C ++ i orientacja obiektowa

C ++ jest zorientowanym obiektowo językiem programowania zbudowanym na bazie języka C. Ta prezentacja zawiera bardzo krótkie wprowadzenie do C ++, dotykając wielu ważnych aspektów C ++, dzięki czemu będziesz mógł śledzić nasze prezentacje implementacji modeli sieci neuronowych w C ++ i pisać własne programy w C ++. Język C ++ jest nadzbiorem języka C. Możesz pisać programy C ++, takie jak programy C, lub możesz korzystać z obiektowych funkcji C ++, aby pisać programy zorientowane obiektowo. Co sprawia, że język programowania lub metodologia programowania jest obiektowa? Cóż, istnieje kilka niepodważalnych filarów orientacji obiektowej. Funkcje te wyróżniają się bardziej niż inne pod względem orientacji obiektowej. Są to hermetyzacja, ukrywanie danych, przeciążanie, polimorfizm i dziadek ich wszystkich: dziedziczenie. Każdy z filarów orientacji obiektowej zostanie omówiony później, ale zanim się z nimi uporamy, musimy odpowiedzieć na pytanie: co dają te wszystkie rzeczy zorientowane obiektowo? Korzystając z obiektowych funkcji C ++ w połączeniu z Object-Oriented Analysis and Design (OOAD), która jest metodologią w pełni wykorzystującą orientację obiektową, możesz mieć dobrze zapakowane, wielokrotnego użytku, rozszerzalne i niezawodne programy i segmenty programów . Jest to poza zakresem naszej dyskusji na temat OOAD

Hermetyzacja

W C ++ masz możliwość enkapsulacji danych i operacji, które manipulują tymi danymi w odpowiednim obiekcie. Umożliwia to wykorzystanie tych zbiorów danych i funkcji, zwanych obiektami, w programach innych niż program, dla którego zostały pierwotnie utworzone. Z obiektami, podobnie jak w tradycyjnej koncepcji podprogramów, tworzysz funkcjonalne bloki kodu. Nadal masz dostępne abstrakcje obsługiwane w języku, takie jak zakres i osobna kompilacja. Jest to podstawowa forma hermetyzacji. Obiekty przenoszą hermetyzację o krok dalej. W przypadku obiektów definiujesz nie tylko sposób działania funkcji lub jej implementacji, ale także sposób dostępu do obiektu lub jego interfejsu. Możesz określić dostęp inaczej dla różnych podmiotów. Na przykład, możesz sprawić, aby funkcja do_operation() zawarta w obiekcie A była dostępna dla obiektu B, ale nie dla obiektu C. Tej kwalifikacji dostępu można również użyć dla elementów danych wewnątrz obiektu. Hermetyzacja danych i zamierzone operacje na nich zapobiegają poddaniu danych działaniu nieprzeznaczonym dla nich. To właśnie sprawia, że przedmioty są wielokrotnego użytku i przenośne! Operacje są zwykle podawane w postaci funkcji działających na elementach danych. Takie funkcje są również nazywane metodami w niektórych obiektowych językach programowania. Elementy danych i funkcje, które nimi manipulują, są łączone w strukturę zwaną klasą. Klasa jest abstrakcyjnym typem danych. Kiedy tworzysz instancję klasy, masz obiekt. Nie różni się to od utworzenia typu liczb całkowitych w celu utworzenia zmiennych i i j. Na przykład projektujesz klasę o nazwie ElectronicBook, z elementem danych o nazwie ArrayofPages. Kiedy tworzysz instancję swojej klasy, tworzysz obiekty typu ElectronicBook. Załóżmy, że utworzysz dwa z nich o nazwie EB_Geography i EB_History. Każdy obiekt, który jest tworzony, ma w sobie swój własny element danych, do którego odwołuje się ArrayOfPages.

Ukrywanie danych

Z pojęciem enkapsulacji związana jest koncepcja ukrywania danych. Hermetyzacja ukrywa dane przed innymi klasami i funkcje w innych klasach. Wracając do klasy ElectronicBook, możesz zdefiniować funkcje takie jak GetNextPage, GetPreviousPage i GetCurrentPage jako jedyny sposób dostępu do informacji w elemencie danych ArrayofPages za pomocą funkcji, które uzyskują dostęp do obiektu ElectronicBook. Chociaż w klasie ElectronicBook może być sto jeden innych atrybutów i elementów danych, wszystkie są ukryte. Dzięki temu programy są bardziej niezawodne, ponieważ opublikowanie określonego interfejsu do obiektu zapobiega przypadkowemu dostępowi do danych w sposób, który nie został zaprojektowany ani uwzględniony. W C ++ dostęp do obiektu oraz jego enkapsulowane dane

i funkcje są traktowane bardzo ostrożnie, przy użyciu słów kluczowych private, protected i public. Można określić specyfikacje dostępu do obiektów danych i funkcji jako prywatne, chronione lub publiczne podczas definiowania klasy. Tylko wtedy, gdy deklaracja jest publiczna, inne funkcje i obiekty mają dostęp do obiektu i jego komponentów bez pytania. Z drugiej strony, jeśli deklaracja ma charakter prywatny, nie ma takiej możliwości dostępu. Gdy podana deklaracja jest tak samo chroniona, to dostęp do danych i funkcji w klasie przez innych nie jest tak swobodny jak wtedy, gdy jest publiczny, ani tak ograniczony, jak gdy jest prywatny. Możesz zadeklarować jedną klasę jako pochodną innej klasy, co zostanie omówione wkrótce. Tak zwane klasy pochodne i klasa deklarująca uzyskują dostęp do komponentów obiektu, które są zadeklarowane jako chronione. Jedna klasa, która nie jest klasą pochodną drugiej klasy, może uzyskać dostęp do elementów danych i funkcji drugiej klasy, jeśli zostanie zadeklarowana jako klasa zaprzyjaźniona w drugiej. Trzy typy deklaracji specyfikacji dostępu mogą być różne dla różnych składników obiektu. Na przykład niektóre elementy danych można uznać za publiczne, niektóre prywatne, a inne chronione. Ta sama sytuacja może wystąpić w przypadku funkcji w obiekcie. Jeśli nie zostanie złożone wyraźne oświadczenie, domyślna specyfikacja jest prywatna.

Konstruktory i Destruktory jako specjalne funkcje C ++

Konstruktory i destruktory są specjalnymi funkcjami w C ++. Definiują sposób tworzenia i niszczenia obiektu. Nie można zdefiniować klasy w programie C ++ bez zadeklarowania i zdefiniowania dla niej co najmniej jednego konstruktora. Możesz pominąć deklarowanie, a następnie zdefiniowanie destruktora tylko dlatego, że używany kompilator utworzy domyślny destruktor. Dla klasy można zadeklarować więcej niż jednego konstruktora, ale tylko jeden destruktor. Konstruktory służą do tworzenia obiektu klasy i inicjowania go. C ++ wymaga, aby każda funkcja miała typ return. Jedynymi wyjątkami są konstruktory i destruktory. Konstruktor otrzymuje taką samą nazwę jak klasa, dla której jest konstruktorem. Może to wymagać argumentów lub może ich nie potrzebować. Różne konstruktory dla tej samej klasy różnią się liczbą i typami argumentów, które przyjmują. Dobrym pomysłem jest zapewnienie dla każdej klasy przynajmniej domyślnego konstruktora, który nie przyjmuje żadnych argumentów i nie robi nic poza tworzeniem obiektu tego typu klasy. Konstruktor jest wywoływany w momencie, gdy trzeba utworzyć obiekt jego klasy. Destruktor ma również taką samą nazwę jak klasa, dla której jest destruktorem, ale z tyldą (~) poprzedzającą nazwę. Zazwyczaj w destruktorze wykonuje się instrukcje, które proszą system o usunięcie różnych struktur danych utworzonych dla klasy. Pomaga to zwolnić przydzieloną pamięć dla tych struktur danych. Destruktor jest wywoływany, gdy wcześniej utworzony obiekt nie jest już potrzebny w programie.

Dynamiczna alokacja pamięci

C ++ ma słowa kluczowe new i delete, które są używane jako para w tej kolejności, chociaż są oddzielone innymi instrukcjami programu. Służą do dynamicznego przydzielania pamięci w momencie tworzenia obiektu klasy i do zwalniania takiej przydzielonej pamięci, gdy nie jest już potrzebna. Za pomocą nowego tworzysz przestrzeń na stercie. Eliminuje to potrzebę C ++ dla malloc, który jest funkcją dynamicznego przydzielania pamięci używaną w C.

Przeciążenie

Hermetyzacja danych i funkcji pozwoliłaby również na użycie tej samej nazwy funkcji w dwóch różnych obiektach. Wielokrotne użycie nazwy dla funkcji nie musi być tylko w różnych deklaracjach obiektowych. W tym samym obiekcie można używać tej samej nazwy dla funkcji o różnych funkcjach, jeśli można je rozróżnić pod względem typu zwracanego lub pod względem typów argumentów i liczby. Ta funkcja nazywa się przeciążeniem. Na przykład, jeśli dwa różne typy zmiennych są elementami danych w obiekcie, często nazywaną funkcją może być dodatek, po jednym dla każdego z dwóch typów

zmiennych - wykorzystując w ten sposób przeciążenie. Następnie mówi się, że dodanie funkcji jest przeciążone. Pamiętaj jednak, że funkcja main jest prawie jedyną funkcją, której nie można przeciążyć.

Polimorfizm i funkcje polimorficzne

Funkcja polimorficzna to funkcja, której nazwa jest używana w programie na różne sposoby. Można jej również uznać za wirtualny, jeśli zamiar jest później wiążący. Umożliwia to związanie go w czasie wykonywania. Późne wiązanie jest również określane jako wiązanie dynamiczne. Zaletą zadeklarowania funkcji w obiekcie jako wirtualnym jest to, że jeśli program korzystający z tego obiektu wywołuje tę funkcję tylko warunkowo, nie ma potrzeby wcześniejszego wiązania funkcji podczas kompilacji programu. Będzie związany tylko wtedy, gdy warunek zostanie spełniony i nastąpi wywołanie funkcji. Na przykład możesz mieć funkcję polimorficzną o nazwie draw(), która jest powiązana z różnymi obiektami graficznymi, takimi jak prostokąt, okrąg i kula. Szczegóły lub metody funkcji są różne, ale nazwa draw() jest powszechna. Jeśli teraz masz kolekcję tych obiektów i wybierzesz dowolny obiekt, nie wiedząc dokładnie, co to jest (na przykład za pomocą wskaźnika), nadal możesz wywołać funkcję rysowania dla obiektu i mieć pewność, że właściwa funkcja rysowania będzie związany z obiektem i wywoływany.

Przeciążenie operatorów

Oprócz funkcji przeciążania można przeciążać operatorów. W rzeczywistości zdefiniowany przez system operator przesunięcia w lewo << jest również przeciążony w C ++, gdy jest używany z cout, odmianą C ++ funkcji printf języka C. Podobna sytuacja występuje z prawym operatorem shift >> w C ++, gdy jest używany z cin, odmianą C ++ funkcji skanowania języka C. Możesz wziąć dowolny operator i go przeciążyć. Ale chcesz być ostrożny i nie przesadzać, a także nie powodować zamieszania, gdy przeciążasz operator. Główną zasadą w tym względzie jest stworzenie obiektu oszczędzającego kod i oszczędzającego czas przy jednoczesnym zachowaniu prostoty i przejrzystości. Przeciążenie operatora jest szczególnie przydatne do wykonywania normalnej arytmetyki na niestandardowych typach danych. Możesz na przykład przeciążyć symbol mnożenia, aby na przykład pracować z liczbami zespolonymi.

Dziedziczenie

Podstawowym rozróżnieniem dla C ++ od C jest to, że C ++ ma klasy. Obiekty są zdefiniowane w klasach. Same klasy mogą być elementami danych w innych klasach, w którym to przypadku jedna klasa byłaby elementem innej klasy. Oczywiście wtedy jedna klasa jest członkiem, który niesie ze sobą własne dane i funkcje, w drugiej klasie. Ten rodzaj relacji jest określany jako relacja „has-a”: Obiekt A ma w sobie Obiekt B. Relację między klasami można ustanowić nie tylko poprzez uczynienie jednej klasy członkiem innej, ale także poprzez proces wyprowadzania jednej klasy z drugiej. Jedna klasa może pochodzić z innej klasy, która staje się jej klasą podstawową. Następnie ustanawia się hierarchię klas i ustanawia się rodzaj relacji rodzic-dziecko między klasami. Klasa pochodna dziedziczy po klasie podstawowej niektóre elementy danych i funkcje. Ten rodzaj relacji określa się mianem relacji „is-a”. Możesz mieć klasę Prostokąt pochodzącą z klasy Kształt, ponieważ Prostokąt jest Kształtem. Oczywiście, jeśli klasa A pochodzi z klasy B, a jeśli sama B pochodzi z klasy C, to A dziedziczy zarówno z B, jak i C. Klasa może pochodzić z więcej niż jednej klasy. Tak zachodzi wielokrotne dziedziczenie. Dziedziczenie to potężny mechanizm do tworzenia podstawowej funkcjonalności, który jest przekazywany następnej generacji w celu dalszego ulepszenia lub modyfikacji.

Klasy pochodne

Gdy jedna klasa ma niektórych członków zadeklarowanych w niej jako chronione, wówczas takie elementy byłyby ukryte przed innymi klasami, ale nie przed klasami pochodnymi. Innymi słowy, wyprowadzenie jednej klasy z drugiej jest sposobem na uzyskanie dostępu do chronionych członków klasy nadrzędnej przez klasy pochodne. Następnie mówimy, że klasa pochodna dziedziczy po klasie nadrzędnej tych członków klasy nadrzędnej, które są zadeklarowane jako chronione lub publiczne. Deklarując klasę pochodną z innej klasy, można wprowadzić specyfikację dostępu lub widoczności, co oznacza, że taka pochodna może być publiczna lub przypadek domyślny prywatny.

Ponowne użycie kodu

C ++ jest również atrakcyjny ze względu na możliwość rozszerzenia napisanych w nim programów i możliwość ponownego użycia, dzięki wspomnianym wcześniej funkcjom w C ++, takim jak dziedziczenie i polimorfizm. Nowy projekt programistyczny może nie tylko ponownie wykorzystywać klasy utworzone dla innego programu, jeśli są one odpowiednie, ale może rozszerzyć o inny program o dodatkowe klasy i funkcje, które uznają za konieczne. Możesz dziedziczyć z istniejącej hierarchii klas i zmieniać funkcjonalność tylko tam, gdzie jest to konieczne.

Kompilatory C ++

Wszystkie nasze programy zostały skompilowane i przetestowane z Turbo C ++, Borland C ++, Microsoft C / C ++ i Microsoft Visual C ++. Oto niektóre z popularnych dostępnych komercyjnych kompilatorów C ++. Powinieneś także móc korzystać z większości innych komercyjnych kompilatorów C ++. Wszystkie programy powinny również łatwo przenosić się na inne systemy operacyjne, takie jak Unix i Mac, ponieważ wszystkie są programami znakowymi.

Pisanie programów w C ++

Zanim zaczniesz pisać program C ++ dla konkretnego problemu, musisz mieć jasny obraz różnych parametrów i zmiennych, które byłyby częścią definicji problemu i / lub jego rozwiązania. Ponadto powinno być jasne, jakie manipulacje należy wykonać podczas procesu rozwiązania. Następnie dokładnie określa się, jakie klasy są potrzebne i jakie relacje mają ze sobą w hierarchii klas. Pomyśl o relacjach is-a i has-a, aby zobaczyć, gdzie należy zdefiniować klasy i jakie klasy mogą pochodzić od innych. W tym momencie planu programista byłby znacznie bardziej zrozumiały dla programistów, jakie powinny być specyfikacje dostępu do danych i funkcji itd. Typowe komunikaty o błędach kompilacji, które może napotkać programista w C ++, informują, że dana funkcja lub dane nie należą do określonej klasy lub nie są dostępne dla danej klasy, lub że dla konkretnej klasy nie był dostępny konstruktor. Kiedy argumenty funkcji w deklaracji i miejscu, w którym wywoływana jest funkcja, nie pasują ani do liczb, ani do typów, ani do obu, kompilator uważa je za dwie różne funkcje. Kompilator nie znajduje definicji i / lub deklaracji jednego z nich i ma powód do narzekań. Ten rodzaj błędu w jednym wierszu kodu może spowodować, że kompilator ostrzeże Cię, że występuje również kilka innych błędów, być może niektóre z nich związane z niewłaściwą interpunkcją. W takim przypadku naprawienie wskazanego podstawowego błędu rozwiązałoby wiele innych spornych kwestii. Poniższa lista zawiera kilka dodatkowych informacji, o których należy pamiętać, kiedy piszesz programy w C ++.

• Element x obiektu A jest określany za pomocą A.x, podobnie jak w przypadku elementów konstrukcji w C.

• Jeśli zadeklarujesz klasę B, wówczas funkcja konstruktora ma również nazwę B. B nie ma typu zwracanego. Jeśli ten konstruktor przyjmuje, powiedzmy, jeden argument typu liczba całkowita, definiujesz konstruktor za pomocą składni: B :: B (int) {cokolwiek robi funkcja};

• Jeśli zadeklarujesz funkcję członka C klasy B, gdzie zwracanym typem C jest, powiedzmy, zmiennoprzecinkowe, a C przyjmuje dwa argumenty, jeden typu zmiennoprzecinkowego, a drugi int, wówczas definiujesz C składnią: float B :: C (float, int) {cokolwiek robi funkcja};

• Jeśli zadeklarujesz funkcję składową D klasy B, w której D nie zwraca żadnej wartości i nie przyjmuje żadnych argumentów, definiujesz D używając składni: void B :: D () {cokolwiek robi funkcja};

• Jeśli G jest klasą pochodną, powiedzmy, klasy B poprzednio wspomnianej, deklarujesz G używając składni: class G: B. Konstruktor dla G jest zdefiniowany za pomocą składni: G :: G (argumenty G): B (int) {cokolwiek robi funkcja}. Jeśli natomiast G wywodzi się, powiedzmy, z klasy B, a także z klasy T, to deklarujesz G używając składni: class G: B, T.

• Jeśli jedna klasa zostanie zadeklarowana jako pochodząca z więcej niż jednej innej klasy, to znaczy, jeśli istnieje dla niej więcej niż jedna klasa bazowa, specyfikacja pochodnych może być inna lub taka sama. Zatem klasa może pochodzić z jednej klasy publicznie, a jednocześnie z innej klasy prywatnie.

• Jeśli zadeklarowałeś zmienną globalną y zewnętrzną względem klasy B i jeśli masz również element danych y w klasie B, możesz użyć zewnętrznego y z symbolem odniesienia

::. Zatem :: y odnosi się do zmiennej globalnej, podczas gdy y, w funkcji składowej B, lub B.y odnosi się do elementu danych B. W ten sposób można odróżnić funkcje polimorficzne od każdej innej