Programmazione procedurale e ad oggetti - 22/07/2019

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

1. Spiegate cosa avviene durante l'esecuzione della seguente funzione:

```
void what(void){
int* pt;
pt=(int*)calloc(10,sizeof(int));
pt++;
free(pt);return;}
```

Sol. Viene allocato dinamicamente un vettore di 10 interi e assegnato alla variabile `pt` l'indirizzo base. Tale puntatore assume poi l'indirizzo del secondo intero di tale vettore (++) ed è tale indirizzo ad essere fornito come argomento alla funzione `free`. Poichè questo valore non corrisponde ad un indirizzo restituito da una funzione per l'allocazione dinamica di memoria, si osserva un errore in esecuzione.

2. Definite una funzione che non fa nulla e che alla decima volta che viene chiamata si limita a terminare dopo aver stampato il messaggio `credito esaurito`.

Sol.

```
void nulla(void){
static int count=9;
if (count==0) printf("credito esaurito");
 else count--;
return;}
```

3. Illustrate gli strumenti che il `C` mette a disposizione per leggere gli argomenti all'interno del record di attivazione di procedura.

Sol. Si vedano le macro `va_start`, `va_arg` e il tipo `va_list` (lucidi del corso e libro di testo).

4. Cosa si intende per genericità o polimorfismo parametrico? Fornite un esempio di codice C++ che ne fa uso.

Sol. Per genericità si intende l'uso di tipi come parametri nella definizione di classi e funzioni. Come esempio si vedano le classi contenitore (ad es. la classe Stack - slide del corso) o una delle funzioni di ordinamento (ad es. bubblesort - slide del corso).

5. A quali vincoli è soggetto l'overloading di operatori in C++? Mostrate un esempio.

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

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

Un classico esempio è costituito dall'overloading dell'operatore di somma `+` per la classe dei numeri complessi `Complex`. Basta definire una funzione amica all'interno della classe,

```
friend Complex operator+(const complex& a,const complex& b)
{return  Complex(a.re+b.re,a.im+b.im);}
```

6. Considerato il seguente codice

```
class A {public:void f(void) {std::cout <<"f di A\n";};};
class B: public A {public:  void f(void) {std::cout <<"f di B\n";};};
class C: public B {public: virtual void f(void) {std::cout <<"f di C\n";};
    void g(void){f();};};
class D: protected C {public:  void f(void){std::cout <<"f di D\n";};};
class E: public D {public:  virtual void g(void)  {C::g();};};
int main(void){ B oBB; C oBC; D oBD; E oBE;
 A* ptA; B* ptB; C* ptC; D* ptD;
 ptA=&oBB; ptA->f();
 ptA=&oBC; ptA->f();
 ptB=&oBC; ptB->f();
 ptC=&oBD; ptC->f();
 ptD=&oBE; ptD->g();
 oBE.g();
 return 0;}
```

dite se ci sono istruzioni che danno luogo ad errori in compilazione (motivazione), e mostrate l'effetto dell'esecuzione delle chiamate di funzione (una volta eliminate eventuali istruzioni illegali).

Sol.

(`ptA=&oBB; ptA->f();`) Essendo `B` un sottotipo di `A`, è lecito assegnare a un puntatore ad un oggetto di classe `A` l'indirizzo di un oggetto di classe `B` (che può comportarsi come un oggetto di classe `A`). La funzione `f` chiamata tramite `ptA` è quindi la funzione `f` (non virtuale) presente nella classe `A`, con il messaggio `f di A` stampato in output.

(`ptA=&oBC; ptA->f();`) Come nel caso precedente, anche `C` è un sottotipo di `A`, quindi è lecito assegnare a un puntatore ad un oggetto di classe `A` l'indirizzo di un oggetto di classe `C` (che può comportarsi come un oggetto di classe `A`). La funzione `f` chiamata tramite `ptA` è quindi sempre la funzione `f` (non virtuale) presente nella classe `A`, anche se la classe `C` possiede una funzione `f` virtuale. Il messaggio stampato è `f di A`.

(`ptB=&oBC; ptB->f();`) La classe `C` è un sottotipo di `B`, quindi è lecito assegnare a un puntatore ad un oggetto di classe `B` l'indirizzo di un oggetto di classe `C` (che può comportarsi come un oggetto di classe `B`). Dato che nella classe `B` è presente una funzione `f` (non virtuale), tramite `ptB` viene quindi sempre chiamata la funzione `f` che stampa `f di B`, anche se la classe `C` possiede una funzione `f` virtuale.

(`ptC=&oBD; ptc->f();`) Poiché la classe `D` eredita da `C` in modalità protetta, un oggetto di classe `D` non rispetta l'interfaccia (servizi pubblici) di un oggetto di classe `C`. L'assegnamento `ptC=&oBD` è quindi illegale e causa un errore in compilazione (le due istruzioni vanno quindi rimosse - la sola `ptc->f();` darebbe luogo ad un errore in compilazione in quanto `ptc` è nullo).

(`ptD=&oBE; ptD->g()`) La classe `E` eredita da `D` in modalità pubblica, quindi è lecito assegnare a un puntatore ad un oggetto di classe `D` l'indirizzo di un oggetto di classe `E` (che può comportarsi come un oggetto di classe `B`). L'istruzione `ptD->g()` è però illegale in quanto `g` è protetta e perciò accessibile solo all'interno di funzioni membro della classe e/o in classi e funzioni amiche. Tale istruzione compare invece nella funzione `main`.

(`oBE.g();`) Viene eseguita la funzione `g` della classe `E`, la quale chiama la funzione `g` della classe `C` che chiama la funzione virtuale `f`, con la stampa del messaggio `f di D`, in quanto la versione di `f` più vicina alla classe `E` è quella presente nella classe `D`.