Plepa Eğitim Hizmetleri

C/C++ Eğitim

C++ Kurs İçeriği

- **C++ Dilinin Genel Tanıtımı**
  - C++ dilinin tarihçesi
  - C++ dili ve programlama paradigmaları
  - C++ dili standartları
    - C++98 – 03
    - C++11
    - C++14
    - C++17
    - C++20
  - eski C++ ve modern C++

- **C Dili ve C++ İçindeki C Dili**
  - C dilinden C++ diline geçiş
  - işlev bildirimleri ve tanımlamalarına ilişkin farklılıklar
  - türlere ve tür dönüşümlerine ilişkin farklılıklar
  - C'de geçerli C++'da geçersiz durumlar
  - C99 ve C++

- **Temel Kavramlar**
  - tamamlanmış ve eksik türler *(complete & incomplete types)*
  - ifadelerin değer kategorileri *(value categories)*
  - tanımsız davranış *(undefined behavior)*
  - derleyiciye bağlı davranışlar. *(implementation defined & implementaion specified)*
  - derleyici eklentileri *(compiler extensions)*
  - kapsam *(scope)* ve isim arama *(name lookup)*
  - erişim kontrolü *(access control)*
  - çift anlamlılık hatası *(ambiguity)*

- **İlk Değer Verme (Initialization)**
  - eş biçimli ilk değer verme *(uniform initialization)*
    - daraltıcı dönüşümler *(narrowing conversions)*

    - *most vexing parse*
  - doğrudan ilk değer verme (*direct intialization*)
  - kopyalama ile ilk değer verme *(copy initialization)*
  - varsayılan ilk değer verme *(default initialization)*
  - *bileşiklere ilk değer verme (aggregate initialization)*

- **Tür Çıkarımı (Type Deduction)**
  - *auto* belirteci ile tür çıkarımı *(auto type deduction)*
  - *decltype* belirteci ile tür çıkarımı
  - sonradan gelen geri dönüş türü *(trailing return type)*
  - *auto* geri dönüş değeri türü *(auto return type)*

- **enum Sınıfları (enum Classes)**
  - geleneksel *enum* türleri
  - baz tür seçimi *(underlying type)*
  - tür dönüşümleri *(type conversions)*
  - *enum* sınıfları ve kapsam *(scope)*

- **Sabit İfadeleri (Constant Expressions)**
  - *const* anahtar sözcüğü ve *const* semantiği
  - *const* nesneler
  - *constexpr* anahtar sözcüğü
  - *constexpr* işlevler

- **İşlevlerin Varsayılan Argüman Alması (Default Arguments)**

- **Referans Semantiği**
  - sol taraf referansları *(L value references)*
  - sağ taraf referansları *(R value references)*
  - referanslar ve const semantiği *(references & const semantics)*
  - referanslar ile göstericilerin *(pointer)* karşılaştırılması
  - parametresi referans olan işlevler
  - referans döndüren işlevler

- **İşlev Yüklemesi (Function Overloading)**
  - genel kurallar
  - yüklenmiş işlev çözümlenmesi *(function overload resolution)*
  - *const* yüklemesi *(const overloading)*
  - *extern “C”* bildirimi
  - işlev yüklemesinde dikkat edilmesi gereken durumlar

- **Tür Dönüştürme Operatörleri (Type-cast Operators)**
  - *static_cast*<> operatörü

- *const_cast*<> operatörü
- *reinterpret_cast*<> operatörü
- *dynamic_cast*<> operatörü

- **Sınıflara giriş**
  - *class scope* kavramı
  - sınıflar ve isim arama *(name lookup)*
  - erişim kontrolü *(access control)* ve veri gizleme *(data hiding)*
    - *public* öğeler
    - *private* öğeler
    - *protected* öğeler
  - sınıfların öğeleri *(class members)*
    - sınıfların veri öğeleri *(data members)*
      - *non-static* veri öğeleri
        - *mutable* veri öğeleri
      - *static* veri öğeleri
    - sınıfların üye işlevleri
      - *non-static* üye işlevler
        - *const* üye işlevler
      - *static* üye işlevler
    - sınıfların tür öğeleri *(type members)*
  - sınıfların kurucu işlevleri *(constructors)*
    - kurucu işlev ilk değer verme listesi *(constructor initializer list)*
    - delege eden kurucu işlevler *(delegating constructors)*
  - sınıfların sonlandırıcı işlevleri *(destructors)*
  - üye işlevlerin çağrılması
  - sınıflar ve *const* semantiği
    - *const* sınıf nesneleri
  - geçici sınıf nesneleri *(temporary objects)*
    - otomatik *(implicit)* tür dönüşümleri
    - *explicit* kurucu işlevler
  - *mutable* anahtar sözcüğü
  - *friend* bildirimi
    - global işlevlere friend bildirimi
    - sınıfların üye işlevlerine friend bildirimi
    - sınıflara friend bildirimi
    - *attorney client idiyomu*

- **Sınıfların Özel Üye İşlevleri ve Kopyalama İşlemleri (Special Member Functions & Copy Control)**
  - default constructor

- destructor
- copy constructor
- move constructor
- copy assignment
- move assignment
- özel işlevlerin *default* edilmesi
- özel işlevlerin *delete* edilmesi
- sınıflar ve taşıma semantiği *(move semantics)*
- rule of zero
- rule of five
- kopyala takas et idiyomu *(copy & swap idiom)*
- kopyalamanın eliminasyonu *(copy elimination)*

- **Operatör Yüklemesi (Operator Overloading)**
  - operatör yüklemesine ilişkin genel kurallar
  - üye operatör fonksiyonları
  - global operatör fonksiyonları
  - aritmetik operatörlerin yüklenmesi
  - karşılaştırma operatörlerinin yüklenmesi
  - ++ ve — operatörlerinin yüklenmesi
  - ok operatörü ve içerik operatörlerinin yüklenmesi
  - [] operatörünün yüklenmesi
  - fonksiyon çağrı operatörünün yüklenmesi
  - tür dönüştürme operatör fonksiyonlarının yüklenmesi

- **Dinamik Ömürlü Nesneler**
  - *new* ve *delete* ifadeleri
  - *new[]* ve *delete []* ifadeleri
  - *operator new* işlevleri
  - *operator delete* işlevleri
  - *operator new* ve *operator delete* işlevlerinin yüklenmesi
  - std::bad_alloc
  - std::set_new_handler ve *std::get_new_handler*
  - *placement new* operatörleri
  - *nothrow* new

- **Tür Eş İsimleri (Type Alias)**
  - *typedef* bildirimleri
  - using bildirimleri

- **İsim Alanları (Namespaces)**
  - isim alanlarının oluşturulması

- isim alanları ve isim arama
  - çözünürlük operatörü ve isim alanları
- *using* bildirimi *(using declaration)*
- using namespace direktifi
- argümana bağlı isim arama *(argument dependent lookup)*
- isimsiz isim alanı *(unnamed namespace)*
- içsel isim alanları *(nested namespace)*
- inline isim alanları *(inline namespaces)*
- isim alanı eş ismi *(namespace alias)*
- işlev yüklemesi ve isim alanları

- **Sınıflar ve Kalıtım**
  - nesne yönelimli programlama ve kalıtım (*oop & inheritance*)
  - *public* kalıtımı *(public inheritance)*
  - çalışma zamanı çok biçimliliği *(runtime polymorphism)*
    - sanal işlevler *(virtual function)*
    - saf sanal işlevler *(pure virtual function)*
    - sanal sonlandırıcı işlev *(virtual destructor)*
    - sanal kurucu işlev idiyomu *(virtual constructor idiom)*
    - *override* bağlamsal anahtar sözcüğü
    - nesne dilimlenmesi *(object slicing)*
  - final bağlamsal anahtar sözcüğü
    - *final* sınıflar *(final classes)*
    - *final* override
  - çoklu kalıtım *(multiple inheritance)*
    - çoklu kalıtımda kapsam ve isim arama
    - çoklu kalıtımda kurucu ve sonlandırıcı işlevler
    - elmas formasyonu *(diamond formation)*
    - sanal kalıtım *(virtual inheritance)*
    - çoklu kalıtım ve kalıtımla alınan kurucu işlevler
    - çoklu kalıtımda kopyalama ve taşıma işlemleri
  - *private* kalıtımı *(private inheritance)*
  - *protected* kalıtımı *(protected inheritance)*
  - sınıf içi *using* bildirimi
  - kalıtımla alınan kurucu işlev *(inherited constructor)*
  - sanal olmayan arayüz idiyomu *(non-virtual interface idiom)*

- **Olağan Dışı Durumların İşlenmesi (Exception Handling)**
  - exception güvenliği *(exception safety)*
  - hata nesnelerinin gönderilmesi
    - *throw* deyimi *(throw statement)*

- *rethrow* deyimi *(rethrow statement)*
- *try* blokları
- *catch* blokları
  - *catch all*
- yakalanamayan hata nesnesi *(uncaught exception)*
  - *std::terminate*
  - *std::set_terminate*
- hata nesnesinin yeniden gönderilmesi *(rethrow statement)*
- yığının geri sarımı *(stack unwinding)*
- kurucu işlevlerden *exception* gönderimi
- sonlandırıcı işlevler ve hata gönderimi
- *exception handling* ve kalıtım
- *exception handling* ve dinamik ömürlü sınıf nesneleri
  - exception güvenliği için akıllı göstericilerin kullanımı
- işlev *try* blokları *(function try block)*
- *noexcept* belirleyicisi
  - beklenmeyen hata nesnesi
  - *std::unexpected_exception*
- *std::exception* sınıfı ve hiyerarşisi
  - *std::exception* sınıfı ve *what* sanal fonksiyonu
  - *std::logic_error*
    - *std::invalid_argument*
    - *std:.domain_error*
    - *std::length_error*
    - *std::out_of_range*
    - *std::future_error*
  - *std::runtime_error*
    - *std::range_error*
    - *std::overflow_error*
    - *std::underflow_error*
    - *std::system_error*
    - *std::regex_error*
  - *std::bad_alloc*
  - *std::bad_typeid*
  - *std::bad_cast*
  - *std::bad_exception*
  - *std::bad_weak_ptr*
  - *std::bad_function_call*
- kendi hata sınıflarımızı oluşturmak
- basic exception guarantee

- strong exception gurantee
- no exception gurantee
- std::current_exception
- std::exception_ptr
- std::rethrow_exception

- **Çalışma Zamanında Tür Belirlenmesi (RTTI)**
  - *dynamic_cast* operatörü
  - *typeid* operatörü
  - *std::typeinfo* sınıfı

- **std::string sınıfı**
  - genel kavramlar
  - arama işlevleri
  - set işlemleri
  - erişim işlemleri
  - karşılaştırma işlevleri
  - sayısal dönüşüm işlevleri

- **Bileşik Nesneler (composition)**
  - öğe olan nesneler ve özel işlevler, kopyalama kontrolü.
  - öğe olan nesneler ve erişim kontrolü
  - bileşik nesnelerin kullanıldığı temalar

- **İçsel türler (Type Members)**
  - sınıf içinde yapılan eş isim bildirimleri
  - içsel sınıflar *(nested classes)*
  - *pimpl* idiyomu

- **Şablonlar (Templates)**
  - şablon tür parametreleri *(template type parameters)*
  - şablon sabit parametreleri *(template non-type parameters)*
  - şablon şablon parametreleri *(template template parameters)*
  - şablon argümanları *(template arguments)*
  - şablonlardan kod üretimi *(template instantiation)*
  - fonksiyon şablonları *(function templates)*
    - fonksiyon şablonlarında tür çıkarımı *(function template argument deduction)*
    - fonksiyon şablonlarının yüklenmesi *(function template overloading)*
  - sınıf şablonları *(class templates)*
    - kurucu işlev ile tür çıkarımı *(CTAD)*
    - üye şablonlar *(member templates)*
  - şablonların özelleştirilmesi *(template specialization)*

    - tam özelleştirme *(full specialization)*
    - kısmi özelleştirme *(partial specialization)*
  - *sfinae*
  - değişken sayıda parametreli şablonlar *(variadic templates)*
  - mükemmel gönderim *(perfect forwarding)*
  - *katlama ifadeleri (fold expressions)*
  - *if constexpr*
  - değişken şablonları *(variable templates)*
  - eş isim şablonları *(alias templates)*

- **İteratörler (iterators)**
  - aralık *(range)* kavramı
  - iteratörlerin kategorileri
  - kapların *begin* ve *end* işlevleri
  - global *begin* ve *end* işlevleri
  - *iterator* işlevleri
    - std::next
    - std::prev
    - std::iter_swap
    - std::advance
    - std::distance
  - iterator uyumlandırıcıları *(iterator adaptors)*
    - akım iteratörleri *(stream iterators)*
      - *istream_iterator*
      - *ostream_iterator*
      - *istreambuf_iterator*
      - *ostreambuf_iterator*
    - *reverse_iterators*
    - *move_iterator*
    - *insert_iterator*
    - *back_insert _iterator*
    - *front_insert_iterator*

- **Kaplar***(Containers)*
  - STL kapları ve veri yapıları *(STL containers & data structures)*
  - sıralı kaplar *(sequence containers)*
    - *std::vector*
    - *std::deque*
    - std::string
    - std:.array
    - *std::list*

    - *std::forward_list*
  - ilişkisel kaplar *(associative containers)*
    - *std::set*
    - *std::multiset*
    - *std::map*
    - *std::multimap*
  - sırasız ilişkisel kaplar (unordered containers)
    - *std::unordered_set*
    - *std::unordered_multiset*
    - *std::unordered_map*
    - *std::unordered_multimap*
  - kapların tür öğeleri *(type members of containers)*
  - kapların *emplace* işlevleri

- **Kap Uyumlandırıcıları (Container Adaptors)**
  - stack
  - queue
  - priority_queue

- **Algoritmalar (Algorithms)**
  - algoritmaların temel özellikleri ve genel ilkeler
  - salt okuyan algoritmalar *(non-modifying algorithms)*
  - kap öğelerini değiştiren algoritmalar *(modifying algorithms)*
  - kap öğelerini konumlandıran algoritmalar *(mutating algorithms)*
  - sıralama ile ilgili algoritmalar *(sorting algorithms)*
  - sıralanmış aralıklar üzerinde koşturulan algoritmalar *(sorted range algorithms)*
  - nümerik algoritmalar *(numeric algorithms)*
  - algoritmaların lambda ifadelerini kullanması

- **lambda ifadeleri**
  - kapanış türleri ve kapanış nesneleri *(closure types and closure objects)*
  - lambda ifadeleri ve tür çıkarımı *(lambda expressions and type deduction)*
  - lambda yakalama ifadeleri *(lambda captures)*
  - lambda init capture
  - capture this
  - capture *this
  - *mutable* lambdalar
  - *trailing return type*
  - genelleştirilmiş lambda ifadeleri *(generalized lambda expressions)*
  - algoritmalarda lambda ifadelerinin kullanımı

- **Akıllı Gösterici Sınıfları (Standard Smart Pointer Classes)**

- unique_ptr sınıfı
  - *std::make_unique*
  - *std::default_delete ve custom deleters*
  - tipik hatalar
- shared_ptr sınıfı
  - referans sayımı *(reference counting)*
  - *std::make_shared*
- weak_ptr sınıfı

- **Standart Giriş Çıkış Kütüphanesi**
  - giriş çıkış akımlarına ilişkin standart sınıflar *(standard stream classes)*
  - global akım nesneleri
  - formatlı giriş çıkış işlemleri *(formatted input output)*
  - << ve >> operatörlerinin yüklenmesi *(inserter & extractors)*
  - formatlama ve formatlama işlemleri (*formatting*)
  - manipülatörler *(manipulators)*
  - akımın durumu (*condition states*)
  - string akımları *(stringstreams)*
  - dosya işlemleri *(file operations)*
  - formatsız giriş ve çıkış işlemleri *(unformatted input output)*
  - bellek üstünde yapılan giriş çıkış işlemleri

- **Bazı önemli STL Öğelerinin Tanıtımı**
  - *std::pair*
  - std::tuple
  - *std::initializer_list*
  - std::bitset
  - std::regex
  - *type_traits* kütüphanesi
  - *std::allocator*
  - *std::ratio*
  - *std::chrono*
  - standart *random* kütüphanesi
  - *std::string_view* sınıfı
  - *std::optional* sınıfı
  - *std::variant* sınıfı
  - *std::any* sınıfı
  - *std::byte*
  - *std::invoke*

- **Tamamlayıcı Araçlar ve Sentaks Öğeleri**
  - *static_assert*

- decltype (auto)
- declval
- üye fonksiyon göstericileri *(member function pointers)*
- ham string sabitleri *(raw string literals)*
- ikilik sayı sisteminde yazılan sabitler *(binary literals)*
- basamak ayırıcısı *(digit seperator)*
- ilk değer vermeli if deyimi *(if with initializer)*
- ilk değer vermeli *switch* deyimi *(if with initializer)*
- *alignas* belirteci *(alignas specifier)*
- *alignof* operatörü *(alignof operator)*
- yapısal bağlama *(structural binding)*
- *attribute'lar*

- **Concurrency**
  - memory model
  - thread'ler ve thread yönetimi
  - *std::this_thread* isim alanı
  - data race kavramı ve data_race'den kaçınma
  - standart *mutex* sınıfları ve *mutex* işlemleri
  - *lock_guard* ve *unique_lock* sınıfları
  - *std::condition_variable* sınıfı
  - *std::future* ve *std::promise* sınıfları
  - *std::async* işlevi
  - atomik türler *(atomic types)*
  - görev tabanlı *(task based)* programlama
  - *std::packaged_task* sınıfı
  - paralel STL algoritmaları

Düzenlediğimiz eğitim programları hakkında bilgi edinmek ya da eğitim taleplerinizi iletmek için **info@plepa.com** adresine e-posta mesajı gönderebilirsiniz.

Bizi Takip Edin

Subscribe