**KEY FINAL EXAM** [30 questions, variable point on each, Total 50 points, 110 min]

WRITE YOUR MAJOR CODE, NAME and LAST FOUR DIGITS OF ID:

Presume the following two lines exist in every code.

#include <iostream>

using namespace std;

**Q1.** Which one is the correct sequence about the following fundamental data types? [2]

unsigned char

char32\_t

unsigned long long int

void

a. invalid, invalid, valid, valid

b. invalid, valid, valid, valid

c. valid, valid, valid, valid

d. valid, valid, valid, invalid

**ANS-1. b.** invalid, valid, valid, valid

*Look at the table in “variables” of cplusplus.com*

**Q2.** Underscore ( \_ ) is a valid starting character for identifier: *TRUE / FALSE*[1]

**ANS-2. *TRUE***

**Q3.** For output purpose, ‘\n’ , “\n”, and endl mean *different* things: *TRUE / FALSE* [1]

**ANS-3. *FALSE***

**Q4.** *Fill in*:Literals are used to express particular values within the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of a program. [2]

a. source code

b. object code

c. both of the above

d. none of the above

**ANS-4. a.** source code

*Look at “constants” in cplusplus.com*

**Q5.** What is the output of the following code? [2]

int main ()

{ int a=1, b=3;

 a += b;

 ++a;

 cout << a << b;

}

a. 23

b. 44

c. 53

d. none of the above

**ANS-5: c**

First a=1, then, a=1+3=4, and then a=4+1. b remains same, 3. Output is concatenation of *a* and *b* values.

**Q6.** Which one is correct below? [1]

a. #define NEWLINE ‘\n’

b. #define ‘\n’ NEWLINE

c. #define NEWLINE \n

d. #DEFINE newline “\n”

**ANS-5. a.**

**Q7.** What will the following code print? Choose all correct answers from below. [2]

int main ()

{

 long number = 3;

 cout << factorial(number);

 return 0;

}

long factorial(long x)

{

 long x;

 x = x \* factorial(x-1);

 return x;

}

a) Will not compile.

b) 6

c) 3

d) Will not terminate.

**ANS-7. a.**

Will not compile, because

long factorial(long)

is not declared before it is used in int main().

**Q8.** Choose one of the answers from below.  [2]

#include <iostream>

#include <typeinfo>

using namespace std;

template <class T, class U> bool are\_equal(T a, U b)

{

 return (typeid(a)==typeid(b));

}

int main ()

{

 if (are\_equal(10,10.0))

 cout << "x and y are equal\n";

 else

 cout << "x and y are not equal\n";

 return 0;

}

*a) Code will not compile.*

*b) It will output “x and y are equal”*

*c) It will output “x and y are not equal”*

**ANS-8.** c

**Q9.** What will be the value printed by the program? [2]

int foo(int y);

int main()

{ int a = 4;

 cout << foo(a) << endl;

 return 0;

}

int foo(int x)

{ return --x;

}

**a.** 3

**b.** 4

**c.** 5

**d.** undefined

**ANS-9.** a. 3

 The trick here is, we change the parameter names from the function declaration to function definition. This is legal, and only the parameter names in the function definition are used.

**Q10.** What will be the values printed by the program? [2]

int foo(int &x)

{ return ++x;

}

int main()

{ int x = 4, y=9;

 y = foo(x);

 cout << foo(y);

 return 0;

}

a. 5

b. 6

c. 11

**ANS-10. b**

**Q11.** What is the output from the following code? [2]

int main ()

{

 int p[5] = {4,5,6,7,8};

 cout << ++\*p << ' ';

 cout << ++(\*p) << ' ';

 cout << \*p << ' ';

 cout << \*(p+1);

}

a) 5 7 6 5

b) 5 6 6 5

c) 6 9 10 7

d) 6 9 10 8

**ANS-1. b.** 5 6 6 5

**Q12.** What is the output from the following code? [2]

int main ()

{ int x = 2;

 *switch* (x) {

 *case* 1: cout << "1 ";

 *case* 2: cout << "2 ";

 *case* 3: cout << "3 ";

 *break*;

 *default*:

 cout << "not 1, 2 or 3";

 }

}

a. 1

b. 2

c. 2 3

d. 1 2 3

**ANS-12. c**

**Q13.** Identify from below which one is NOT a cause any compilation error in the following program. [2]

void increment\_all (const int\* start, const int\* stop)

{

 int \* const current = start;

 while (current != stop) {

 ++(\*current); // increment value pointed

 ++current; // increment the pointer

 }

}

void print\_all (int\* start, int\* stop)

{

 int \* const current = start;

 while (current != stop) {

 cout << \*current << '\n';

 ++current; // increment pointer

 }

}

int main ()

{

 int numbers[] = {10,20,30};

 increment\_all (numbers,numbers+3);

 print\_all (numbers,numbers+3);

 return 0;

}

a) In function *increment\_all* the variable *start* is *const int\** in argument but the variable *current* is declared as *int \* const* but initialized with *start.*

b) In function *increment\_all* the variable *current* is declared as *int \* const* but attempted to be changed with *current~~++~~*.

c) In function *print\_all* declaration of the pointer variable *current* should not be constant as the address is being changed.

d) In main function *numbers* array size is not declared but initialized with three elements.

**Ans-13. d.**

All of a, b, and c will produce errors: In function *increment\_all* there should be no const as values and pointers, both are being changed within the loop; and

in function *print\_all* declaration of the pointer variable *current* should not be a constant as the address is being changed.

(This is from the fifth running code on <http://www.cplusplus.com/doc/tutorial/pointers/>)

**Q14.** In C++ const is available just for the purpose of making a programmer aware that the variable should not be changed: *TRUE / FALSE* [1]

**ANS-14. *FALSE***

**Q15.** A struct type may not have any method within it: *TRUE / FALSE* [1]

**ANS-15. *FALSE***

**Q16.** You have a class called *Image*. Choose one answer from below to dynamically allocate an array called *video* of 100 images (assume the Image class has no explicit constructor). [2]

a. Image \* video = new Image[100];

b. vector<Image> video(100);

c. array<Image, 100> video;

d. b or c only

e. Any one of the above a, b or c

**ANS-16: e.**

**Q17.** What is the destructor method’s name that should be written for filling in the blank. [2]

class Example {

 string\* ptr;

 public:

 // constructors:

 Example() : ptr (new string) {}

 Example(const string& str) : ptr(new string(str)) {}

 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ () {delete ptr;} // destructor

 // access content:

 const string& content() const {return \*ptr;}

};

a. void ~Example

b. ~Example

c. void Example~

d. Example~

**ANS-17. b.** ~Example

**Q18.**Mark correct ways to construct the object myexample below with initial values in the blank space.

[2]

class Example {

 int x, y;

 public:

 Example (int a, int b) {x=a; y=b;}

};

int main()

{ Example \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ~~)~~;

// whatever else code …

}

a. myexample = (100, 50);

b. myexample (100, 50);

c. myexample :: (100, 50);

d. myexample {100, 50};

**ANS-1. b & d both are correct answers.**

**Q19.** What is/are the output of the following code? [2]

int main ()

{

 int \* p;

 p= new (nothrow) int[5];

 for (int n=0; n<7; n++)

 cout << p[n] << ", ";

 delete[] p;

 return 0;

}

a. 0, 0, 0, 0, 0, 0, 0,

b. 0, 1, 2, 3, 4, 5,

c. 0, 1, 2, 3, 4, 5, undefined value

d. All undefined values (on most compilers)

**ANS-19. d.**

**Q20.** Answer following *three* questions on the code below by choosing yes/no. [3]

*int* divide (*int* a, *int* b=2)

{

 *int* r;

 r=a/b;

 *return* (r);

}

*int* divide (*int* a)

{

 *int* r;

 r=a/3;

 *return* (r);

}

*int* main ()

{

 cout << divide (12) << '\n';

 *return* 0;

}

a. Will the code compile? *YES / NO / NOT-APPLICABLE*

b. Will the code crash at run time? *YES / NO / NOT-APPLICABLE*

c. Which function will be called from the main()? *FIRST / SECOND / NOT-APPLIABLE*

**ANS-20. a. *NO b. NOT-APPLICABLE c. NOT-APPLICABLE***

**Q21.** The following linked list contains the nodes a, b, and c in the order shown below. Arrows represent the "next" pointer of each node. [2]

head -> [a] -> [b] -> [c] -> nullptr

If we use the following code to remove node [b] from the list while keeping all other nodes in the list:

a->next = b->next;

delete(b);

Which of the statements below is correct?

* the following statement is needed at the beginning

b->next = nullptr;

* the following statement is needed afterwards

b = nullptr;

* These statements work correctly
* These statements will cause a memory leak

**Q22.** What is the output from the following code? [2]

class Base

{

 public:

 void func1()

 {

 cout<<"Base class's func1()"<<endl;

 }

 virtual void func2()

 {

 cout<<"Base class's func2()"<<endl;

 }

};

class Derived :public Base

{

 public:

 void func1()

 {

 cout<<"Derived class's func1()"<<endl;

 }

 void func2()

 {

 cout<<"Derived class's func2()"<<endl;

 }

};

int main()

{

 Base \*obj = new Derived();

 obj->func1();

 obj->func2();

}

* Base class's func1()

Base class's func2()

* Base class's func1()

Derived class's func2()

* Derived class's func1()

Base class's func2()

* Derived class's func1()

Derived class's func2()

**Q23.** The following program will not compile as it contains error(s). Which of the options below will fix this or these error(s) (choose one). [2]

class Base

{

 public:

 virtual void func1()=0;

};

class Derived :public Base

{

};

int main()

{

 Derived obj;

 obj.func1();

}

* Remove the =0 from func1() declaration in *Base* class, so it becomes:

virtual void func1();

* Provide a body for func1() in *Base* class, so it becomes:

virtual void func1()=0

{

…//Any valid c++ code.

}

* Override func1() in *Derived* class
* Include the word “pure” in the declaration of func1() in *Base* class, so it becomes:

pure virtual void func1()=0;

**Q24.** What is the output from the following code? [2]

class Vehicle

{

 public:

 void ApplyBreaks()

 {

 cout<<"Vehicle Break";

 }

 virtual void ApplyHorns()

 {

 cout<<"Vehicle Horns";

 }

};

class Truck: public Vehicle

{

 public:

 void ApplyBreaks()

 {

 cout<<"Truck Break";

 }

};

class Car:public Vehicle

{

 public:

 void ApplyHorns()

 {

 cout<<"Car Horns";

 }

};

int main()

{

 Truck t;

 Car c;

 Vehicle \* v1 = &t;

 Vehicle \* v2 = &c;

 v1->ApplyBreaks(); cout << '\n'; v2->ApplyHorns();

}

* Vehicle Break

Vehicle Horns

* Vehicle Break

Car Horns

* Truck Break

Vehicle Horns

* None of the above

**Q25.** Abstract base class is a base class that has: [2]

a. All methods other than constructors and the destructor are virtual.

b. At least one method that is virtual.

c. All methods other than constructors and the destructor are pure virtual.

d. At least one method that is pure virtual.

**Q26.** The following linked list contains the nodes *a, b,* and *c* in the order shown below. Arrows represent the "next" pointer of each node. [2]

head -> [a] -> [b] -> [c] -> nullptr

Which one of the following codes will exchange node [b] and [c] in the list:

* [b]->next =nullptr; [a]->next =c; [c]->next =b;
* [a]->next = c; [c]->next =b; [b]->next =nullptr;
* [b]->next =nullptr; [c]->next =b; [a]->next =[b]->next;
* [b]->next =[a]->next; [c]->next =b; [a]->next =[b]->next;

NOTE: the first one works but is not elegant. Both answers will be accepted.

**Q27.**  Friend relationships may be inherited: *TRUE / FALSE* [1]

**Q28.** Constructors may be inherited: *TRUE / FALSE* [1]

**Q29.** Constructors may be overloaded: *TRUE / FALSE* [1]

**Q30.** Operator [] may be overloaded: *TRUE / FALSE* [1]