# Test di valutazione – Modulo 1

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Mara |
|  |  | Cognome | Barani |
|  |  | Data | 5/02/2021 |

Leggete attentamente ogni domanda e argomentare quanto più possibile fornendo anche degli esempi.  
ATTENZIONE: Le domande a risposta multipla possono contenere più risposte corrette.

1. *Spiegare le differenze tra memoria principale, secondaria e virtuale*

Le tipologie ti memoria si distinguono per capcità (quantità di informazioni che possono essere immagazzinate, in GigaByte di solito), volatilità (quanto tempo le informazioni inserite in memoria permangono), velocità (tempo di lettura e scrittura di informazioni) e costo in euro/dollorari (costo per ogni bit o byte di memoria).

La memoria principale è una memoria volatile, in cui le informazioni permangono fino allo spegnimento del computer, ha una capacità bassa e, per questo, una velocità alta e anche un costo alto. Inoltre l’accesso alla memoria principale non è sequenziale ma è random. Esempi di questo tipo di memoria sono la RAM e la Cache.

La memoria secondaria invece non è volatile, cioè le informazioni permangono anche allo spegnimento del computer, ha una capacità alta, una velocità bassa e un costo medio/basso. L’accesso a questa memoria è sequenziale, quindi è meno performante della memoria principale. Esempi di questa memoria sono il disco rigido e i CD/DVD.

Il concetto di memoria virtuale nasce nel momento in cui bisogna far girare nella RAM un’applicazione che ha una capacità più grande della RAM stessa. Quello che accade è che le istruzioni dell’applicazione vengono salvate nella memoria secondaria e le prime istruzioni che il processore deve compiere vengono copiate nella RAM. Una volta eseguite queste prime istruzioni, vengono eliminate e vengono copiate dalla memoria secondaria e quella principale il successivo set di istruzioni. grazie allo swap tra le due memorie, sembra che la RAM abbia una capacità maggiore di quanto ha in realtà, e questa viene chiamata memoria virtuale.

1. *Cosa significa che un linguaggio è tipizzato? Spiegare la suddivisione in tipi in C# e le relative caratteristiche.*

Un linguaggio è tipizzato quando è costituito da tipi. I tipi rappresentano i ruoli delle variabili all’interno del codice e la loro funzionalità, cioè che dato si può inserire nella variabile e la memoria che occupa.

Esistono dei tipi base in C# e dei relativi alias:

* Boolean (può essere true o false) - bool
* Int16, Int32, In64 (per numeri interi che occupano diverse quantità di memoria) – int per l’Int32
* Byte (per numeri interi senza segno che vanno da 0 a 255) - byte
* Decimal, Single, Double (per numeri reali che occupano diverse quantità di memoria) – decimal, float, double
* Char (per un singolo carattere) - char
* String (per un insieme di caratteri) – string

Si possono anche costruire direttamente dei tipi, cioè l’utente può definire lui stesso dei tipi, come le classi.

I tipi si dividono in Value Type e Reference Type. I value type contengono il dato e lo copiano nello stack, non possono essere nulli e possono essere istanziati. Al contrario I reference type contengono l’indirizzo all’area di memoria in cui si trova il dato, possono essere nulli e non possono essere istanziati. I tipi base sono di tipo valore, tranne le stringhe. Mentre stringhe e classis ono di tipo reference.

1. *Cos’è un sistema operativo? Dare una spiegazione delle funzionalità.*

Un sistema operativo è un insieme di software per la gestione delle componenti fisiche del computer, cioè della parte hardware. In pratica i software contengono le istruzioni in files che poi vengono eseguite dall’hardware.

Un sistema operativo ha come componente fondamentale il kernel, che gestisce le funzionalità del processore nascondendole alle applicazioni. In questo modo un applicazione non può sovrascrivere un’area di memoria destinata ad un’altra e può dividere la capacità della memoria in modo da permettere il funzionamento in contemporanea di più programmi (multi-tasking). Il sistema operativo, oltre a gestire le funzionalità del processore tramite il kernel, gestisce la memoria, l’interfaccia con l’utente, i dispositivi di Input/Output e la memoria.

* La memoria viene gestita dal kernel che ne permette il frazionamento per l’esecuzione contemporanea di più programmi. Inoltre il sistema operativo è quello che permette la gestione di più memorie, cioè permette il meccanismo di swap tra memoria principale e secondaria descritto alla domanda 1. La conservazione delle informazioni/istruzioni avviene nella memoria secondaria tramite una struttura gerarchica di files e cartelle, chiamata File System. Quindi il sistema operativo di occupa anche della gestione del File System.
* L’interfaccia può essere grafica (con le icone e le finestre, come quelle che vediamo sul Desktop), bash (da terminale, tutti i SO hanno una bash, ma non tutti hanno l’interfaccia grafica) o vocale (Siri/Cortana)
* I dispositivi di input e output sono per esempio tastiera e schermo. Questi hanno tempi di risposta diversi, quindi il sistema operativo cerca di rendere omogenei i tempi di risposta di questi dispositivi in modo che per l’utente avvengano contemporaneamente. Questo viene fatto tramite una zona di memoria chiamata buffer, che immagazina una certa quantità di comandi e poi li restituisce tutti ad uno stesso istante.

1. *Descrivere il funzionamento degli operatori logici e delle relative tabelle di verità.*

Gli operatori logici, come &&(and) e ||(or), permettono di confrontare due variabili e restituiscono un booleano, cioè vero o falso. Funzionano secondo le cosiddette “Tabelle di verità”, cioè tabelle in cui si hanno due variabili booleane, per esempio A e B, e a seconda dell’operatore che viene inserito tra queste due variabili e del valore delle variabili stesse, ritornerà vero o falso. Per esempio, supponiamo di fare un’operazione di unione di due variabili, quindi l’operatore logico corrispondente è l’OR. La corrispondente tabella di verità sarà data da:

* Se A=true || B=true 🡪 A unione B = true
* Se A=true || B=false 🡪 A unione B = true
* Se A=false || B=true 🡪 A unione B = true
* Se A=false || B=false 🡪 A unione B = false

Supponiamo ora di fare un’operazione di intersezione di due variabili, quindi l’operatore logico corrispondente è l’AND. La corrispondente tabella di verità sarà data da:

* Se A=true || B=true 🡪 A intersezione B = true
* Se A=true || B=false 🡪 A intersezione B = false
* Se A=false || B=true 🡪 A intersezione B = false
* Se A=false || B=false 🡪 A intersezione B = false

1. *Spiegare la differenza tra ricorsione e iterazione e le relative peculiarità.*

L’iterazione consiste nella ripetizione di un blocco di codice finché una condizione non viene soddisfatta. È quello che accade negli statement iterativi, con i cicli for e while per esempio.

La ricorsione consiste invece nel richiamare una certa funzione all’interno della funzione stessa, cioè uno statement in una funzione richiama la funzione stessa. Generalmente la ricorsione è meno performante dell’iterazione, a meno di un numero piccolo di ricorsioni da esequire, ma il vero problema è l’occupazione della memoria stack. Infatti, quando si fa una ricorsione, viene mantenuta attiva anche la chiamata alla funzione precedente. Quindi ad ogni ricorsione vado ad occupare nuovo spazio di memoria perché tutte le chiamate precedenti alla funzione sono ancora attive. Questo non accade nel caso dell’iterazione, perché una volta che la funzione è stata chiamata una volta, non rimane “appesa”, ma viene chiusa ad ogni iterazione.

1. *Dare una definizione di casting. Fornire degli esempi.*

Il casting è un’operazione di conversione di un tipo in un altro affine. Si può anche convertire in un tipo non affine ma questo può comportare una perdita di dati. La conversione varia se il tipo di variabile è value o ref.

Se la variabile è di tipo valore, il cast può essere implicito o esplicito. Il casting implicito avviene automaticamente, senza che neanche lo sviluppatore se ne accorga, e non c’è perdita di informazioni es:

* Byte b=5;
* Int i=b;

Nel casting esplicito invece può avvenire Perdita di informazioni, per esempio se il range della variabile di partenza è più grande della variabile di ritorno, ad esempio se vogliamo trasformare un numero intero maggiore di 255 in un byte. Per il casting esplicito esistono diversi metodi:

* Le parentesi tonde: int i=2980; byte b=(byte) i;
* Un metodo chiamato TryParse, che restituisce un booleano che mi dice se la conversione si può fare oppure no
* I metodo Parse e Convert, che però non fanno un check se la conversione può avvenire, e quindi si può avere una perdita di dati

1. *Spiegare cosa è una routine. Specificare le tipologie di routine e fornire degli esempi per ciascuna.*

Una routine è un insieme di istruzioni contenute in un blocco di codice che costituiscono un’unica entità e svolgono un’unica funzionalità. Possono essere richiamate in punti diversi del codice. Le routine possono avere dei parametri in input e restituire un valore di ritorno. Se la routine non restituisce un valore di ritorno viene chiamata procedura, altrimenti funzione.

Una routine è così costituita:

* Parole chiave (come public, per renderla visibile ovunque nel codice, e static)
* Tipo di ritorno (o void se non ritorna niente)
* Nome della routine
* Possibili parametri di input

Quando chiamaiamo una routine in una parte di codice utilizziamo la “firma”, cioè il nome + i parametri di input + il loro tipo.

I parametri di input possono essere di tipo value o di tipo ref, e possono essere passati come valori o referenze.

Esempio di proceura:

public static void Esempio(int i)

Esempio di funzione:

public static int Esempio()

1. *Descrivere la funzione del compilatore*

Il compilatore ha il compito di tradurre il file scritto nel linguaggio di programmazione dal programmatore, che viene chiamato anche file sorgente, in un file eseguibile, cioè in un file scritto in linguaggio macchina (cioè binario) comprensibile alla macchina.

1. *Descrivere le funzioni di un sistema di versionamento e le diverse tipologie*

Un Sistema di versionamento ha lo scopo di registrare nel tempo I cambiamenti di uno o più files. In particolare permette di:

* Ripristinare i files ad una versione precedente;
* Ripristinare un intero progetto ad una versione precedente;
* Revisionare le modifiche fatte nel tempo.

Esistono diverse tipologie di sistemi di versionamento:

* locali, cioè I files vengono salvati unicamente nel database del nostro computer. Se si sta collaborando ad un progetto si deve dare l’accesso al proprio pc agli altri utenti;
* centralizzati: le versioni del codice si trovano in un database su un server esterno, quindi sono accessibili da tutte le persone e gli utenti possono interagire e modificare il codice. Il problema è che le versioni si trovano tutte in un unico punto, il database esterno. Se il server ha dei problem il codice e le sue version non sono più accessibili;
* distribuiti: gli utenti non solo possono vedere la versione più recente del codice e lo storico, ma possono anche copiarli in locale sul proprio computer.

*Esercizio Pratico*

Creare una Console Application che gestisca i Task dell’utente.

Per Task viene inteso un oggetto che ha una descrizione, una data di scadenza e un livello di importanza (Basso, Medio, Alto).

L’utente può:

* Vedere i Task inseriti
* Aggiungere un nuovo Task
* Eliminare un Task
* Filtrare i Task per importanza

Requisiti Tecnici:

-Salvare i Task in un file

-Utilizzare adeguatamente il concetto di classe

-Dividere le funzionalità in relative funzioni e procedure

-Commentare

-Mettere una nomenclatura conforme

-Le date di scadenza devono essere posteriori o uguali rispetto alla data di inserimento

-Controllare l’input utente

Opzionale: Utilizzare Enum

Mettere il codice dell’esercizio in un Repository di GitHub.