqiq uyğunluq varsa, həmin konstruktor çağırılacaq. Əks halda kompilyator arqumentləri uyğun tiplərə çevirməyə çalışacaq və uyğunluq aşkar edilərsə, həmin konstruktor çağırılacaq.

Budur bir nümunə:

```
class Person {
   private:
      string name;
      int age;

   public:
      Person(string n, int a) {
         name = n;
         age = a;
      }

      Person(int a) {
         name = "Unknown";
         age = a;
      }

      void display() {
         cout << "Name: " << name << endl;
         cout << "Age: " << age << endl;
      }
```

```
};

int main() {
    Person p1("John", 30); // calls first constructor
    p1.display(); // output: Name: John Age: 30

    Person p2(40); // calls second constructor
    p2.display(); // output: Name: Unknown Age: 40

    return 0;
}
```

Yuxarıdakı nümunədə birinci obyekt **p1** var, çünki birinci konstruktordan istifadə etməklə yaradılır arqument növləri arasında dəqiq uyğunluq (**simli** və **int**) və parametr növləri ilk konstruktor. İkinci obyekt **səh2** ar, çünki ikinci konstruktordan istifadə etməklə yaradılır arqument növü üçün dəqiq uyğunluq yoxdur (**int**), lakin uyğun tipə çevrilə bilər.

18.Obyektlərə göstəricilərin massivləri

C++ dilində obyektlərə göstəricilər massivləri yaratmaq mümkündür. Bu, obyektləri dinamik şəkildə yaratmaq və onları massivdə saxlamaq istədiyimiz vəziyyətlərdə faydalı ola bilər. Hər bir obyekti yaratmaq və onun ünvanını massivin elementinə təyin etmək üçün dövrədən istifadə edə bilərik.

Məsələn, iki arqument (uzunluq və genişlik) götürən konstruktoru olan "Düzbucaqlı" adlı sinfi nəzərdən keçirək:

```
class Rectangle {
  private:
     int length, width;
  public:
     Rectangle(int l, int w) { length = l; width = w; }
};
```

Rectangle obyektlərinə göstəricilər massivi yaratmaq üçün biz ayırmaq üçün new operatorundan istifadə edə bilərik massiv üçün yaddaş:

```
Rectangle** rectangles = new Rectangle*[10];
```

Bu, Rectangle obyektlərinə 10 göstəricidən ibarət massiv yaradır. Hər bir obyekt yaratmaq və onu təyin etmək massivin elementinə müraciət etmək üçün bir döngədən istifadə edə bilərik:

```
for (int i = 0; i < 10; i++) {
   rectangles[i] = new Rectangle(i+1, i+2);
}
```

Bu, hər biri fərqli uzunluq və eni olan 10 Rectangle obyekti yaradır və onların ünvanlarını "düzbucaqlılar" massivində saxlayır.

Massivdəki obyektə daxil olmaq üçün biz massiv indeksi operatorundan və göstəriciyə istinad operatorundan istifadə edə bilərik:

```
Rectangle* rect = rectangles[3];
int length = rect->getLength();
int width = rect->getWidth();
```

Bu, massivdəki dördüncü obyektin ünvanını alır, obyektin özünə daxil olmaq üçün ona istinad edir və düzbucaqlının uzunluğunu və enini əldə etmək üçün "getLength()" və "getWidth()" funksiyalarını çağırır.

Obyektlərdən və massivdən istifadəni bitirdikdən sonra sil operatoru ilə ayrılmış yaddaşı boşaltmalıyıq:

```
for (int i = 0; i < 10; i++) {
   delete rectangles[i];
}
delete[] rectangles;
```

Bu, hər bir obyekti və obyektlərə göstəricilər massivini silir.

19.Obyektlərin içərisindəki obyektlər

C++ dilində başqa obyektlərin daxilində obyektləri müəyyən etmək mümkündür. Bu kompozisiya kimi tanınır və daha sadə obyektləri birləşdirərək mürəkkəb obyektlər yaratmağa imkan verən obyekt yönümlü proqramlaşdırma formasıdır.

Məsələn, iki ölçülü fəzada nöqtəni x və y koordinatları ilə təmsil edən "Nöqtə" adlı sinfi nəzərdən keçirək:

```
class Point {
   private:
      int x, y;
   public:
      Point(int xval, int yval) { x = xval; y = yval; }
};
```

İndi tutaq ki, biz iki-də bir düzbucaqlı təmsil edən "Düzbucaqlı" adlı bir sinif təyin etmək istəyirik. düzbucaqlının əks künclərini təmsil etmək üçün iki "Nöqtə" obyektindən istifadə edərək ölçülü boşluq:

```
class Rectangle {
   private:
      Point topLeft, bottomRight;
   public:
      Rectangle(Point tl, Point br) { topLeft = tl; bottomRight =
          br; }
};
```

Bu nümunədə, "Düzbucaqlı" sinfi iki "Nöqtə" obyektini ehtiva edir - biri düzbucağın yuxarı küncünü, digəri isə aşağı sağ küncü təmsil edir. Biz “Düzbucaqlı” obyekti yaratdıqda, bu üzv dəyişənləri işə salmaq üçün iki “Nöqtə” obyektini konstruktora ötürürük.

İndi müəyyən etdiyimiz "Nöqtə" obyektlərindən istifadə edərək "Dördbucaqlı" obyektləri yarada bilərik:

```
Point topLeft(0, 0);
Point bottomRight(5, 5);
Rectangle rect(topLeft, bottomRight);
```

Bu, yuxarı sol küncü (0,0) və sağ alt küncü isə "Dördbucaqlı" obyekti yaradır. (5,5).

20. Siyahı və Linkedlist arasındakı fərq

C++ dilində List və Linked List elementlər toplusunu saxlamaq üçün istifadə olunan iki məlumat strukturudur. Hər ikisinin öz üstünlükləri və mənfi cəhətləri var və düzgün birini seçmək istifadə vəziyyətindən asılıdır.

Siyahı elementləri xətti ardıcıllıqla saxlayan ardıcıl konteynerdir. Bu, elementlərin sürətli daxil edilməsinə və silinməsinə, həmçinin indeksdən istifadə edərək elementlərə təsadüfi girişə imkan verir. Siyahı massiv və ya dinamik massivdən istifadə etməklə həyata keçirilir.

Digər tərəfdən, Əlaqəli Siyahı elementləri qovşaqlar zəncirində saxlayan ardıcıl olmayan bir konteynerdir. Hər bir node bir element və zəncirdəki növbəti node üçün bir göstəricidən ibarətdir. Əlaqəli Siyahılar siyahının istənilən nöqtəsində elementlərin səmərəli daxil edilməsinə və silinməsinə imkan verir, lakin elementlərə təsadüfi giriş mümkün deyil, çünki bu, siyahıdan əvvəldən keçməyi tələb edir.

Siyahı və Əlaqəli Siyahı arasındakı əsas fərqlərdən bəziləri bunlardır:

1. İcra: Siyahı massiv və ya dinamik massivdən istifadə etməklə həyata keçirilir, Əlaqəli Siyahı isə qovşaqlar və göstəricilərdən istifadə etməklə həyata keçirilir.
2. Giriş: Siyahı indeksdən istifadə edərək elementlərə təsadüfi daxil olmağa imkan verir, halbuki Əlaqəli Siyahı elementə daxil olmaq üçün siyahıdan keçməyi tələb edir.
3. Daxiletmə və Silinmə: Siyahı siyahının əvvəlində və ya sonunda əlavə etmək və silmək üçün sabit vaxt mürəkkəbliyinə, ortada isə daxil etmək və ya silmək üçün xətti vaxt mürəkkəbliyinə malikdir. Əlaqəli Siyahı siyahının istənilən nöqtəsində daxil etmək və silmək üçün sabit vaxt mürəkkəbliyinə malikdir.
4. Yaddaş İstifadəsi: Siyahıda bitişik yaddaş ayrılmasından, Əlaqəli Siyahıda isə bitişik olmayan yaddaş ayrılmasından istifadə edilir.
5. Yaddaşın yuxarı həddi: Siyahıda Bağlı Siyahıdan daha kiçik yaddaş yükü var, çünki o, yalnız elementləri və birinci və sonuncu elementlərə göstəricini saxlayır. Əlaqəli Siyahı hər bir qovşaq üçün əlavə göstərici tələb edir.

Xülasə, List sürətli təsadüfi giriş tələb olunduqda və elementlərin sayı əvvəlcədən məlum olduqda yaxşı seçimdir. Digər tərəfdən, Əlaqəli Siyahı səmərəli daxiletmə və silmə tələb olunduqda və elementlərin sayı əvvəlcədən məlum olmayanda yaxşı seçimdir.

21. Virtual üsullar və onun istifadəsi

C++-da virtual metodlar polimorfizmin vacib xüsusiyyətidir və törəmə siniflərə öz baza siniflərində müəyyən edilmiş metodların davranışını ləğv etməyə imkan verir. Baza sinfi metodu virtual olaraq elan etdikdə, gec bağlanma və ya dinamik göndərmə imkanı verir, bu o deməkdir ki, müvafiq metodun icrası elan edilmiş tipdən çox faktiki obyekt növü əsasında icra müddətində müəyyən edilir.

C++-da virtual metodu müəyyən etmək üçün onu əsas sinifdə virtual olaraq elan etməli və həyata keçirilməsini təmin etməlisiniz. Budur bir nümunə:

```cpp
class Base {
public:
    virtual void foo() {
        // Base class implementation
    }
};

class Derived : public Base {
public:
    void foo() override {
        // Derived class implementation
    }
};
```

Bu misalda foo() metodu Base sinfində virtual olaraq elan edilmişdir. Derived sinfi daha sonra bu metodu override açar sözündən istifadə edərək ləğv edir və onun xüsusi həyata keçirilməsini təmin etmək niyyətində olduğunu göstərir. Əgər əsas sinif metodu düzgün ləğv edilmirsə, ləğv açar sözü kompilyasiya zamanı səhvləri tutmağa kömək edir.

Əlinizdə faktiki olaraq törəmə sinif obyektinə işarə edən əsas sinif obyektinə göstərici və ya istinadınız olduqda, virtual metodu işə sala bilərsiniz və müvafiq icra faktiki obyekt növü əsasında çağırılacaq:

```cpp
Base* basePtr = new Derived();
basePtr->foo();  // Calls the derived class implement
```

Virtual metodlar, adətən, əldə edilmiş siniflər tərəfindən ixtisaslaşdırıla bilən ümumi interfeys və ya davranışı təmin etmək üçün baza siniflərində istifadə olunur. Bu, dəqiq törəmə sinif tipini bilmədən əsas sinif göstəriciləri və ya istinadlar üzərində işləyən kodu yazmağa imkan verir. O, obyekt yönümlü proqramlaşdırmada kodun təkrar istifadəsinə, genişlənməsinə və çevikliyinə imkan verir.

22.Baza funksiyasından virtual metodun çağırılması

C++-da, əgər siz əsas sinif funksiyasından virtual metodu çağırmaq istəyirsinizsə, sadəcə olaraq metodu virtual olmayan funksiya kimi çağıra bilərsiniz. Alınan sinfin faktiki icrası obyektin işləmə növünə əsasən çağırılacaq.

Budur bir nümunə:

```cpp
#include <iostream>

class Base {
public:
    virtual void foo() {
        std::cout << "Base::foo()" << std::endl;
    }

    void bar() {
        std::cout << "Base::bar()" << std::endl;
        foo();  // Call the virtual method
    }
};

class Derived : public Base {
public:
    void foo() override {
        std::cout << "Derived::foo()" << std::endl;
    }
};

int main() {
    Base* basePtr = new Derived();
    basePtr->bar();  // Calls Base::bar() and Derived::foo()

    delete basePtr;
    return 0;
}
```

BundaMəsələn, bizdə foo() virtual metodu və qeyri-virtual metod bar() ilə Base baza sinifimiz var. bar() funksiyası virtual olan foo() metodunu çağırır.

Bizdə həmçinin öz tətbiqi ilə foo() metodunu ləğv edən törəmə sinifimiz var.

main() funksiyasında biz Derived tipli obyektə işarə edən Base* tipli basePtr göstəricisi yaradırıq. Biz basePtr->bar() adlandırdığımız zaman o, əvvəlcə Base::bar() funksiyasını çağırır, o da sonra daxili olaraq çağırır.

foo(). Bununla belə, basePtr faktiki olaraq Derived obyektinə işarə etdiyi üçün ləğv edilmiş Derived::foo() metodu işə salınır.

Proqramın nəticəsi belə olacaq:

```
Base::bar()
Derived::foo()
```

Gördüyünüz kimi, əsas sinifdə bar() funksiyası daxilindən çağırılan virtual metod foo() törəmə sinifdə ləğv edilmiş tətbiqi işə salır.

23.Çoxlu vərəsəlik

Çoxlu irsiyyət C++-da bir sinfə bir neçə əsas sinifdən miras almağa imkan verən xüsusiyyətdir. Bu o deməkdir ki, törəmə sinif birdən çox ana sinifdən üzvlər və funksionallıq miras ala bilər.

Çoxlu irsiyyəti nümayiş etdirmək üçün bir nümunə:

```cpp
#include <iostream>

// Base class 1
class Base1 {
public:
    void display1() {
        std::cout << "Base1::display1()" << std::endl;
    }
};

// Base class 2
class Base2 {
public:
    void display2() {
        std::cout << "Base2::display2()" << std::endl;
    }
};

// Derived class inheriting from Base1 and Base2
class Derived : public Base1, public Base2 {
```

```
public:
    void display3() {
        std::cout << "Derived::display3()" << std::endl;
    }
};

int main() {
    Derived obj;
    obj.display1();  // Call Base1::display1()
    obj.display2();  // Call Base2::display2()
    obj.display3();  // Call Derived::display3()

    return 0;
}
```

Bu nümunədə iki əsas sinifimiz var, Base1 və Base2. Alınmış sinif Derived çoxlu mirasdan istifadə edərək həm Base1, həm də Base2-dən miras alır. Alınmış sinif hər iki əsas sinifdən üzvlərə və funksiyalara daxil ola bilər.

display1() funksiyası Base1-ə, display2() Base2-ə, display3() isə Derived-ə aiddir. main() funksiyası Derived sinfinin obyektində bu funksiyaların çağırılmasını nümayiş etdirir.

Qeyd etmək vacibdir ki, çoxsaylı miras almaz problemi və bir neçə əsas sinifdə eyni ada malik üzvlər olduqda qeyri-müəyyənlik kimi müəyyən mürəkkəbliklərə səbəb ola bilər. Belə hallarda, əhatə dairəsi operatorundan (::) istifadə edərək, əhatə dairəsindən istifadə etməli və ya qeyri-müəyyənliyi aydın şəkildə həll etməlisiniz.

24. C++-da istisnalar

C++-da istisnalar proqramın icrası zamanı baş verən müstəsna vəziyyətləri və ya səhvləri idarə etmək üçün mexanizm təmin edir. İstisna müəyyən bir növ səhv və ya müstəsna vəziyyəti təmsil edən obyektdir. Səhv baş verdikdə, siz istisna ata bilərsiniz və bu işlənilməsə, proqram dayandırılacaq.

C++ dilində istisnalarla işləməyə dair əsas icmal:

İstisna atmaq:

Səhv və ya müstəsna vəziyyəti göstərmək üçün throw açar sözündən istifadə edərək istisna ata bilərsiniz. İstənilən obyekti istisna kimi ata bilərsiniz, lakin əvvəlcədən təyin edilmiş istisna növlərindən istifadə etmək və ya xüsusi istisna sinifləri yaratmaq adi haldır.

Misal:

```
throw std::runtime_error("An error occurred!");
```

İstisnaların tutulması:

İstisnaları idarə etmək üçün siz try-catch blokundan istifadə edirsiniz. Try bloku istisna yarada biləcək kodu ehtiva edir və tutma bloku baş verərsə, istisnanı idarə edir. Müxtəlif növ istisnaları idarə etmək üçün bir neçə tutma blokunuz ola bilər.

Misal:

```
try {
    // Code that might throw an exception
} catch (const std::exception& ex) {
    // Exception handling code
}
```

Xüsusi İstisnaları Tutmaq:

Catch blokunda istisna növünü göstərərək xüsusi istisnaları tuta bilərsiniz. Bu, müxtəlif növ istisnaları fərqli şəkildə idarə etməyə imkan verir.

Misal:

```
try {
    // Code that might throw an exception
} catch (const std::runtime_error& ex) {
    // Exception handling code for runtime_error
} catch (const std::exception& ex) {
    // Exception handling code for other exception
}
```

Tutulmamış İstisnaların idarə edilməsi:

İstisna atılırsa, lakin tutulmazsa, o, uyğun tutma blokuna çatana və ya proqramı dayandırana qədər zəng yığınını yayacaq. Tutulmamış istisnaları idarə etmək üçün std::terminate() və ya std::set_terminate() istifadə edərək qlobal tutma blokunu təyin edə bilərsiniz.

Misal:

```
std::set_terminate([]() {
    std::cout << "Uncaught exception occurred!" << std::endl;
    std::terminate();
});
```

Fərdi İstisna Siniflərinin yaradılması:

std::exception və ya onun törəmə siniflərindən hər hansı birini əldə etməklə öz istisna siniflərinizi yarada bilərsiniz. Xüsusi istisna sinifləri səhv və ya müstəsna vəziyyət haqqında xüsusi məlumat təqdim etməyə imkan verir.

Misal:

```
class MyException : public std::exception {
public:
    const char* what() const noexcept override {
        return "Custom exception occurred!";
    }
};

throw MyException();
```

İstisnalar səhvləri və müstəsna şərtləri strukturlaşdırılmış şəkildə idarə etmək üçün güclü mexanizm təmin edir. İstisnaları atmaqla və tutmaqla siz C++ proqramlarınızda səhvləri zərif şəkildə idarə edə və müvafiq tədbirlər görə bilərsiniz.

25. İstisnaların idarə olunması metodu

C++-da istisnalarla işləyərkən, try-catch mexanizmindən istifadə etməklə onları idarə edə bilərsiniz. Try bloku istisna yarada biləcək kodu ehtiva edir və tutma bloku atılan istisnanı idarə etmək üçün istifadə olunur. C++-da istisnaların necə idarə olunacağına dair bir nümunə:

```cpp
#include <iostream>

int main() {
    try {
        // Code that might throw an exception
        int numerator = 10;
        int denominator = 0;
        if (denominator == 0) {
            throw std::runtime_error("Divide by zero exception"
                );
        }
        int result = numerator / denominator;
        std::cout << "Result: " << result << std::endl;
    } catch (const std::exception& ex) {
        // Exception handling code
        std::cout << "Exception caught: " << ex.what() << std
            ::endl;
    }

    return 0;
```

Bu nümunədə try bloku bölmə əməliyyatını yerinə yetirən kodu ehtiva edir. Məxrəc sıfırdırsa, throw ifadəsindən istifadə edərək std::runtime_error istisnası atılır. Tutmaq bloku atılan istisnanın idarə edilməsinə cavabdehdir. İstisna növünü (std::exception) göstərərək istisnanı tutur və ex adlı tutulan istisna obyektinə daimi istinad yaradır. Catch blokunun içərisində siz istisnanı idarə etmək üçün kod yaza bilərsiniz. Bu halda o, sadəcə what() funksiyasından istifadə edərək istisna mesajını çap edir.

Əgər try blokunda bir istisna atılırsa, proqram axını dərhal müvafiq tutma blokuna keçir. Uyğun istisna növü ilə tutma bloku icra olunur. Uyğun tutma bloku tapılmadıqda, istisna uyğun tutma blokuna çatana və ya proqramı dayandırana qədər çağırış yığınını yayır.

Qeyd etmək vacibdir ki, müxtəlif növ istisnaları idarə etmək üçün bir neçə tutma blokunuz ola bilər. Tutma blokları ardıcıllıqla yoxlanılır və ilk uyğun gələn tutma bloku yerinə yetirilir.

Sınaq-tutmaq mexanizmindən istifadə edərək, siz C++ kodunuzda istisnaları idarə edə və səhvləri və ya müstəsna şərtləri zərif şəkildə idarə edə bilərsiniz, bu da sizə müvafiq səhvlərin idarə edilməsi və bərpa məntiqini təmin etməyə imkan verir.

26. try-catch blokları ilə istisnaların tutulması:

C++-da istisnalar try-catch blokları vasitəsilə tutula və idarə oluna bilər. Try bloku istisna yarada biləcək kodu ehtiva edir və tutma bloku atılan istisnanı idarə etmək üçün istifadə olunur. Budur ümumi sintaksis:

```
try {
    // Code that might throw an exception
} catch (ExceptionType1& ex) {
    // Exception handling code for ExceptionType1
} catch (ExceptionType2& ex) {
    // Exception handling code for ExceptionType2
} catch (...) {
    // Exception handling code for any other exceptio
}
```

Bu sintaksisdə hər biri müəyyən bir istisna növü ilə məşğul olan bir neçə tutma blokunuz ola bilər. İstisna növünə uyğun gələn tutma bloku həmin istisna növü atıldıqda yerinə yetirilir. Son tutma blokundaki ellipsis ... hər hansı digər işlənməmiş istisnaları tutmaq üçün istifadə olunur.

27. STL kitabxana elementləri nədir:

STL (Standart Şablon Kitabxanası) ümumi alqoritmlərin və məlumat strukturlarının toplusunu təmin edən C++ dilində güclü kitabxanadır. O, C++ Standart Kitabxanasının bir hissəsidir və geniş çeşiddə konteyner sinifləri, alqoritmlər və iteratorlar təklif edir. STL kitabxanasının əsas elementlərinə aşağıdakılar daxildir:

Konteynerlər: STL vektorlar, siyahılar, yığınlar, növbələr, dəstlər, xəritələr və s. kimi müxtəlif konteyner sinifləri təqdim edir. Bu konteynerlər obyektlərin kolleksiyalarını saxlamağa və idarə etməyə imkan verir.

Alqoritmlər: STL konteynerlərdə və ya hər hansı digər uyğun məlumat strukturlarında işləyə bilən ümumi alqoritmlər toplusunu təmin edir. Bu alqoritmlərə verilənlər üzərində çeşidləmə, axtarış, manipulyasiya və müxtəlif əməliyyatların yerinə yetirilməsi daxildir.

İteratorlar: STL-də iteratorlar konteynerin elementləri arasında keçmək üçün bir yol təqdim edir. Onlar ümumiləşdirilmiş göstərici kimi çıxış edir və konteynerin elementlərinə ardıcıl və ya qeyri-ardıcıl daxil olmaq və manipulyasiya etməyə imkan verir.

Funksiya Obyektləri: STL-də funksiya obyektləri (həmçinin funktorlar kimi tanınır) funksiyalar kimi davranan obyektlərdir. Onlar elementlər üzərində xüsusi davranış və ya əməliyyatları müəyyən etmək üçün alqoritmlərlə istifadə edilə bilər.

Ayırıcılar: STL-dəki ayırıcılar konteynerlər üçün yaddaşı ayırmaq və ayırmaq üçün istifadə olunur. Onlar konteynerlərin yaddaş idarəçiliyinə nəzarət etmək üçün bir yol təqdim edir və xüsusi tələblərə cavab vermək üçün fərdiləşdirilə bilər.

STL kitabxanası kodun təkrar istifadəsini və səmərəliliyini təşviq edərək ümumi və təkrar istifadə edilə bilən komponentləri təmin etmək üçün nəzərdə tutulmuşdur. O, C++ dilində verilənlər strukturları və alqoritmləri ilə işləmək üçün standartlaşdırılmış və səmərəli üsul təklif edir.

28. Vektorlar nədir

C++ dilində vektor STL (Standart Şablon Kitabxanası) tərəfindən təmin edilən dinamik massiv konteyneridir. O, eyni tipli elementlərin ardıcıllığını saxlamağa və manipulyasiya etməyə imkan verir. Vektorlar ölçüsü dəyişdirilə bilən massivləri idarə etmək üçün çevik və səmərəli yol təqdim edir.

Vektorların bəzi əsas xüsusiyyətləri bunlardır:

Dinamik Ölçü: Vektorlar əlavə etdiyiniz və ya sildiyiniz elementlərin sayına əsasən dinamik şəkildə böyüyə və ya kiçilə bilər. Bəyanat zamanı ölçüləri göstərməyə ehtiyac yoxdur.

Random Access: Vektorlar elementlərə səmərəli təsadüfi girişi dəstəkləyir. Siz daimi giriş imkanı verən indeksdən istifadə edərək elementlərə daxil ola bilərsiniz.

Davamlı Yaddaş: Vektordakı elementlər yaddaşın bitişik blokunda saxlanılır, yaddaşa səmərəli girişi və keşdən istifadəni təmin edir.

Dinamik Daxiletmə və Silinmə: Vektorlar vektorun məzmununu dinamik şəkildə dəyişməyə imkan verən istənilən mövqedə elementləri daxil etmək və ya silmək üsullarını təmin edir.

C++ dilində vektordan istifadənin sadə nümunəsi:

```cpp
#include <iostream>
#include <vector>

int main() {
    std::vector<int> numbers;

    // Adding elements to the vector
    numbers.push_back(10);
    numbers.push_back(20);
    numbers.push_back(30);

    // Accessing elements using index
    std::cout << numbers[0] << std::endl;  // Output: 10

    // Iterating over the vector
    for (const auto& num : numbers) {
        std::cout << num << " ";
    }
    // Output: 10 20 30
```

```cpp
    return 0;
}
```

Bu nümunədə tam ədədləri saxlamaq üçün ədədlər adlı vektor yaradırıq. Push_back() metodundan istifadə etməklə elementlər vektora əlavə edilir. Elementlərə subscript operatoru ([]) vasitəsilə daxil olmaq və ya diapazona əsaslanan for loopundan istifadə etməklə təkrarlamaq olar.

29.Vektorlar və siyahılar arasındakı fərq:

Vektorlar və siyahılar C++ dilində STL tərəfindən təmin edilən konteyner siniflərdir, lakin onların xüsusiyyətləri və istifadəsində bəzi fərqlər var:

Saxlama: Vektorlar elementləri bitişik yaddaş blokunda saxlayır, siyahılar isə ikiqat bağlı siyahı strukturundan istifadə edir. Bu o deməkdir ki, vektor elementləri səmərəli təsadüfi girişə imkan verən davamlı yaddaş yığınında saxlanılır, siyahı elementləri isə yaddaşa səpələnir və hər bir elementə daxil olmaq üçün keçid tələb olunur.

Daxiletmə və Silmə: Vektorlar sonunda effektiv daxiletmə və silmə təklif edir (push_back() və pop_back() əməliyyatları), lakin ortada və ya başlanğıcda əlavələr və ya silmələr üçün nisbətən yavaş olur. Siyahılar isə istənilən mövqedə sabit vaxtda səmərəli daxiletmə və silinməni təmin edir.

Təsadüfi giriş: Vektorlar alt işarə operatorundan ([]) və ya iterator arifmetikasından istifadə edərək elementlərə daimi təsadüfi giriş imkanı verir. Siyahılar təsadüfi girişi dəstəkləmir və müəyyən elementə çatmaq üçün əvvəldən və ya sondan ardıcıl keçid tələb edir.

İteratorun etibarlılığı: vektordan elementlər əlavə edildikdə və ya çıxarıldıqda, iteratorlar və elementlərə istinadlar etibarsız ola bilər. Siyahıda iteratorlar və istinadlar daxil edildikdən və ya silindikdən sonra belə etibarlı qalır.

Yaddaş yükü: Vektorların siyahılarla müqayisədə daha kiçik yaddaş yükü var, çünki siyahılar kimi elementləri əlaqələndirmək üçün əlavə göstəricilərə ehtiyac yoxdur.

Vektorlar və siyahılar arasında seçim sizin xüsusi tələblərinizdən asılıdır. Əgər tez-tez təsadüfi giriş və elementlərin səmərəli saxlanmasına ehtiyacınız varsa, vektorlar daha yaxşı seçimdir. Əgər ixtiyari mövqelərdə səmərəli daxil etmə və silməyə ehtiyacınız varsa və ya iteratorun etibarlılığı narahatlıq doğurursa, siyahılar daha uyğun ola bilər.

30.Bir yığın nədir:

Kompüter elmində yığın LIFO (Last-In, First-Out) prinsipinə əməl edən mücərrəd məlumat növüdür. O, özünü obyektlərin fiziki yığını kimi aparır, burada yalnız yuxarıdan elementlər əlavə edə və ya silə bilərsiniz.

Proqramlaşdırma kontekstində yığın iki əsas əməliyyatı dəstəkləyən məlumat strukturudur:

Push: yığının yuxarı hissəsinə element əlavə edir.

Pop: Ən üst elementi yığından çıxarır.

Bundan əlavə, yığın nəzər salmaq (yuxarı elementə onu çıxarmadan daxil olmaq) və yığının boş olub olmadığını yoxlamaq kimi digər əməliyyatları da təmin edə bilər.

Yığınlar tez-tez dərinlikdən əvvəl axtarış, geri izləmə, ifadələrin təhlili, funksiya çağırış yığınının idarə edilməsi və s. ilə bağlı problemləri həll etmək üçün istifadə olunur. Onlar massivlər və ya əlaqəli siyahılar kimi müxtəlif məlumat strukturlarından istifadə etməklə həyata keçirilə bilər.

STL yığın konteynerindən istifadə edərək C++ dilində yığından istifadənin sadə nümunəsi:

```cpp
#include <iostream>
#include <stack>

int main() {
    std::stack<int> stack;

    // Pushing elements onto the stack
    stack.push(10);
    stack.push(20);
    stack.push(30);

    // Accessing and removing the top element
    int topElement = stack.top();  // Accessing the top element
    stack.pop();                   // Removing the top element

    std::cout << "Top element: " << topElement << std::endl;
    std::cout << "Stack size: " << stack.size() << std::endl;

    return 0;
}
```

Bu misalda tam ədədləri saxlamaq üçün stack adlı yığın yaradırıq. Elementlər istifadə edərək yığına əlavə olunur push() metodu və üst elementə top() metodundan istifadə etməklə daxil olmaq olar. pop() metodu üst elementi yığından çıxarır. size() metodu yığındakı elementlərin sayını qaytarır.

Yığınlar LIFO prinsipinə uyğun olaraq elementləri idarə etmək üçün səmərəli üsul təqdim edərək onları geniş tətbiqlərdə faydalı edir.

31. Növbə nədir?

C++ dilində növbə First-In-First-Out (FIFO) prinsipinə əməl edən məlumat strukturudur. O, yeni elementlərin arxaya daxil edildiyi (həmçinin "arxa" kimi tanınır) və mövcud elementlərin öndən çıxarıldığı elementlər toplusunu təmsil edir. Bu o deməkdir ki, ən uzun növbədə olan element çıxarılan ilk elementdir.

Növbə məlumat strukturu iki əsas əməliyyatı təmin edir:

1.Quraşdırma: O, növbənin arxasına element əlavə edir. Bu əməliyyat həm də "push" və ya "insert" adlanır.

2.Dequeue: Elementi növbənin ön hissəsindən çıxarır. Bu əməliyyat "pop" və ya "silmək" kimi də tanınır.

C++, növbə məlumat strukturunu həyata keçirən std::queue adlı standart şablon kitabxanası (STL) konteynerini təmin edir. Onu istifadə etmək üçün <queue> başlıq faylını daxil etməlisiniz.

32. C++ dilində sinif və funksiya dəyişənlərinin həyat müddəti

C++-da dəyişənlərin həyat müddəti onların harada elan olunduğundan və əhatə dairəsindən asılıdır. Sinif və ya funksiya daxilində elan edilən dəyişənlərin müxtəlif ömürləri var:

1. Sinif Üzv Dəyişənləri: Bu dəyişənlər sinif daxilində elan edilir və həmin sinfin obyektləri ilə əlaqələndirilir. Obyekt mövcud olduğu müddətcə onlar mövcuddur. Obyekt yaradıldıqda onun üzv dəyişənləri üçün yaddaş ayrılır, obyekt məhv edildikdə isə yaddaş buraxılır. Sinif üzvü dəyişənlərinin ömrü obyektin ömrü ilə bağlıdır.

2. Yerli Dəyişənlər: Bu dəyişənlər funksiya və ya kod bloku daxilində elan edilir və yalnız həmin funksiya və ya blok daxilində əlçatandır. Lokal dəyişənlər funksiya və ya blok daxil edildikdə yaranır və funksiya və ya blokdan çıxdıqda məhv edilir. Lokal dəyişənlərin ömrü onların elan olunduğu əhatə dairəsi ilə məhdudlaşır. Onlar adətən yığında saxlanılır.

3. Statik dəyişənlər: Statik dəyişənlər `static açar sözü ilə elan edilir. Funksiya daxilində statik dəyişən funksiya çağırışları arasında öz dəyərini saxlayır. Statik dəyişənlərin işə salındığı andan proqramın dayandırılmasına qədər proqramın icrası boyu uzanan ömür müddəti var. Onlar adətən ayrıca statik məlumat sahəsində saxlanılır.

4. Qlobal dəyişənlər: Qlobal dəyişənlər hər hansı funksiya və ya sinifdən kənarda, adətən faylın əvvəlində elan edilir. Onların statik dəyişənlərə bənzər proqramın icrası boyu uzanan ömürləri var. Qlobal dəyişənlərə proqramın istənilən hissəsindən daxil olmaq mümkündür və onların ömrü proqram başa çatdıqda başa çatır. Onlar adətən ayrıca qlobal məlumat zonasında saxlanılır.

Qeyd etmək vacibdir ki, dəyişənlərin işə salınması və məhv edilməsi onların növlərindən və obyektlər və ya primitiv tiplərdən asılı olaraq müvafiq olaraq konstruktorlar və dağıdıcıları da əhatə edə bilər.

Dəyişənlərin ömrünü başa düşmək yaddaşı idarə etmək və dəyişənlərin əhatə dairəsindən kənara çıxdıqdan sonra onlara daxil olmaq və ya işə salınmamış dəyişənlərdən istifadə etmək kimi potensial problemlərin qarşısını almaq üçün çox vacibdir.

33. Sinfin statik komponentləri

C++-da sinfin statik komponentləri həmin sinfin bütün obyektləri arasında paylaşılır. Onlar fərdi obyektlərlə deyil, sinfin özü ilə əlaqələndirilir. Statik komponentlərə statik üzv dəyişənləri və statik üzv funksiyaları daxildir.

1.Statik Üzv Dəyişənləri: Bunlar sinif daxilində static açar sözü ilə elan edilən dəyişənlərdir. Onlar sinfin bütün obyektləri tərəfindən paylaşılır və bütün nümunələrdə eyni dəyərə malikdir. Statik üzv dəyişənləri adətən sinif üzrə məlumatı və ya paylaşılan məlumatları saxlamaq üçün istifadə olunur. Onlar hər hansı bir şeydən əvvəl bir dəfə işə salınır

sinfin obyekti yaradılır və onların yaddaşı ayrıca statik verilənlər sahəsinə ayrılır. Statik üzv dəyişənlərinə sinif adından sonra əhatə dairəsinin həlli operatoru (::) və ya sinif daxilində sinif adının özündən istifadə etməklə daxil olmaq olar.

2.Statik Üzv Funksiyaları: Bunlar sinif daxilində static açar sözü ilə elan edilən funksiyalardır. Statik üzv dəyişənləri kimi, statik üzv funksiyaları da fərdi obyektlərə deyil, sinfə aiddir. Onları sinif adından sonra əhatə dairəsinin həlli operatorundan (::) istifadə etməklə çağırmaq olar. Statik üzv funksiyaların statik olmayan üzv dəyişənlərinə və ya sinfin funksiyalarına çıxışı yoxdur, çünki onlar konkret obyektdə fəaliyyət göstərmirlər. Onlar adətən obyekt-xüsusi məlumatlardan asılı olmayan kommunal funksiyalar və ya əməliyyatlar üçün istifadə olunur.

Statik üzv dəyişənlərə və funksiyalara sinif nümunəsi yaratmadan daxil olmaq və istifadə etmək olar. Onlar adətən sinif nümunələrinin sayılması, obyektlər arasında ümumi funksionallığın təmin edilməsi və ya paylaşılan resursların saxlanması kimi tapşırıqlar üçün istifadə olunur.

34. Statik sinif dəyişənləri

C++ dilində statik sinif dəyişənləri sinfin fərdi obyektlərinə deyil, sinfin özünə aid olan statik üzv dəyişənlərdir. Onlar sinfin bütün obyektləri arasında paylaşılır və bütün nümunələr üçün eyni dəyərə malikdir. Statik sinif dəyişənləri sinif tərifi daxilində statik açar sözlə elan edilir. Statik sinif dəyişənləri adətən sinifin bütün nümunələri arasında paylaşılan məlumat və ya vəziyyəti saxlamaq lazım olduqda istifadə olunur. Onlara sinfin hər hansı xüsusi obyektindən asılı olmayaraq daxil olmaq və dəyişdirmək olar və statik sinif dəyişəninə edilən dəyişikliklər bütün instansiyalarda əks olunur.

35. Statik və qeyri-statik komponentlər

C++ dilində statik və qeyri-statik komponentlər sinif daxilində fərqli xüsusiyyətlərə və istifadə ssenarilərinə malikdir. Statik və qeyri-statik komponentlər arasında müqayisə:

1. Statik komponentlər:

- Statik Üzv Dəyişənləri: Statik üzv dəyişənləri fərdi obyektlərə deyil, sinfin özünə aiddir. Onlar sinfin bütün nümunələri arasında paylaşılır və bütün obyektlər arasında eyni dəyərə malikdir. Onlar statìc açar sözü ilə elan edilir. Statik üzv dəyişənləri adətən sinif üzrə məlumat və ya paylaşılan resurslar üçün istifadə olunur.

- Statik Üzv Funksiyaları: Statik üzv funksiyaları xüsusi obyektlərlə deyil, siniflə əlaqələndirilir. Onları sinif nümunəsi yaratmadan çağırmaq olar. Statik üzv funksiyaların qeyri-statik üzv dəyişənlərə və ya funksiyalara çıxışı yoxdur, çünki onlar konkret obyektdə fəaliyyət göstərmirlər. Onlar statìc açar sözü ilə elan edilir və tez-tez kommunal funksiyalar və ya obyektə aid məlumatlardan asılı olmayan əməliyyatlar üçün istifadə olunur.

2. Statik olmayan komponentlər:

- Qeyri-Statik Üzv Dəyişənləri: Qeyri-statik üzv dəyişənlər sinfin fərdi obyektləri ilə əlaqələndirilir. Hər bir obyektin statik olmayan üzv dəyişənlərinin öz nüsxəsi var və onların dəyərləri müxtəlif nümunələrdə dəyişə bilər. Qeyri-statik üzv dəyişənlər static` açar sözü olmadan elan edilir. Onlar obyektə məxsus məlumatları və ya vəziyyəti təmsil etmək üçün istifadə olunur.

- Qeyri-Statik Üzv Funksiyaları: Qeyri-statik üzv funksiyaları sinfin xüsusi obyektlərində işləyir. Onların qeyri-statik üzv dəyişənlərə girişi var və ayrı-ayrı obyektlərin vəziyyətini dəyişdirə bilər. Qeyri-statik üzv funksiyaları `static` açar sözü olmadan elan edilir və obyektlərə xas olan davranış və əməliyyatları əhatə etmək üçün istifadə olunur.

Statik və qeyri-statik komponentlər arasındakı əsas fərqlər:

- Statik komponentlər sinfin bütün nümunələri arasında paylaşılır, qeyri-statik komponentlər isə fərdi obyektlərə xasdır.

- Statik üzv dəyişənlərinin bütün sinif üçün bir nümunəsi var, qeyri-statik üzv dəyişənlərin isə hər bir obyekt üçün ayrıca nümunəsi var.

- Statik üzv funksiyaların statik olmayan üzv dəyişənlərə və ya funksiyalara çıxışı yoxdur, qeyri-statik üzv funksiyalar isə həm statik, həm də qeyri-statik üzvlərə daxil ola bilir.

- Statik komponentlərə sinif obyekti yaratmadan daxil olmaq olar, qeyri-statik komponentlərə isə obyekt nümunələri vasitəsilə daxil olmaq.

Statik və ya qeyri-statik komponentlərdən istifadə etmək qərarına gəldikdə, təmsil etmək istədiyiniz məlumatın və ya davranışın paylaşılan xarakterini nəzərə alın. Məlumat və ya davranış bütün obyektlər arasında ardıcıl olmalıdırsa, statik komponentlər uyğun ola bilər. Digər tərəfdən, məlumat və ya davranış fərdi obyektlərə xasdırsa, qeyri-statik komponentlərdən istifadə edilməlidir.

36. Obyektin dəyərinə görə ötürülməsi

C++ dilində obyekti qiymətə görə ötürərkən obyektin surəti hazırlanır və funksiyaya və ya metoda ötürülür. Funksiya daxilində obyektə edilən hər hansı dəyişiklik funksiyadan kənar orijinal obyektə təsir etməyəcək. Obyektin dəyərinə görə ötürülməsi aşağıdakı addımları əhatə edir:

1.Bütün üzv dəyişənlər və onların qiymətləri daxil olmaqla obyektin yeni nüsxəsi yaradılır.

2.Obyektin surəti funksiyaya parametr kimi ötürülür.

3.Funksiya daxilində obyektin surəti dəyişdirilir və ya lazım olduqda istifadə olunur.

4.Funksiya daxilində obyektə edilən hər hansı dəyişiklik həmin funksiya üçün lokaldır və orijinal obyektə təsir göstərmir.

37. Obyektin istinadla ötürülməsi

C++ dilində obyekti istinadla ötürərkən, orijinal obyektə istinad funksiyaya və ya metoda ötürülür. Funksiya daxilində obyektə edilən hər hansı dəyişiklik funksiyadan kənar orijinal obyektə təsir edəcək. Obyektin istinad yolu ilə ötürülməsi aşağıdakı addımları əhatə edir:

1.Funksiya parametrləri siyahısında orijinal obyektə istinad edən istinad dəyişəni yaradılır.

2.Funksiya çağırıldıqda istinad orijinal obyektə bağlanır.

3.Funksiya daxilində istinad orijinal obyektə daxil olmaq və onu dəyişdirmək üçün istifadə edilə bilər.

4.Funksiya daxilində obyektə edilən hər hansı dəyişiklik birbaşa funksiyadan kənar orijinal obyektə təsir edəcək.

38. Dostluq dərsləri

C++ dilində dost sinfi başqa sinfin özəl və qorunan üzvlərinə çıxışı olan sinifdir. Bu, xüsusi siniflərə xüsusi giriş imtiyazları vermək üsuludur və onlara sinfin şəxsi üzvlərini özlərininki kimi manipulyasiya etməyə imkan verir. Dost sinifləri sinif tərifi daxilində dost açar sözündən istifadə edilməklə elan edilir. Dost sinifləri inkapsulyasiyanı və məlumatların gizlədilməsini təmin etməklə yanaşı, müəyyən siniflərə xüsusi giriş təmin etməli olduğunuz müəyyən ssenarilərdə faydalı ola bilər. Bununla belə, dost siniflərindən ehtiyatla istifadə etmək vacibdir, çünki həddindən artıq girişin verilməsi obyekt yönümlü proqramlaşdırmanın inkapsulyasiya və məlumat gizlətmə prinsiplərini poza bilər.

39. Git versiyasına nəzarət sistemi və onun istifadəsi

Git mənbə kodunu idarə etmək və proqram təminatının hazırlanması layihələrində, o cümlədən C++ dilində yazılmış dəyişiklikləri izləmək üçün geniş istifadə olunan paylanmış versiya idarəetmə sistemidir. Git kod anbarlarında əməkdaşlığa, versiyaya, budaqlanmaya, birləşməyə və dəyişiklikləri izləməyə imkan verən bir sıra əmrlər və funksiyalar təqdim edir.

Git-in C++ inkişafında necə istifadə edildiyinə dair ümumi məlumat:

1. Git Repozitoriyasının işə salınması: Git-dən C++ layihəsində istifadə etməyə başlamaq üçün siz adətən layihənizin kök kataloqunda Git repozitoriyasını işə salırsınız. Bu, terminalda `git init əmrini işlətməklə və ya Git müştərisindən istifadə etməklə həyata keçirilir.

2. Faylların İzlənməsi: Git sizə layihə fayllarınızdakı dəyişiklikləri izləməyə imkan verir. `git adȁ əmrindən istifadə edərək Git deposuna xüsusi faylları və ya bütün qovluqları əlavə edə bilərsiniz. Məsələn, "main.cpp" adlı C++ faylını əlavə etmək üçün `git add main.cpȍ əmrindən istifadə edərdiniz.

3. Dəyişikliklərin həyata keçirilməsi: Fayllar əlavə edildikdən sonra dəyişikliklərin şəklini saxlamaq üçün öhdəlik yarada bilərsiniz. Öhdəliklər layihə tarixində mərhələlər kimi çıxış edir. Siz öhdəlik yaratmaq üçün commit mesajı ilə birlikdə `git commit` əmrindən istifadə edirsiniz. Məsələn, `git commit -m "Əlavə edilmiş main.cpp faylı"`.

4. Budaqlanma və Birləşmə: Git əsas kod bazasına təsir etmədən yeni funksiyalar, səhvlərin düzəldilməsi və ya təcrübələr üzərində işləmək üçün filiallar yaratmağa imkan verir. Siz `git branch` istifadə edərək filial yarada və `git checkout` istifadə edərək ona keçə bilərsiniz. Filialdakı dəyişiklikləri bitirdikdən sonra onu `git mergè istifadə edərək yenidən əsas filiala birləşdirə bilərsiniz.

5. Əməkdaşlıq və Uzaq Repozitoriyalar: Git sizə GitHub, GitLab və ya Bitbucket kimi xidmətlərdə yerləşdirilən uzaq depolara qoşulmağa imkan verməklə əməkdaşlığı dəstəkləyir. Siz `git push` istifadə edərək yerli öhdəliklərinizi uzaq depoya köçürə, `git fetch` istifadə edərək uzaq depodan ən son dəyişiklikləri əldə edə və `git mergè və ya `git puȉl istifadə edərək həmin dəyişiklikləri yerli filialınıza birləşdirə bilərsiniz.

6. Versiyaların idarə edilməsi: Git versiyaları idarə etmək və layihə tarixçəsi arasında hərəkət etmək üçün güclü alətlər təqdim edir. Siz icra jurnalına baxa, müxtəlif versiyaları müqayisə edə, dəyişiklikləri geri qaytara və xüsusi versiyaları və ya buraxılışları qeyd etmək üçün teqlər yarada bilərsiniz.

Bunlar Git-də əsas anlayış və əmrlərdən yalnız bir neçəsidir. Git, C++ inkişaf layihəsinin ehtiyaclarına uyğunlaşdırıla bilən geniş xüsusiyyətlər və iş axınları təklif edir. Git və onun müxtəlif əmrlərini öyrənmək əməkdaşlığı əhəmiyyətli dərəcədə artıra, effektiv versiya nəzarətini təmin edə və C++ layihələri üçün dəyişikliklərin tarixini təqdim edə bilər.

40.Git versiyasına nəzarət sistemləri əmrləri

Git versiyasına nəzarət sistemi əmrləri C++ daxil olmaqla heç bir proqramlaşdırma dilinə xas deyil. Bununla belə, mən sizə C++ layihələrinin idarə edilməsi kontekstində istifadə olunan çox istifadə olunan Git əmrlərinin siyahısını təqdim edə bilərəm. Bu əmrlər komanda xəttində və ya Git müştəriləri vasitəsilə yerinə yetirilə bilər:

1. `git inît : Cari kataloqda yeni Git repozitoriyasını işə salır.

2. `git clone <repository_url>`: Yerli maşınınızda uzaq Git repozitoriyasının surətini yaradır.

3. `git add <file> : Səhnə sahəsinə fayl və ya dəyişikliklər əlavə edir, onları növbəti öhdəliyə hazırlayır. Məsələn, `git add main.cpp .

4. `git commit -m "<commit_message>"`: Təhlükə mesajı ilə birlikdə səhnələşdirmə sahəsindəki dəyişikliklərlə yeni öhdəlik yaradır. Məsələn, git commit -m "Əlavə edilmiş main.cpp faylı" .

5. `git statusu`: Dəyişdirilmiş fayllar və izlənilməmiş fayllar daxil olmaqla, deponun cari vəziyyətini göstərir.

6. `git filiàl : Repozitoriyadakı bütün filialları sadalayır. Cari filial ulduz işarəsi ilə göstərilir.

7. `git checkout <branch_name>`: Göstərilən filiala keçir. Məsələn, `git checkout featurebranch`.

8. `git merge <branch_name>`: Göstərilən filialı cari filiala birləşdirir. Məsələn, `git birləşən xüsusiyyət şöbəsi`.

9. `git push`: Yerli öhdəliyi uzaq depoya itələyir. Məsələn, git push origin masteṙ yerli öhdəlikləri "master" filialındakı "mənşə" adlı uzaq depoya itələyir.

10. git puỉl : Uzaq depodan ən son dəyişiklikləri gətirir və onları cari filialda birləşdirir.

11. git fetch`: Uzaq depodan ən son dəyişiklikləri cari filiala birləşdirmədən götürür.

12. git loġ : Göndərmə mesajları, müəlliflər, tarixlər və icra heşləri daxil olmaqla, icra tarixini göstərir.

13. git diff`: Cari kod və son təsdiq edilmiş versiya arasındakı fərqləri göstərir.

14. git tag`: Xüsusi versiyaları və ya buraxılışları qeyd edərək, depodakı bütün teqləri siyahıya alır.

Bunlar C++ layihələrində istifadə olunan ümumi Git əmrlərindən bəzilərdir. Hər bir əmr və onların müxtəlif variantları və istifadə halları haqqında daha ətraflı məlumat üçün Git sənədlərinə və ya onlayn resurslara müraciət etmək tövsiyə olunur.

41.Massivin verilmiş indeksinə yeni element əlavə etmək kodunu yazın (massiv ölçüsü artacaq)

 # daxil <iostream>

ad sahəsi std istifadə edərək;

```
int main() {
  int arr[100] = {1, 2, 3, 4, 5};
  int n = 5;
  int indeksi = 2;
  int dəyəri = 10; //ortaya salınacaq rəqəm

  //elementlərin sağa çəkilməsi
  for(int i = n-1; i >= index; i--) {
    arr[i+1] = arr[i];
  }

  // yeni rəqəmin qoyulması
  arr[indeks] = dəyər;
  n++; // arrayi böyütməsi
```

```
  for(int i = 0; i < n; i++) {
    cout << arr[i] << " ";
  }
  cout << endl;
  0 qaytarmaq;
}
```

---

42. Strukturlarla Əlaqəli Siyahının yaradılması kodunu yazın

```
# daxil <iostream>
ad sahəsi std istifadə edərək;

struktur qovşağı {
  int məlumatları;
  node* növbəti;
};

int main() {
  Node* baş = NULL;
  Node* cari = NULL;

  // ilk node yaradın və onu siyahının başı olaraq təyin edin
  baş = yeni Node;
  baş->data = 1;
  head->next = NULL;
  cari = baş;

  // ikinci node yaradın və onun siyahısının sonuna əlavə edin
```

```
  cari->növbəti = yeni Node;
  cari = cari->növbəti;
  cari->data = 2;
  cari->növbəti = NULL;

  // üçüncü node yaradın və onu siyahının sonuna əlavə edin
  cari->növbəti = yeni Node;
  cari = cari->növbəti;
  cari->data = 3;
  cari->növbəti = NULL;

  // siyahını çap edin
  cari = baş;
  while(cari != NULL) {
    cout << cari->data << " ";
    cari = cari->növbəti;
  }
  cout << endl;

  cari = baş;
  while(cari != NULL) {
    Node* temp = cari;
    cari = cari->növbəti;
    tempi silmək;
  }
  0 qaytarmaq;
}
```

43.C++ dilində inkapsullaşdırılmış sinif yaratmaq kodunu yazın

```
# daxil <iostream>

ad sahəsi std istifadə edərək;


sinif adamı {
  özəl:
    sətir adı;
    int yaş;
  ictimai:
    etibarsız dəst adı (sətir n) {
      ad = n;
    }
    void setAge(int a) {
      yaş = a;
    }
    string getName() {
      qayıt adı;
    }
    int getAge() {
      qayıtma yaşı;
    }
};


int main() {
  şəxs p1; // Şəxs sinifinin nümunəsini yaradın
  p1.setName("Vaqkus"); // adı "istədiyiniz hər hansı bir ad" olaraq təyin edin
  p1.setAge(33); // yaşı 30-a təyin edin
  cout << "Ad: " << p1.getName() << endl; // adını çap edin
  cout << "Yaş: " << p1.getAge() << endl; // çap yaşı
  0 qaytarmaq;
}
```

44.Subclass yaradılması kodunu yazın və C++ daxilində super sinif üzvlərinə daxil olun

 # daxil <iostream>

ad sahəsi std istifadə edərək;

// superclass təyinin

sinif SuperClass {

ictimai:

int x; // Üzv dəyişəni

SuperClass(int x) : x(x) {} // Konstruktor

void printX() { // Üzv funksiyası

cout << "x = " << x << endl;

}

};

// Superclassdan miras qalan subclasssi müəyyənin

sinif Alt Sinif: ictimai SuperClass {

ictimai:

SubClass(int x) : SuperClass(x) {} // Konstruktor

void printXPlusOne() { // Üzv funksiyası

cout << "x + 1 = " << x + 1 << endl; // irsi üzv dəyişən daxil olun

}

};

int main() {

SubClass sc(5); //yarımsınıfın bir nümunəsini yaradın

sc.printX(); // irsi üzv funksiya cağirin

sc.printXPlusOne(); // altsınıf üzvü funksiyası cağirin

0 qaytarmaq;

}

45.Sınıf obyektlərindən polimorfizm yaratmaq kodunu yazın və C++ dilində casting operatorundan istifadə edin.

 # daxil <iostream>

ad sahəsi std istifadə edərək;

// əsas sinif təyinin

sinif bazası {

ictimai:

virtual boş çap () { // Virtual funksiya

cout << "Baza" << endl;

}

};

// törəmə sinif təyinin

əldə edilən sinif: ictimai baza {

ictimai:

void print() override { // virtual funksiyanın ləğv edilməsi

cout << "Törəmə" << endl;

}

};

int main() {

Baza* b = yeni törəmə(); // törəmə obyekt ucun göstərici yaradin

b->print(); // Virtual funksiyaya zəng edin (“Təxsis edilmiş”i çap edir)

static_cast<Derived*>(b)->print(); //tökmə operatoru islederek törəmə sinif funksiyası caqirilmasi ("Tərmə" çapı)

sil b; // yaddasin temizlenmesi

0 qaytarmaq;

```
}
```

46. Sinif funksiyasının əvəzlənməsinin yaradılması kodunu yazın

```
# daxil <iostream>
ad sahəsi std istifadə edərək;

// əsas sinif təyinin
sinif bazası {
ictimai:
   virtual boş çap () { // Virtual funksiya
      cout << "Baza" << endl;
   }
};

// törəmə sinif təyinin
əldə edilən sinif: ictimai baza {
ictimai:
   void print() override { // Virtual funksiyanı ləğv edin
      cout << "Törəmə" << endl;
   }
};

int main() {
   Baza* b = yeni törəmə(); // törəmə obyekt ucun göstərici yaradin
   b->print(); // Virtual funksiyaya zəng edin (“Təxsis edilmiş”i çap edir)
   sil b; // yaddasin temizlenmesi
   0 qaytarmaq;
}
```

47. Funksiyalar üçün qiymət və istinadla keçmək üçün nümunə kodunu yazın

```
# daxil <iostream>

ad sahəsi std istifadə edərək;


// Qiymet(value) üzrə arqument alan funksiyası

Void byValue(int x) {

   x = 10; // x-in yerli surətini dəyişdirin

}


// İstinad(reference) üzrə arqument alan funksiyası

Referansla etibarsız (int& x) {

   x = 10; // Orijinal x-i dəyişdirin

}


int main() {

   int a = 5;

   int b = 5;


   byValue(a); // Funksiyaya a-nı qiymet (value) kimi ötürün

   cout << "a = " << a << endl; // "a = 5" çap edir


   istinadla(b); // Funksiyaya b-ni istinad (reference) kimi ötürün

   cout << "b = " << b << endl; // "b = 10" çap edir


   0 qaytarmaq;

}
```

48.C++ dilində Dostluq yaratmaq funksiyasının kodunu yazın

```
# daxil <iostream>
ad sahəsi std istifadə edərək;

sinif MyClass {
özəl:
   int x;
ictimai:
   MyClass(int x) : x(x) {}
   dost void printX(MyClass& mc); // dost funksiyası təyinin
};

// dost funksiyasını təyin edin
void printX(MyClass & mc) {
   cout << "x = " << mc.x << endl; // Şəxsi üzv dəyişəninə daxil olun
}

int main() {
   MyClass mc(5);
   printX(mc); // Dosta zəng funksiyası ("x = 5" çap edir)
   0 qaytarmaq;
}
```

---

49. Konstruktorların müxtəlif üsullarla surətinin çıxarılmasının kodunu yazın

C++-da konstruktorların kopyalanması:

C++-da konstruktorları kopyalamağın iki ümumi yolu var: standart surət konstruktoru və istifadəçi tərəfindən təyin edilmiş surət konstruktoru.

a) Defolt Kopya Konstruktoru:

```
sinif MyClass {
ictimai:
```

```
    int məlumatları;


    MyClass(const MyClass və digər) {
        data = other.data;
    }
};
```

b) İstifadəçi tərəfindən müəyyən edilmiş Kopya Konstruktoru:

```
sinif MyClass {
ictimai:
    int məlumatları;


    MyClass(int dəyər) : data(dəyər) {}


    MyClass(const MyClass və digər) {
        data = other.data;
    }
};
```

---

50. Məhvedicilərdən istifadə kodunu yazın

```
# daxil <iostream>


sinif MyClass {
ictimai:
    Mənim sinfim() {
        std::cout << "Konstruktor çağırıldı." << std::endl;
    }
```

```
    ~MyClass() {
        std::cout << "Destruktor çağırıldı." << std::endl;
    }
};

int main() {
    MyClass obyekti; // MyClass obyektini yaradın

    // Proqramın qalan hissəsi

    0 qaytarmaq;
}
```

Yuxarıdakı kodda konstruktor və dağıdıcı ilə MyClass sinifini təyin edirik. Əsas funksiyada MyClass-ın obyekt obyektini yaradırıq. Obj obyekti əhatə dairəsindən kənara çıxdıqda (yəni, əsas funksiyanın sonunda) MyClass-ın destruktoru avtomatik olaraq çağırılır.

51.Staka element əlavə etmək və silmək üçün C++ kodunu yazın

```
# daxil <iostream>
# daxil <yığın>

int main() {
    std::stack<int> myStack;

    // Yığına elementlərin əlavə edilməsi
    myStack.push(10);
    myStack.push(20);
    myStack.push(30);

    // Yığından elementlərin çıxarılması
    isə (!myStack.empty()) {
```

```
    int element = myStack.top();
    myStack.pop();
    std::cout << "Poplanmış element: " << element << std::endl;
  }

  0 qaytarmaq;
}
```

52.Queue elementi əlavə etmək və silmək üçün C++ kodunu yazın

```
# daxil <iostream>
# daxil <növbə>

int main() {
  std::queue<int> myQueue;

  // Növbəyə elementlərin əlavə edilməsi
  myQueue.push(10);
  myQueue.push(20);
  myQueue.push(30);

  // Elementlərin növbədən çıxarılması
  isə (!myQueue.empty()) {
    int element = myQueue.front();
    myQueue.pop();
    std::cout << "Silinmiş element: " << element << std::endl;
  }

  0 qaytarmaq;
```

```
}
```

---

53. Sıfır istisnaya bölmə ilə işləmək üçün C++ kodunu yazın

```
# daxil <iostream>
# daxil <stdexcept>

int bölmə (int a, int b) {
    əgər (b == 0) {
        throw std::runtime_error("Sıfıra bölün!");
    }

    a / b qaytarın;
}

int main() {
    cəhd {
        int nəticə = bölmək (10, 0);
        std::cout << "Nəticə: " << nəticə << std::endl;
    } tutmaq (const std::runtime_error & xəta) {
        std::cout << "İstisna tutuldu: " << error.what() << std::endl;
    }

    0 qaytarmaq;
}
```

---

54. Statik olmayan üzvdən statik üzvə qoşulmaq üçün C++ kodunu yazın

```
# daxil <iostream>

sinif MyClass {
ictimai:
   statik int staticMember;
   int nonStaticMember;

   void printMembers() {
      std::cout << "Statik Üzv: " << staticMember << std::endl;
      std::cout << "Qeyri-Statik Üzv: " << nonStaticMember << std::endl;
   }
};

int MyClass::staticMember = 10;

int main() {
   MyClass obyekti;
   obj.nonStaticMember = 20;
   obj.printMembers();

   0 qaytarmaq;
}
```

Yuxarıdakı kodda biz statik üzvü staticMember və qeyri-statik üzvü nonStaticMember olan MyClass sinfini təyin edirik. Statik üzvə sinif adından istifadə etməklə, qeyri-statik üzvə isə obyekt nümunəsindən istifadə etməklə daxil olur.

55.Siniflərin bütün obyektlərinin sayını tapmaq üçün C++ kodunu yazın

```
# daxil <iostream>
```

```
sinif MyClass1 {
   statik int sayı;

ictimai:
   MyClass1() {
      count++;
   }

   ~MyClass1() {
      saymaq --;
   }

   statik int getCount() {
      geri sayı;
   }
};

int MyClass1::count = 0;

sinif MyClass2 {
   statik int sayı;

ictimai:
   MyClass2() {
      count++;
   }
```

```
    ~MyClass2() {
        saymaq --;
    }

    statik int getCount() {
        geri sayı;
    }
};

int MyClass2::count = 0;

int main() {
    MyClass1 obj1_1;
    MyClass1 obj1_2;
    MyClass2 obj2_1;
    MyClass2 obj2_2;

    std::cout << "MyClass1 obyektlərinin sayı: " << MyClass1::getCount() <<
    std::endl;
    std::cout << "MyClass2 obyektlərinin sayı: " << MyClass2::getCount() <<
    std::endl;

    0 qaytarmaq;
}
```

56.Lokal depoya fayl əlavə etmək üçün git əmrlərini yazın

Yerli Git deposuna C++ faylları əlavə etmək üçün aşağıdakı Git əmrlərindən istifadə edə bilərsiniz:

1.Yeni Git repozitoriyasını işə salın (əgər artıq işə salınmayıbsa):

git init (charsp)

2. Hansı faylların hələ izlənilmədiyini görmək üçün deponun statusunu yoxlayın:

git statusu (lua)

3. Hazırlama sahəsinə tək bir C++ faylı əlavə edin:

git əlavə edin <filename.cpp> (csharp)

<filename.cpp>-ni əlavə etmək istədiyiniz C++ faylının adı ilə əvəz edin. Məsələn, git add main.cpp.

4. Hazırlama sahəsinə birdən çox C++ faylı əlavə edin:

git əlavə et <file1.cpp> <file2.cpp> <file3.cpp> ...(php)

Əvəz edin**<file1.cpp>, <file2.cpp>, <file3.cpp>,**və s. əlavə etmək istədiyiniz C++ faylларının adları ilə. Misal üçün,**git əlavə et file1.cpp file2.cpp file3.cpp.**

5. Bütün dəyişdirilmiş və yeni C++ faylларını səhnələşdirmə sahəsinə əlavə edin:

git əlavə et.(csharp)

The . simvol cari kataloqu təmsil edir.

6. Vəziyyəti yenidən yoxlamaqla etdiyiniz dəyişikliklərin düzgün tərtib edildiyini yoxlayın:

git statusu (lua)

C++ faylларını səhnələşdirmə sahəsinə əlavə etdikdən sonra git commit əmrindən istifadə edərək dəyişiklikləri yerinə yetirməyə davam edə bilərsiniz.

57.Uzaq depoya faylların əlavə edilməsi üçün git əmrlərini yazın

To add C++ files to a remote Git repository, you can use the following Git commands:

1. Clone the remote repository to your local machine (if not already cloned):

```bash
git clone <remote_repository_url>
```

Replace `<remote_repository_url>` with the URL of the remote Git repository. For example, `git clone https://github.com/username/repo.git`.

2. Navigate to the cloned repository directory:

```bash
cd <repository_directory>
```

Replace `<repository_directory>` with the name of the directory where the repository was cloned.

3. Check the status of the repository to see which files are not yet tracked:

3. Check the status of the repository to see which files are not yet tracked:

```lua
git status
```

4. Add a single C++ file to the staging area:

```csharp
git add <filename.cpp>
```

Replace `<filename.cpp>` with the name of the C++ file you want to add. For example, `git add main.cpp`.

5. Add multiple C++ files to the staging area at once:

```php
git add <file1.cpp> <file2.cpp> <file3.cpp> ...
```

Replace `<file1.cpp>`, `<file2.cpp>`, `<file3.cpp>`, and so on with the names of the C++ files you want to add. For example, `git add file1.cpp file2.cpp file3.cpp`.

6. Add all modified and new C++ files to the staging area:

csharp Copy code

```
git add .
```

The `.` symbol represents the current directory.

7. Verify the changes you've made are staged correctly by checking the status again:

lua Copy code

```
git status
```

8. Commit the changes:

sql Copy code

```
git commit -m "Add C++ files"
```

8. Commit the changes:

sql Copy code

```
git commit -m "Add C++ files"
```

Replace the commit message `"Add C++ files"` with an appropriate message describing the changes.

9. Push the changes to the remote repository:

perl Copy code

```
git push origin <branch_name>
```

Replace `<branch_name>` with the name of the branch where you want to push the changes. For example, `git push origin main`.

Note: Before pushing the changes, ensure that you have the necessary permissions to write to the remote repository. If you do not have write access, you may need to fork the repository or request access from the repository owner.

58.Uzaq depodan fayl yükləmək üçün git əmrlərini yazın

To download a file from a remote repository using Git, you need to clone the repository. Here are the steps to clone a remote repository and download a file for a C++ project:

1. Open your terminal or Git Bash.
2. Navigate to the directory where you want to download the file.
3. Use the `git clone` command followed by the URL of the remote repository:

```shell
git clone <repository-url>
```

   Replace `<repository-url>` with the URL of the remote repository you want to clone.
4. Change your current directory to the cloned repository:

```shell
cd <repository-name>
```

   Replace `<repository-name>` with the name of the repository you just cloned.
5. List the files in the repository to find the desired file:

```shell
ls
```

Regenerate response

   This command will display all the files and directories in the repository.
6. Once you locate the file you want to download, you can either copy it manually or use the `git checkout` command to download the file to your current directory:

```shell
git checkout <branch-name> -- <path-to-file>
```

   Replace `<branch-name>` with the name of the branch where the file is located and `<path-to-file>` with the relative path to the file within the repository.
   For example, if the file you want to download is located in the `src` directory on the `main` branch, the command would be:

```shell
git checkout main -- src/file.cpp
```

   This will download the `file.cpp` to your current directory.

Now you have successfully downloaded the file from the remote repository to your local machine. You can use it for your C++ project as needed.

59.Bəzi hallarda atma xətaları üçün C++ kodunu yazın

```
 # daxil <iostream>
# daxil <stdexcept>

etibarsız dividend Nömrələri (int dividend, int divisor) {
    əgər (bölən == 0) {
        throw std::runtime_error("Sıfıra bölünmə xətası!");
    }

    int nəticə = dividend / bölücü;
    std::cout << "Nəticə: " << nəticə << std::endl;
}

int main() {
    int dividend, bölücü;

    std::cout << "Dividend daxil edin: ";
    std::cin >> dividend;

    std::cout << "Bölən daxil edin: ";
    std::cin >> bölən;

    cəhd {
        bölünən nömrələr (dividend, bölən);
    } tutmaq (const std::runtime_error & xəta) {
        std::cerr << "Xəta: " << error.what() << std::endl;
```

```
    }


    0 qaytarmaq;
}
```

---

60. Daimi göstəricilər üçün C++ kodunu yazın

```
# daxil <iostream>


int main() {
    int dəyəri = 5;
    int anotherValue = 10;

    // Tam ədədə daimi göstərici
    int* const ptr = &value;

    std::cout << "Dəyər: " << *ptr << std::endl; // Nəticə: Dəyər: 5

    * ptr = 8;
    std::cout << "Dəyişdirilmiş dəyər: " << *ptr << std::endl; // Çıxış: Dəyişdirilmiş dəyər: 8

    // Kompilyasiya xətası: ptr yenidən təyin edilə bilməz
    // ptr = &anotherValue;

    0 qaytarmaq;
}
```