1. Czym jest kompilator i linker?

**Kompilator** – program zamieniający kod źródłowy, napisany w jednym z języków

programowania, na kod maszynowy w postaci oddzielnych modułów.

**Linker** łączy skompilowane moduły kodu i inne pliki w jeden plik wykonywalny, czyli

program (w przypadku Windows – plik EXE).

2. Co to jest “include guard” lub inaczej “header guard”?

Zabezpieczenie pliku nagłówkowego przez wielokrotnym załączeniem do programu

#ifndef NazwaPliku\_h  
#define NazwaPliku\_h

Dzięki blokowi warunkowemu, zawartość pomiędzy #ifndef, a #endif jest pomijana, gdy została już w którymś fragmencie kodu załączona. Uniwersalne rozwiązanie, działające pod każdym kompilatorem.

3. Do czego służy #pragma once?

Zabezpieczenie pliku nagłówkowego przez wielokrotnym załączeniem do programu

#pragma once

Dyrektywa #pragma once nakazuje by dany plik był załączony tylko raz. Dzięki tej dyrektywie kompilator jest w stanie nieco szybciej stwierdzić czy ten plik należy ponownie dołączyć czy nie (Visual C++).

4. Do czego służy słowo kluczowe “auto”?

Słowo kluczowe auto w standardzie C++11 zmieniło swoje znaczenie w stosunku do standardu C++. Słowo kluczowe auto wg standardu C++11 oznacza **zastępczy typ zmiennej**, który zostanie wydedukowany na podstawie wartości za pomocą której zmienna zostanie zainicjalizowana w chwili jej tworzenia. W przeciwnym wypadku zostanie zwrócony błąd kompilacji. Tworzona zmienna może zawierać modyfikatory takie jak const i volatile. Zmienna może być również **wskaźnikiem** (\*), **referencją** (&) czy też **referencją do r-wartości** (&&). Na zmiennych o typie zastępczym auto można używać słów kluczowych sizeof i typeid. Nie można natomiast odczytywać informacji za pomocą wspomnianych słów kluczowych bezpośrednio ze słowa kluczowego auto - wystąpi wówczas błąd kompilacji.

5. Omów na czym polega programowanie obiektowe?

**Programowanie obiektowe** (ang. *object-oriented programming*) – paradygmat programowania, w którym programy definiuje się za pomocą obiektów – elementów łączących *stan* (nazywane polami) i *zachowanie* (metody). Obiektowy program komputerowy wyrażony jest jako zbiór takich obiektów, komunikujących się pomiędzy sobą w celu wykonywania zadań.

Główne cechy programowania obiektowego

**Abstrakcja**

Każdy obiekt w systemie służy jako model abstrakcyjnego „wykonawcy”, który może wykonywać pracę, opisywać i zmieniać swój stan oraz komunikować się z innymi obiektami w systemie bez ujawniania, w jaki sposób zaimplementowano dane cechy. Procesy, funkcje lub metody mogą być również abstrahowane, a kiedy tak się dzieje, konieczne są rozmaite techniki rozszerzania abstrakcji.

Hermetyzacja

Czyli ukrywanie implementacji, enkapsulacja. Zapewnia, że obiekt nie może zmieniać stanu wewnętrznego innych obiektów w nieoczekiwany sposób. Tylko własne metody obiektu są uprawnione do zmiany jego stanu. Każdy typ obiektu prezentuje innym obiektom swój interfejs, który określa dopuszczalne metody współpracy.

Polimorfizm

Referencje i kolekcje obiektów mogą dotyczyć obiektów różnego typu, a wywołanie metody dla referencji spowoduje zachowanie odpowiednie dla pełnego typu obiektu wywoływanego. Jeśli dzieje się to w czasie działania programu, to nazywa się to *późnym wiązaniem* lub *wiązaniem dynamicznym*.

Dziedziczenie

Porządkuje i wspomaga polimorfizm i enkapsulację dzięki umożliwieniu definiowania i tworzenia specjalizowanych obiektów na podstawie bardziej ogólnych. Dla obiektów specjalizowanych nie trzeba redefiniować całej funkcjonalności, lecz tylko tę, której nie ma obiekt ogólniejszy. W typowym przypadku powstają grupy obiektów zwane *klasami*, oraz grupy klas zwane *drzewami*. Odzwierciedlają one wspólne cechy obiektów.

6. Omów wpływ modyfikatorów dostępu użytych podczas dziedziczenia na widoczność pól i metod.

Do modyfikatorów dostępu należą: *private, protected, public.* Wpływają na reguły widoczności i umożliwiają kontrolę dostępu do pól danych i metod klasy z innych klas.

Modyfikatory dostępu, mają następujące znaczenie:

***public*** - wszystkie klasy mają dostęp do pól danych i metod *public,*

***private*** - dostęp do metod i pól danych posiadają jedynie inne metody tej samej klasy,

***protected*** - metoda lub pole danych *protected* lub może być używana jedynie przez metody swojej klasy oraz metody wszystkich jej klas pochodnych,

7. Czym różni się metoda “virtual” od “pure virtual”?

Metoda wirtualna jest przygotowana na zastąpienie siebie przez nową wersję, zdefiniowaną w klasie pochodnej. Jej rolą nie jest już przede wszystkim tworzenie obiektów, gdyż równie ważne staje się służenie jako baza dla klas pochodnych. Aby daną funkcję składową zadeklarować jako wirtualną, należy poprzedzić jej prototyp słowem kluczowym virtual.

Metody nazywane czysto wirtualnymi (ang. pure virtual ) nie posiadają żadnej implementacji i są przeznaczone wyłącznie do przedefiniowania w klasach pochodnych.

Deklaracja takiej metody ma dość osobliwą postać. Oczywiście z racji nie posiadania żadnego kodu zbędne stają się nawiasy klamrowe wyznaczające jej blok, zatem całość przypomina zwykły prototyp funkcji.

8. Co to jest “vtable”?

Wywołanie w przypadku metod wirtualnych jest dwuetapowe i polega na wykorzystaniu pewnych dodatkowych informacji. Są nimi: tablica funkcji wirtualnych (zwana vtable) oraz wskaźnik na nią (czyli vptr). Tablica występuje tylko w jednej kopii na całą klasę i zawiera tyle elementów, ile funkcji wirtualnych klasa ta posiada. Jej elementami są adresy w pamięci tych właśnie funkcji: pierwsza funkcja wirtualna ma więc adres zapisany w elemencie vtable o indeksie 0, druga – o indeksie 1 itd.

9. Opisz różnicę pomiędzy destruktorem a wirtualnym destruktorem. Jaki należy stosować częściej i dlaczego?

Nieobecność wirtualnego destruktora w klasie bazowej może prowadzić do tzw. wycieków pamięci, czyli bezpowrotnej utraty zaalokowanej pamięci operacyjnej.

Jeżeli destruktor nie będzie oznaczony jako virtual , to kompilator potraktuje go jako zwyczajną metodę i zastosuje wobec niej technikę wczesnego wiązania i wywoła wyłącznie destruktor klasy bazowej. Rozwiązaniem problemu jest uczynienie destruktora klasy bazowej metodą wirtualną. Wtedy operator delete będzie usuwał obiekt, na który faktycznie wskazuje podany mu wskaźnik. Pamiętać należy zatem, aby zawsze umieszczać wirtualny destruktor w klasie bazowej.