RGPVNOTES.IN

Subject Name: **Principles of Programming Languages**

Subject Code: **IT-5002**

Semester: **5th**

Unit 4:

Abstraction

- Abstraction allows one to collect instances of entities into groups in which their common attributes need not be considered.
- Two kinds of abstractions in programming languages are process abstraction and data abstraction.
- The concept of process abstraction is one of the oldest. All subprograms are process abstractions because they provide a way for a program to specify that some process is to be done, without providing the details of how it is to be done.
- Process abstraction is crucial to the programming process. The ability to abstract away many of the details of algorithms in subprograms makes it possible to construct, read, and understand large programs.
- All subprograms, including concurrent subprograms, and exception handlers, are process abstractions

Encapsulation

- Encapsulation is a grouping of subprograms and the data that they manipulate
- An encapsulation provides an abstracted system and a logical organization for a collection of related computations
- They are often placed in libraries and made available for reuse in programs other than those for which they are written

Introduction To Data Abstraction

- An abstract data type is simply an encapsulation that includes only the data representation of one specific data type and the subprograms that provide the operations for that type
- An instance of an abstract data type is called an object
- Object-oriented programming is an outgrowth of the use of data abstraction

Static Storage Management

It is the simplest form of allocation i.e an allocation which remains fixed throughout the execution.

Properties of static storage management:-

- Storage for all variables allocated in static block.
- Allocation can be done by translator.
- Memory reference can be calculated during run time.
- Subprogram variables uses space even subprogram never called.
- Recursion is not possible.
- All storage known at translator time and memory reference is calculated.
- Activation records are directly associated with code segment.
- Procedure call and return straight forward.

Advantage

Not time or space is expanded for storage management during execution.

The translator can directly generate values address for all data item.

Disadvantage

In compatible with recursive subprograms or any data structure whose size, is dependent on computed or input data.

Stack Based Storage Management

Stacks are used to hold information about procedures calling and arrange them in such a manner that when a procedure terminate, it will move to the context of the main program. They follow a run time protocol between caller and callee to save arrangements and return value on the stack. They also support recursive subprogram and that is a system controlled storage management.

Some language use stack to create local referencing environment which will vanish on the exit of the procedure.

A stack has two basic operations: **Push and Pop.**

- **Push**: adds a node to the top.
- **Pop**: Removes and return current top (LIFO).

The stack is implemented with a little more than just push and pop. The length of a stack can also returned as a parameter top can return the top element without removing it.

Advantages:

As data is added and removed by LIFO. So, allocation is simpler and faster.

Memory on the stack is automatically reclaimed with the function exits.

Disadvantage:-

A stack can be very small and is dynamically grow and shrink in size. So, sometimes allocating more memory on the stack than available can result in crash due to overflow.

Heap Storage Management:-

A heap is a block of storage with in which pieces are allocated and freed in any manner. Here the problems of storage allocation, recovery, compaction and reuse may be serve. The heap storage management is a collection of techniques, which needs to be run side by side.

There are two types of dynamic storage which needs to be co-exist in memory. They both grow and shrink throughout program execution.

Properties of Heap Storage Management

- Generally used as storage for object with unrestricted lifetime.
- Maintains a list of free space (free list).
- An allocation memory allocation finds space according to any of the method and mark it as used : 1) best fit 2) worst fit 3) first fit.
- On deallocation-memory manager marks it as free.
- Memory fragmentation - the memory is divided in small blocks during lifetime because for eg. when a block is allocated memory to then the size may be small as compared to free block that may again break.
- Garbage collection.

The **allocation** and **deallocation** can be done at arbitrary points. Heap storage management techniques are divided into two categories depending on whether the elements allocated are always of the same fixed size or of variable size.

Garbage collection

The basic principle of how a garbage collector works are:

- Determine what data objects in a program will not be accessed in future.
- After determination, will reclaim the resources used by those objects.

We sat that dangling reference is bigger problem of two; they both are associated with the freeing of storage. In dangling reference, the memory is freed too early and in garbage memory is freed too late and it is better to have garbage in a program rather than dangling reference. So, we avoid dangling and allow garbage to be created. Which creates problem when the heap is full so garbage collection occurs once heap is full and process is called rarely. So it is allowable for the procedure to be fairly costly. Two stages are involved.

Object Oriented Programming in small talk, C++, Java, C#, PHP, Perl

Language	Intended use	Imperative	Object-oriented	Functional	Procedural	Generic	Reflective	Event-driven
Smalltalk	Application, general, business, artificial intelligence, education, web	Yes	Yes	Yes	Yes	No	Yes	Yes
C++	Application, system	Yes	Yes	Yes	Yes	Yes	No	No

Java	Application, business, client-side, general, mobile development, server-side, web	Yes	Yes	Yes	Yes	Yes	Yes	Yes
C#	Application, RAD, business, client-side, general, server-side, web	Yes	Yes	Yes	Yes	Yes	Yes	Yes
PHP	Server-side, web application, web	Yes	Yes	Yes	Yes	No	Yes	No
Perl	Application, scripting, text processing, Web	Yes	Yes	Yes	Yes	Yes	Yes	No

Concurrency: Subprogram level concurrency

A task is a unit of a program, similar to a subprogram that can be in concurrent execution with other units of the same program. Each task in a program can support one thread of control. Tasks are sometimes called processes.

A task can communicate with other tasks through shared nonlocal variables, through message passing, or through parameters.

Synchronization is a mechanism that controls the order in which tasks execute. Two kinds of synchronization are required when tasks share data: **cooperation and competition**. Cooperation synchronization is required between task A and task B when task A must wait for task B to complete some specific activity before task A can begin or continue its execution. Competition synchronization is required between two tasks when both require the use of to access shared data location x while task B is accessing x, task A must wait for task B to complete its processing of x. So, for cooperation synchronization, tasks may need to wait for the completion of specific processing on which their correct operation depends, whereas for competition synchronization, tasks

may need to wait for the completion of any other processing by any task currently occurring on specific shared data.

Semaphores

A semaphore is a simple mechanism that can be used to provide synchronization of tasks. Although semaphores are an early approach to providing synchronization, they are still used.

To provide limited access to a data structure, guards can be placed around the code that accesses the structure. A **guard** is a linguistic device that allows the guarded code to be executed only when a specified condition is true. So, a guard can be used to allow only one task to access a shared data structure at a time. A semaphore is an implementation of a guard. Specifically, a **semaphore** is a data structure that consists of an integer and a queue that stores task descriptors. A **task descriptor** is a data structure that stores all of the relevant information about the execution state of a task.

Monitors

A monitor is an instance of a Class with the following properties:

- The execution of concurrent calls of methods are mutually excluded in time (in Java, this is indicated by using the synchronized keyword)
- The possibility of using within the body of methods the operations wait (terminating temporarily the mutual exclusion until some programmer-defined condition - depending on internal monitor variables and method parameters - is satisfied), and notify (executed just before terminating a method call - used to activate any waiting process that will again check its waiting condition which may have been modified by the process that executed the notify operation).

A semaphore is a special case of a monitor. In Java, each object instance (that inherits from the Object class) can be used as a monitor; the keyword synchronized indicates the need for mutual exclusion for a given method and the operations **wait and notify** (or **notifyal**l) are defined on any instance of Object.

Message passing

Message passing can be either synchronous or asynchronous. The basic concept of synchronous message passing is that tasks are often busy, and when busy, they cannot be interrupted by other units. Suppose task A and task B are both in execution, and A wishes to send a message to B. Clearly, if B is busy, it is not desirable to allow another task to interrupt it. That would disrupt B's current processing. The alternative is to provide a linguistic mechanism that allows a task to specify to other tasks when it is ready to receive messages.

A task can be designed so that it can suspend its execution at some point, either because it is idle or because it needs information from another unit before it can continue. This is like a person who is waiting for an important call. In some cases, there is nothing else to do but sit and wait. However, if task A is waiting for a message at the time task B sends that message, the message can be transmitted. This actual transmission of the message is called a rendezvous. A rendezvous can occur only if both the sender and receiver want it to happen. During a rendezvous, the information of the message can be transmitted in either or both directions.

Both cooperation and competition synchronization of tasks can be conveniently handled with the message-passing model.

Java threads, C# threads

Java threads	C# threads
To make a thread sleep: Thread.sleep(1000);	To make a thread sleep: Thread.Sleep(1000);

To create a thread: Make a class extend Thread class	To create a thread: Code a protected method.
or to create a thread: implement Runnable interface: Thread athread = new Thread(that_class);	Thread athread = new Thread(new ThreadStart(that_method));
To start a thread: athread.start();	To start a thread: athread.Start();
To kill a thread: athread = null;	To kill a thread: athread.Abort();
To interrupt a thread: athread.stop();	To interrupt a thread: athread.Interrupt();

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in