# Test di valutazione – Modulo 1

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Elisa |
|  |  | Cognome | Posani |
|  |  | 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*

Un calcolatore ha diversi tipi di memoria in modo da poter gestire al meglio i vari processi che deve eseguire. La memoria principale è solitamente molto veloce, ma con una capacità minore. Ha un accesso random e non sequenziale, permettendo quindi l’elevata velocità. E’ una memoria molto volatile, quindi non si presta alla conservazione di file e processi ma solo alla loro temporanea esecuzione. Esempi di memoria principale sono la RAM e la cache.

La memoria secondaria invece è più lenta rispetto a quelle appena viste, a causa dell’accesso sequenziale e dell’elevata capacità. Questo la rende adatta allo stockaggio a lungo termine, ma non alla gestione dei processi in esecuzione. Esempi di memoria secondaria sono i dischi rigidi e i CD.

Per “aumentare” temporaneamente la capacità della memoria principale, solitamente questa “si appoggia” a quella secondaria attraverso un flusso di informazioni. Queste vengono salvate nella memoria secondaria e recuperate al momento del bisogno. Al termine del processo viene liberata anche la memoria virtuale oltre che quella principale.

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

Per gestire i diversi oggetti propri del linguaggio matematico, e non, sono stati ideati i tipi. Se una variabile è di un certo tipo, allora avrà delle caratteristiche proprie di quella categoria. Questo è necessario per la limitata memoria dei calcolatori, ottimizzando le risorse disponibili per descrivere solo le proprietà del determinato tipo. Ad esempio un numero intero, non avendo bisogno di parte decimale, può sfruttare tutti i bit assegnatigli per la parte intera. Oltre a questo, i tipi si differenziano per numero di bit. Se devo lavorare con numeri piccoli mi baseranno tipi con una bassa allocazione di memoria, ad esempio i byte, per non sprecare memoria. Sapendo che i miei numeri saranno molto grandi userò più memoria con un long int (Int64) ma con la sicurezza che i miei dati non saranno tagliati. Lo stesso vale per i numeri decimali, che possono essere salvati in variabili decimal, float e double. Infine se ho bisogno di un true/false, mi basta un bit, anche se potrei ricondurmi alla stessa condizione con un int, e posso usare un bool. Tra i tipi si trovano anche i datetime che hanno la formattazione migliore per memorizzare date e ore, i caratteri, ‘char’ e gli array di altri tipi.

Oltre ai tipi di variabile, di cui si è parlato fin ora, sono presenti i Reference Type, che possono essere più complessi, come stringhe, ArrayList o altre classi.

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

Un sistema operativo è un software fondamentalee nell’utilizzo di un calcolatore. Ha numerose funzioni, di cui la principale è svolta dal *kernel*: esso gestisce le funzionalità del processore, in modo da impedire l’interazione diretta tra quest’ultimo e software diversi dal sistema operativo. Inoltre garantisce il multitasking della macchina, attraverso un’esecuzione sicura delle applicazioni, evitando sovrascrizioni di memoria. Gestisce inoltre le periferiche di input e output, rendendone omogenea l’interazione. Questo viene fatto anche attraverso il buffer, un’area di memoria adibita alla temporanea memorizzazione dei comandi da periferiche esterne, in modo da sincronizzarle tra loro in base ai diversi tempi di risposta.

Anche l’interfaccia grafica è una prerogativa del sistema operativo, mostrando i contenuti delle memorie e le funzionalità del calcolatore attraverso icone, finestre ecc... Può essere presente anche un’interfaccia vocale per l’interazione con la macchina. Infine la bash è sempre presente e contiene tutte le funzionalità che potrebbero essere svolte dall’interfaccia grafica.

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

Gli operatori logici, and e or, sono operatori binari che prendono in input due booleani e come output un altro booleano. Solitamente vengono verificate delle affermazioni e poi viene analizzata la loro “interazione”. Con l’operatore logico AND, il risultato sarà vero solo se entrambe le affermazioni sono vere, falso in tutti gli altri casi. Con l’OR invece basta che una delle due, o entrambe, siano vere. In caso contario, quindi due false in input, si avrà falso come output. Di seguito sono riportate le tabelle di verità.

|  |  |  |
| --- | --- | --- |
| ***AND*** | ***F*** | ***T*** |
| ***F*** | *F* | *F* |
| ***T*** | *F* | *T* |

|  |  |  |
| --- | --- | --- |
| ***OR*** | ***F*** | ***T*** |
| ***F*** | *F* | *T* |
| ***T*** | *T* | *T* |

|  |  |  |
| --- | --- | --- |
| ***NOT*** | ***F*** | ***T*** |
|  | *T* | *F* |

Tra gli operatori logici vi è anche il NOT, che però è unario. Prende quindi in input solo una affermazione e restituisce il contrario.

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

All’interno di un codice si possono trovare situazioni in cui è necessario ripetere un’operazione fino a che non sia verificata una certa condizione. Questo può essere fatto per iterazione o per ricorsione. Il primo, realizzato mediante cicli while, for e foreach, inizia un processo all’inizio dell’iterazione e lo conclude alla fine. Se la condizione non è stata raggiunta, ricomincia un nuovo processo.

Per quanto riguarda la ricorsione, consiste nel richiamare se stessa da parte di una funzione. Può essere tail se ritorna esattamente la funzione stessa, o non se deve compiere altre operazioni nel return. Se per pochi passaggi questo metodo può rivelarsi molto vantaggioso, in termini di tempo bisogna considerare che ad ogni ricorsione la funzione precedente viene lasciata in sospeso, aprendone una nuova. Questo porta ad un veloce consumo dello stack, dando spesso problemi con la memoria. Inoltre più ricorsioni vengono effettuate, più tempo ci vorrà a completarle. Le iterazioni invece, essendo conclusive impiegano sempre lo stesso tempo per “giro”.

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

Il casting consiste nell’assegnare un tipo ad una varaibile. Può essere fatto in modo implicito, semplicemente attraverso un’uguaglianza tra variabili di tipo diverso. Questo non sempre è consentito e porta spesso ad errori. Un metodo più “sicuro” è il casting esplicito, in cui viene dichiarato esplicitamente il tipo a cui deve appartenere la variabile. Può essere fatto in diversi modi, tra cui una conversione (Convert, Parse, TryParse) o una dichiarazione del tipo attraverso (tipo) o .ToTipo.

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

Una routine, o funzione, è un insieme di istruzioni raggruppate in un unico blocco di codice. Queste, una volta completate le istruzioni possono concludersi senza nessun valore di ritorno (funzioni void) o ritornare un valore della tipologia dichiarata nel tipo di funzione (int, double, array, stringhe....). Inoltre la funzione può avere in input delle variabili che vengono utilizzate al suo interno. Possono essere obbligatorie o opzionali, sostituite in caso da un valore di default impostato. La routine può avere, oltre al tipo, altre caratteristiche da specificare al momento della dichiarazione (public, static, async...). L’importanza delle funzioni sta nel “pacchettizzare” le istruzioni in modo da rendere più fruibile e ordinato il codice. Inoltre se devo compiere una serie di istruzioni più volte si può semplicemente richiamare la funzione.

1. *Descrivere la funzione del compilatore*

I linguaggi di programmazione compilativi sono composti da una serie di istruzioni comprensibili allo sviluppatore. Questi sono stati ideati proprio per scostarsi dal linguaggio macchina e rendere la programmazione più facile ed accessibile. Il calcolatore però continua a leggere il linguaggio macchina. Attraverso il compilatore è possibile tradurre il linguaggio di programmazione in linguaggio macchina ed essere da essa compreso per eseguire le istruzioni.

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

Durante la stesura di un codice è possibile che si apportino modifiche che si rivelano poi improduttive. In particolare se si è in più di una persona a lavorare allo stesso progetto è possibile che delle modifiche apportate da uno sviluppatore non siano in linea con il progetto comune o che non si leghino con il lavoro degli altri. Per questo possono essere utili dei sistemi di versionamento, che salvano delle versioni intermedie del codice. Queste possono essere salvate in modo locale, sul dispositivo del singolo, ma non è ottimale per i lavori di gruppo. Il versionamento può essere salvato in un server esterno, nei Sistemi di Controllo Versione Centralizzati, a cui hanno accesso tutti i partecipanti. In caso di malfunzionamento del server però si possono riscontrare disagi Infine ci può essere un Sistema di Controllo Versione Distribuito, in cui ogni utente ha una repository locale, di cui può fare un backup in ogni momento.

*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.