# *CS2433 Homework 2*

*CS18BTECH11001*

*February 24, 2020*

*This document is generated by LaTeX*

## ***Chapter 8 : Control Flow and Abstraction***

### ***Check your Understanding (CYU)***

***3. Describe how to maintain the static chain during a subroutine call.***

***A.*** *Maintaining the static frame must be performed by caller as it's depends on the lexical nesting depth of the caller. In the standard approach the caller computes the callees static link and pass it as an extra, hidden parameter. Two subcases arise:*

1. *if the callee is nested (directly) inside the caller, the callees static link should refer to the callers frame. Therefore, the caller passes its own frame pointer as the callees static link.*
2. *if the callee is k≥0 scopes outward", all scopes that surround the callee also surround the caller. Then the caller dereferences its own static link k times and passes the result as the callees static link.*

***5. What are the purposes of the stack pointer and frame pointer registers? Why does a subroutine often need both?***

***A.*** *The stack pointer register contains the address of either the last used location at the top of the stack, or the first unused location, depending on convention.*
*The frame pointer register contains an address within the frame.*
*The frame pointer is used for debugging.*
*If a new element is about to be added to the stack, stack pointer can be used to find the unused space. Therefore, a subroutine often need both.*

***6. Why do RISC machines typically pass subroutine parameters in registers rather than on the stack?***

***A.*** *As the register sets were significantly smaller and memory access was significantly faster (in comparison to processor speed), and the available instructions on a RISC machine are typically much simpler, RISC machines uses registers for passing parameters to subroutine than on the stack.*

***8. If work can be done in either the caller or the callee, why do we typically prefer to do it in the callee?***

***A.*** *The Callee can assume that there is nothing of value in any of the caller saved register set and as the Compiler uses callee-saves registers for local variables and other long-lived ones, and caller-saves set for transient values, which wont be needed across calls, we prefer to do the work in the callee.*

***9. Why do compilers typically allocate space for arguments in the stack, even when they pass them in registers?***

***A.*** *As the allocation of memory for the arguments in the stack can be done in the similar manner as that of other parameters(no special way required), Compilers allocate sapce even they pass them in registers.*

***11. How does an in-line subroutine differ from a macro?***

***A.*** *The in-line subroutine differs from macro in the way the compilers handles it.*
*A compiler parses in-line functions where as macros are expanded by the preprocessor.*

**19. Give an example in which it is useful to return a reference from a function in C++.**

**A.** *References are used in I/O opeerations.*
*Example : left-shift(<<) and right-shift(>>) operators return a reference to their first argument, which can then passed as an argument to the next << or >> operations.*

**22. What are default parameters? How are they implemented?**

**A.** *An optional parameter(not necessary) that can be passed by the caller, which if missing loads a default value(by the compiler) is known as default parameter.*
*Implementation : In any call, if the actual parameters are missing, the compiler pretends as if the defaults had been provided and loads them into registers or pushes them onto the stack by generating a calling sequence.*

**24. Explain the value of variable-length argument lists. What distinguishes such lists in Java and C# from their counterparts in C and C++?**

**A.** *For a function having an argument list of variable-length, a programmer must have to use a collection of standard routines within the body to access the extra arguments in C and C++. In Java and C#, which ensures type safety by trailing all parameters to share a common type which makes them differ from C and C++.*

**27. How does a generic subroutine differ from a macro?**

**A.**

| ***Macro*** | ***Generic Subroutine*** |
|---|---|
| *Macro can be called only in the program it is defined.* | *ubroutine can be called from other programs also.* |
| *Macro can have maximum 9 parameters.* | *Can have any number of parameters.* |
| *Macro is used when same thing is to be done in a program a number of times.* | *Subroutine is used for modularization.* |

**30. What does it mean for a generic parameter to be constrained? Explain the difference between explicit and implicit constraints.**

**A.** *We know that generic is an abstraction. As any abstraction has to follow the property that all of it's instances should able to provide all the information that user of abstraction must know, languages constrain generic parameters.*
*Implicit constraint : To use a generic class we must create an instance to it.*
*Explicit constraint : An extra generic parameter.*

**35. Explain how to implement exceptions in a way that incurs no cost in the common case (when exceptions dont arise).**

**A.** *Implementation : Create a table with starting address of block, corresponding handler for each entry and maintain the table in sorted order using the starting addresses as key.*
*In the case of Exception, during the run-time, the system performs a binary search to find the handler of the current block using the program counter as key.*
*Therefore, for a common case(when exceptions don't araise) there will be no binary search and incurs no cost.*

**39. Summarize the shortcomings of the setjmp and longjmp library routines of C.**

**A.** *The main feature of these routines is to provide a way that deviates from standard call and return sequence. This is mainly used to implement exception handling in C. setjmp can be used like try (in languages like C++ and Java). The call to longjmp can be used like throw (longjmp() transfers control to the point set by setjmp()).*

**40. What is a volatile variable in C? Under what circumstances is it useful?**

**A.** *A variable whose value in the memory can be changed spontaneously is called volatile variable. If a handler needs to see changes to a variable which can be modified by the protected code, the programmer must include volatile keyword in the variable's declaration.*

**42. What is the difference between a coroutine and a thread?**

**A.** *In coroutine, the instructions are executed sequentially one after the other.*
*In a thread, multiple instructions can be processed sequentially.*

***46. What is discrete event simulation? What is its connection with coroutines?***

***A.*** *A discrete event simulation is a model in which the system operates as a (discrete) sequence of events in time. It is one of the appilication of coroutine.*

***47. What is an event in the programming language sense of the word?***

***A.*** *An event is an action which occurs outside the program for an unpredictable time, in which a process needs to respond.*

***Exercises***

***8.3 Using your favorite language and compiler, write a program that can tell the order in which certain subroutine parameters are evaluated.***

***A.*** *Example program in C language :*

```
#include <stdio.h>
int dir(char* str) {
printf("%s \n",str);
return 0;
}
void fun(int a,int b,int c) { }
int main() {
fun(dir("left"),dir("middle"),dir("right"));
return 0;
}
```

*from the output, we can able to know the order in which the parameter evaluation is done in the subroutines.*

***8.4 Consider the following (erroneous) program in C:***

```
void foo() {
int i;
printf("%d ", i++);
}
int main() {
int j;
for (j = 1; j <= 10; j++) foo();
}
```

***Local variable i in subroutine foo is never initialized. On many systems, however, the program will display repeatable behavior, printing 0 1 2 3 4 5 6 7 8 9 . Suggest an explanation. Also explain why the behavior on other systems might be different, or nondeterministic.***

***A.*** *It occurs, if the function starts from zero in the first iteration(it may happen when the activation record of foo is used for the first time and the space where the stack gets created is filled with zeroes initially which is usually done by the operating system) and the present instance inherit the value of i from the previous instance(it occurs when all the activation records of foo occupy the same space and only if the space for the stack is not used for anything else).*
*As this is the case in most of the systems, the output will be 0 1 2 3 4 5 6 7 8 9. If this case is not satisfied by any of the systems then the output will be nondeterministic.*

***8.8 Consider the following subroutine in Fortran 77:***

```
subroutine shift(a, b, c)
integer a, b, c
a = b
b = c
end
```

***Suppose we want to call shift(x, y, 0) but we dont want to change the value of y . Knowing that built-up expressions are passed as temporaries, we decide to call shift(x, y+0, 0) . Our code works fine at first, but then (with some compilers) fails when we enable optimization. What is going on? What might we do instead?***

***A.*** *When we enable compiler optimization, y+0 will be considered as y and the value of y will be changed when the subroutine is being called. We can call shift(x,y,y) for the value of y not to be changed.*

***8.14 Consider the following declaration in C:***

$$double(*foo(double(*)(double, double[]), double))(double, ...);$$

***Describe in English the type of foo .***

***A.*** *By applying the spiral rule, we get foo as a function having 2 arguments and a return value.*
*Argument 1 : A pointer to function having a double and an array of doubles as arguments and return a double*
*Argument 2 : A double*
*Return : A pointer to a function having double and anything as arguments and return a double.*

***8.15 Does a program run faster when the programmer leaves optional parameters out of a subroutine call? Why or why not?***

***A.*** *The program containing optional parameters is same as program in which the parameters have their default values defined explicitly. So, the program run faster.*

***8.29 Describe a plausible implementation of C++ destructors or Java try . . . finally blocks. What code must the compiler generate, at what points in the program, to ensure that cleanup always occurs when leaving a scope?***

***A.*** *For implementing a C++ destructor or java try, the compiler adds some code in the scope of the object. This code will ensure required cleanups by jumping to the required address which always occurs in the scope. So we can ensure that clenup occurs before leaving the scope.*

## *Chapter 9 : Object Orientation*

### *Check your Understanding (CYU)*

***1. What are generally considered to be the three defining characteristics of object-oriented programming?***

***A.*** *The three defining characteristics of Object-oriented programming are :*

1. *Encapsulation*
2. *Inheritance*
3. *Polymorphism*

***3. Name three important benefits of abstraction.***

***A.*** *The three inportant benefits of Abstraction are :*

1. *Reduction in conceptual load.*
2. *Fault containment is provided.*
3. *Significant degree of independence among program components*

***6. What is the purpose of the private part of an object interface? Why is it required?***

***A.*** *The purpose of the private data type is hide and is visible only to that class. It is used to protect important data regarding an object.*

***7. What is the purpose of the :: operator in C++?***

***A.*** *Scope resolution operator(::) is used to identify the class of the header to which the method belongs to.*

***10. What are constructors and destructors?***

***A.*** *Constructor : Initialisation of an object after it's declaration can be done using the subroutine called constructor.*
*Destructor : Destroying the object when it's use is no longer required is known as destructor.*

***14. Explain the significance of the this parameter in object-oriented languages.***

***A.*** *In Object-oriented programming language this is the reciever parameter in an operation - an argument passed at the time operation is invoked.*

***16. Explain the distinctions among private , protected , and public class members in C++.***

***A.*** *Private members are accessible within the same class in which they are declared and can also be accessed by friend function.*
*Protected members are accessible within the same class and within the derived/sub/child class but can be accessed by friend function.*
*A public member is accessible from anywhere outside the class but within a program.*

***20. How do inner classes in Java differ from most other nested classes?***

***A.*** *Each instance of inner clas nust belong to instance of outer class where as a nested class can use arbitary number of its surrounding classes.*

***22. What are extension methods in C#? What purpose do they serve?***

***A.*** *Extension methods are a special kind of static method that enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type.*

***23. Does a constructor allocate space for an object? Explain.***

***A.*** *A constructor doesn't allocate space for an object instead it initialises the space that has been already allocated.*

***25. Why is object initialization simpler in a language with a reference model of variables (as opposed to a value model)?***

***A.*** *If initialization of object is done without reference, a new copy of it will be created which causes wastage of CPU memory and time.*

***28. Summarize the rules in C++ that determine the order in which constructors are called for a class, its base class(es), and the classes of its fields. How are these rules simplified in other languages?***

***A.*** *Rules :*

1. *Base classs constructor will be invoked first and then the derived classs constructor will be invoked.*
2. *In multiple Inheritace, the base classs constructors are called in the order of inheritance and then the derived classs constructor.*

***29. Explain the difference between initialization and assignment in C++.***

***A.*** *Giving value to an already present instance is called as assignment where as creating an instance with a particular value ois called initialization.*

***31. Explain the difference between dynamic and static method binding (i.e., between virtual and nonvirtual methods).***

***A.***

| ***Static Binding*** | ***Dynamic Binding*** |
|---|---|
| *The binding which can be resolved at compile time by compiler is known as static or early binding* | *In Dynamic binding compiler doesnt decide the method to be called.* |
| *Methods cannot be overridden* | *Overridding is the best example* |
| *Binding will be completed in compile time* | *Binding is delayed to run-time* |
| *private, final and static members use static binding* | *virtual methods use dynamic binding* |
| *Overloaded methods are resolved* | *Overridden methods are resolved* |

***35. Explain the connection between dynamic method binding and polymorphism.***

***A.*** *The Run-time polymorphism is called Dynamic Binding.*

***40. What is an abstract (deferred) class?***

***A.*** *A class which cannot be instantiated but can be subclassed is known as Abstract(deferred) class.*

***43. Explain the importance of virtual methods for object closures.***

***A.*** *Virtual methods support run-time polymorphism in which redefinition of the function can be done in the derived classes and the compiler will decide which function to be called. This is used when we want to create a method with same name(reducing usage of multiple function names).*

***Exercises***

***9.14 Compare Java final methods with C++ nonvirtual methods. How are they the same? How are they different?***

***A.*** *Similarities :*

i. *These methods can't be overridden.*
ii. *There is no need of dispatching the method call by the run-time system.*

*Differences :*

| ***Java Final Methods*** | ***C++ nonvirtual methods*** |
|---|---|
| *These methods are Dispatched* | *These methods are not dispatched* |
| *Can override methods in their classes and superclasses* | *Cannot override anything* |

***9.17 What happens to the implementation of a class if we redefine a data member? For example, suppose we have:***

```
class foo {
public:
int a;
char *b;
};
...
class bar : public foo {
public:
float c;
int b;
};
```

***Does the representation of a bar object contain one b field or two? If two, are both accessible, or only one? Under what circumstances?***

**A.** *The representation of a bar object will contain only one field and this is of derived class only. Only one can be accessed. Both can be accessed individually using their class name(foo::b).*

***9.21 If foo is an abstract class in a C++ program, why is it acceptable to declare variables of type foo* , but not of type foo ?***

**A.** *Since abstarct classes can't be instanstiated, it is not acceptable to declare variables of type foo. As the pointer of foo (foo*) can be used to point the classes that are not abstract and are inherited from foo, it is acceptable to declare variables of type foo*.*