Programmazione procedurale e ad oggetti - 7/01/2020

Nome: .....
Cognome: .....
Matricola: .....

1. Illustrate le principali funzioni per la gestione della memoria dinamica in C.

Sol. Si vedano le funzioni `malloc`, `calloc` e `free` (lucidi del corso e libro di testo)

2. Considerate alberi binari i cui nodi sono così definiti
`struct nodo{int k; struct nodo* sx; struct nodo* dx;}`
e progettate una funzione che accetta in ingresso un albero e restituisce il numero di nodi dell'albero aventi un solo figlio.

```
int  UniqueSon(struct nodo* t)
{if (t==null) return 0;
 if (t->sx==null) return 1+ UniqueSon(t->dx);
 else if (t->dx==null) return 1+ UniqueSon(t->sx);
      else return UniqueSon(t->sx)+UniqueSon(t->dx);
}
```

3. Presentate con degli esempi le principali direttive del preprocessore C.

Sol.
Si vedano le direttive `#include`, `#define`, `#undef`, `#ifdef`, `#ifndef` `#if`, `#elif`, `#else`, `#endif` (lucidi del corso e libro di testo).

4. Quali fattori di qualità del SW sono favoriti da un approccio orientato agli oggetti?

Sol. I principali fattori di qualità che traggono vantaggio dall?approccio OOP sono:

- Correttezza (comportarsi secondo le specifiche e i requisiti)
- Robustezza (capacità di funzionare anche in condizioni anomale)
- Estendibilità (facilità di adattamento del SW al cambiamento delle specifiche)
- Riutilizzo (possibilità di recuperare (porzioni di) un programma per nuove applicazioni)
- Compatibilità (facilità con cui programmi diversi interagiscono)

5. A quali vincoli è soggetto l'overloading di operatori in C++? È possibile effettuare l'overloading dell'operatore di chiamata di funzione? Se sì mostratene un esempio.

Sol. L'overloading di operatori è soggetto ai seguenti vincoli:

- non si può estendere il linguaggio introducendo nuovi operatori (ad es. ** per l'esponenziazione)
- non si può cambiare l'arietà degli operatori (ovvero il numero di operandi)
- non si può modificare la precedenza (ad esempio, `a = b + c*d` equivale sempre a `a = b + (c*d)`)
- non si può modificare l'associatività (es. `a-b-c-d` equivale sempre a `((a-b)-c)-d)`
- non si può modificare la semantica di un operatore rispetto ai tipi predefiniti

L'overloading dell'operatore di chiamata di funzione `()` è possibile e non corrisponde ad una diversa semantica cella chiamata di funzione. Quando un'operazione su un'oggetto assume rilievo (e frequenza d'uso) particolare tra tutte quelle disponibili, possiamo ricorrere all'overloading di `()`. Ad esempio, se in una classe `Punto` capita spesso di moltiplicare un punto per uno scalare scriviamo

```
class Punto {
public:
  ...
    void operator()(int n) { x=n*x;y=n*y; }
private:
    int x,y;
};
```

Dato quindi un punto `p`, l'espressione `p(4)` ha l'effetto di moltiplicare per 4 ascissa e ordinata di `p`.

6. Dite come si comporta il C++ nei riguardi della tipizzazione e del binding.

Sol. C++ è un linguaggio con tipizzazione statica, ovvero il tipo di un elemento sintattico è noto al momento della compilazione. Questo permette di effettuare dei controlli sulle espressioni al fine di intercettare in compilazione eventuali errori di tipo. Non è però possibile escludere il verificarsi di errori di tipo durante l'esecuzione (C++ non gode della *type safety* – si pensi ad un uso poco accorto delle conversioni di tipo). C++ supporta invece il binding dinamico. Il collegamento selettore-metodo che lo implementa può avvenire dinamicamente attraverso l'uso delle cosiddette `funzioni virtuali`. Un metodo invocato attraverso un puntatore porta all'esecuzione di un codice individuato solo in esecuzione. Ad esempio se `pt` è un puntatore a oggetti di una classe `Figure` che presenta un metodo virtuale `Stampa()`, l'espressione `p->Stampa()` porterà all'esecuzione del metodo `Stampa()` specifico per la classe effettiva di appartenenza dell'oggetto puntato, diciamo `X`, se `X` è una classe derivata da `Figure` che definisce una versione propria del metodo `Stampa()`. Tecnicamente, se una classe definisce una funzione virtuale, ogni oggetto di tale classe contiene un puntatore tramite il quale ottenere l'indirizzo preciso della funzione da eseguire a fronte della chiamata. Tale puntatore contiene l'indirizzo di inizio di una tabella (jump table) destinata a contenere gli indirizzi delle funzioni virtuali. Detto in altri termini, a fronte della chiamata `f()` il compilatore genera del codice che prevede un salto all'istruzione il cui indirizzo è contenuto al puntatore (nella jump table) associato alla funzione virtuale `f()`.