**UTS T0044 TAHUN 20XX**

**Pilihan Ganda (40%)**

01.Fungsi *friend* adalah

a. fungsi bukan anggota klas untuk mengimplementasikan function overloading

b. fungsi bukan anggota klas untuk mengimplementasikan function overriding

c. fungsi bukan anggota klas untuk mengimplementasikan polimorfisme

**d. fungsi bukan anggota klas untuk mengakses data anggota klas**

02. *Modifier const* dapat digunakan untuk mendeklarasikan

a. pointer dan reference b. reference dan object

c. object dan pointer **d. pointer, reference dan object**

03. *Function overloading* dibedakan berdasarkan *function signatures* yaitu

a. tipe parameter dan banyak parameter

b. banyak parameter dan urutan parameter

c. urutan parameter dan tipe parameter

**d. tipe parameter, banyak parameter dan urutan parameter**

04. *Generic programming* adalah salah satu fitur utama OOP, dan di C++ generic programming diimplementasikan menggunakan

a. polymorphism b. inheritance **c. templates** d. packages

05. Deklarasi berikut ini yang benar adalah

a. int& r = 77; **b. int& r = nn;**

c. int& r = n++;  d. int& r = cubic(n);

06. Manfaat dari *inheritance* di C++ antara adalah

a. reuseability dan extendability

b. extendability dan function overriding

c. function overriding dan reuseability

**d. reuseability, extendability dan function overriding**

07. Opsi-opsi berikut adalah benar *kecuali*

a. Destruktor tidak boleh mempunyai return type.

b. Destruktor tidak boleh mempunyai argumen.

**c. Konstruktor tidak boleh mempunyai argumen.**

d. Konstruktor tidak boleh mempunyai return type.

08.Output dari penggalan program berikut

class Mhs {

public:

Mhs() {cout << “Constructor mahasiswa dieksekusi\n”}

~Mhs() {cout << “Destructor mahasiswa dieksekusi\n”}

};

void main() {

Mhs M01;

cout << “Antarobject M01 dan M02\n”;

Mhs M02;

}

adalah:

a. Constructor mahasiswa dieksekusi b. Constructor mahasiswa dieksekusi

Constructor mahasiswa dieksekusi Destructor mahasiswa dieksekusi

Antarobject M01 dan M02 Antarobject M01 dan M02

Destructor mahasiswa dieksekusi Constructor mahasiswa dieksekusi

Destructor mahasiswa dieksekusi Destructor mahasiswa dieksekusi

**c. Constructor mahasiswa dieksekusi** d. Destructor mahasiswa dieksekusi

**Antarobject M01 dan M02**  Constructor mahasiswa dieksekusi

**Constructor mahasiswa dieksekusi** Antarobject M01 dan M02

**Destructor mahasiswa dieksekusi** Constructor mahasiswa dieksekusi

**Destructor mahasiswa dieksekusi** Destructor mahasiswa dieksekusi

09. Relasi yang paling *reasonable* antara *class Car* dengan *class Machine* adalah

a. is a **b. has a** c. know a d. derive a

10. Untuk klas T, prototype untuk *overloaded stream output operator* adalah

a. friend ostream& operator>> (ostream&, const T&);

**b. friend ostream& operator<< (ostream&, const T&);**

c. inline ostream& operator>> (ostream&, const T&);

d. inline ostream& operator<< (ostream&, const T&);

11. Opsi berikut ini yang benar adalah

a. pointer dan reference keduanya tidak harus diinisialisasi.

**b. pointer tidak harus diinisialisasi sedangkan reference harus diinisialisasi.**

c. pointer harus diinisialisasi sedangkan reference tidak harus diinisialisasi.

d. pointer dan reference keduanya harus diinisialisasi.

12. Untuk klas T, sintaks *overloaded pre increment operator* adalah

**a. T operator++();** b. T operator++ (int);

c. T :: operator++ (); d. T :: operator++(int);

13. Esensinya *class* dan *struct* adalah sama dalam C++. Perbedaan yang paling signifikan di antara mereka adalah

1. default akses class adalah private sedangkan struct adalah private
2. **default akses class adalah private sedangkan struct adalah public**
3. default akses class adalah public sedangkan struct adalah private
4. default akses class adalah public sedangkan struct adalah private

14. Operator-operator berikut tidak dapat di-overload kecuali

**a. :**  b. :: c. :? d. .\*

15. *Constructor initialization list* dari Rasio (int n, int d) {num = n; den = d; } adalah

a. Rasio (int n, int d) : n (num), d (den) { }

b. Rasio (int n, int d) : n (num); d (den) { }

**c. Rasio (int n, int d) : num (n), den (d) { }**

d. Rasio (int n, int d) : num (n); den (d) { }

16. *Modifier const* dapat diberikan pada object maupun fungsi untuk membuat mereka mempunyai nilai yang tetap. Opsi-opsi berikut adalah benar *kecuali*

a. const Time noon(12, 0, 0); **b. int setData() const;**

c. int getData() const; d. void print() const;

17. *Interface* suatu klas adalah

a. data anggota b. prototipe fungsi anggota

**c. implementasi fungsi anggota** d. data anggota dan prototipe fungsi anggota

18. Opsi berikut yang *tidak dapat di-overload* adalah

a. contructor **b. destructor** c. operator d. function

19. OOP diklaim menggunakan model *dynamic* dalam pemrograman. Klaim ini didasarkan pada adanya komunikasi antara

**a. object dengan object** b. object dengan class

c. class dengan object d. class dengan class

20. Fungsi-fungsi yang dapat diakses o*bject constant* adalah :

a. constructor dan destructor

b. destructor dan fungsi anggota constant

c. fungsi anggota constant dan contructor

**d. constructor, destructor dan fungsi anggota constant**

**Essay** (60 poin)

1. *(15 poin)* Ada sejumlah karakteristik yang membedakan OOP dengan *procedural programming* diantaranya adalah ADT, *inheritance*, *dynamic binding* yang tidak lain adalah *true polymorphism* dan *generic programming*, Tiga topik terakhir belum kita pelajari sehingga dalam soal ini kita hanya fokus pada ADT.

a. Apa yang dimaksud dengan ADT?

b. Mengapa kita memerlukan ADT?

e. Apa yang kamu ketahui tentang relasi antara ADT, *encapsulation* dan *class*. Berikan contohnya.

2. *(15 poin)* *Reference* adalah salah satu tipe data baru di C++ yang sebelumnya tidak ada di C.

a. Apa yang dimaksud dengan *reference* di C++?

b. Bagaimana kita mendeklarasikan tipe reference di C++?

c. “... reference harus diinisialisasi sedangkan pointer tidak perlu diinisialisasi ...” Apa komentar Anda tentang pernyataan ini?

3. *(15 poin)* Ada tiga pendekatan fungsi untuk mengimplementasikan tugas identik untuk tipe data berbeda yaitu *naïve approach*, *function overloading* dan *function template*. *Function template* akan kita temui nanti di UAS sedangkan *naïve approach* jarang kita temui karena *difficult to keeping track of multiple function names* dan *lead to programming errors* . Dengan alasan ini maka kedua metode ini tidak layak jika diujikan disini.

a. Apa yang dimaksud dengan *function overloading* di C++ dan apa manfaatnya?

b. Bagaimana kita membedakan satu fungsi dengan fungsi lain di *function overloading*?

c. Mengapa *function overloading* dengan *return type* berbeda tidak dibolehkan?

4. *(15 poin)* C++ membedakan *passing by address* dan *passing by reference*.

a. Tulislah prototipe passing by address dan passing by reference fungsi *swap()* untuk menukar dua nilai variabel

b. Tulislah definisi kedua prototipe fungsi di atas.

c. Tulislah fungsi *main()* untuk memanggil kedua definisi fungsi di atas.

SOAL PG

1. D 6. D 11. B 16. B
2. D 7. C 12. A 17. C
3. D 8. C 13. B 18. B
4. C 9. B 14. A 19. A
5. B 10. B 15. C 20. D

SOAL URAIAN

1. a. ADT adalah pengelompokan data dan logically related method dalam suatu entitas

b. Encapsulasi adalah mekanisme untuk membentuk ADT

c. Class adalah implementasi dalam bahasa pemrograman

1. a. reference adalah alias atau sinonim suatu variable

b. int& r = n; // r adalah alias dari n, both have same value and same address

c. reference harus diinisialisasi saat deklarasi sedangkan pointer tidak perlu diinisialisai saat deklarasi

1. a. Functions overloading adalah fungsi-fungsi dengan tugas identik yang mempunyai nama yang sama tetapi berbeda prototipenya.

b. Dibedakan menurut three function signatures yaitu beda tipe parameter , beda banyak parameter dan beda urutan parameter

c. Syntax Error

1. a . void swap (int\* , int\*); void swap (int&, int&);

b. void swap (int\* x, int\* y) { void swap (int& x, int& y) {

int\* temp = x; int temp = x;

x = y; x = y;

y = temp; y = temp;

} }

C. void main() { void main() {

int a = 4, b = 7; int a = 4, b = 7;

cout << a << b << endl; cout << a << b << endl;

swap(&a, &b); swap(a,b);

cout << a << b << endl; cout << a << b << endl;

} }

**I AM PROUD AS A BINUSIAN**