Ayudantía 3

Carlos Lagos - carlos.lagosc@usm.cl

Tipos de Datos Abstractos

Tipos de Datos Abstractos

Debes crear una clase que modele un proyecto. Este proyecto incluye una lista de tareas, cada una con un nombre, estado y cantidad de horas requeridas. La implementación debe utilizar memoria dinámica para representar la lista de tareas, y asegurarse de liberar esa memoria correctamente en el destructor de la clase. Además, la clase debe permitir agregar nuevas tareas, listar las existentes y marcar una tarea como terminada.

Tipos de Datos Abstractos

```
struct Tarea {
    string nombre;
    bool estado; // Verdadero si está activa
    int horas;
};
```

El struct Tarea representa una tarea dentro de un proyecto. Cada tarea tiene un nombre que describe la actividad a realizar, un estado que indica si la tarea está activa o no, siendo verdadero si está activa, y un número entero que representa la cantidad de horas estimadas para completar la tarea.

Tipos de Datos Abstractos

```
class Proyecto {
  private:
    int cantidad_de_tareas;
    int cantidad_maxima_de_tareas;
    Tarea* tareas;

  public:
    // Constructor de la clase, recibe la cantidad máxima de tareas
    Proyecto(int cantidad_maxima);
    // Destructor de la clase
    ~Proyecto();
    // Crea una nueva tarea y retorna la posición en la que se asigna
    int agregarTarea(string nombre , int horas);
    // Imprime todas las tareas
    void listarTareas();
    // Marca una tarea en cierta posición como terminada, en caso que exista
    void marcarTareaComoTerminada(int posicion);
};
```

Tipos de Datos Abstractos

El constructor recibe la cantidad máxima de tareas y define una lista de Tareas de largo 'cantidad_maxima' usando memoria dinámica. Además, guarda la cantidad máxima y asigna 0 a la cantidad de tareas actuales.

```
Proyecto::Proyecto(int cantidad_maxima){
  tareas = new Tarea[cantidad_maxima];
  cantidad_maxima_de_tareas = cantidad_maxima;
  cantidad_de_tareas = 0;
}
```

El destructor elimina la memoria dinámica asignada a la lista de tareas. Entonces se llamará automáticamente al borrarse la clase en el stack.

```
Proyecto::~Proyecto(){
    delete[] tareas;
}
```

Tipos de Datos Abstractos

La siguiente función, agregarTarea, añade una nueva tarea a la lista del proyecto. Si el límite de tareas ha sido alcanzado, devuelve -1. En caso contrario, asigna el nombre y las horas proporcionadas a la nueva tarea, estableciendo su estado como activo. Finalmente, devuelve la posición de la nueva tarea en la lista.

```
int Proyecto::agregarTarea(string nombre, int horas){
    if(cantidad_maxima_de_tareas == cantidad_de_tareas){
        return -1;
    }
    tareas[cantidad_de_tareas].nombre = nombre;
    tareas[cantidad_de_tareas].horas = horas;
    tareas[cantidad_de_tareas].estado = true;
    int posicion = cantidad_de_tareas++;
    return posicion;
}
```

Este método recorre la lista de tareas y muestra en pantalla el nombre, estado (activo o inactivo) y la cantidad de horas estimadas para cada tarea del proyecto.

```
void Proyecto::listarTareas(){
    for(int i = 0; i < cantidad_de_tareas; i++){
        cout << tareas[i].nombre << " " << tareas[i].estado << " " << tareas[i].horas << endl;
    }
}
```

Este método permite marcar una tarea específica como terminada, cambiando su estado de activo a inactivo. Se especifica la posición de la tarea que se desea marcar como terminada.

```
void Proyecto::marcarTareaComoTerminada(int posicion){
    if(posicion < 0 || posicion >= cantidad_de_tareas){
        return;
    }
    tareas[posicion].estado = false;
}
```

Análisis de algoritmos

Análisis de algoritmos

La notación O grande se define como:

$$O(g(n)) = \{f(n) : \text{existen constantes } c > 0 \text{ y } n_0 \text{ tales que}$$
$$0 \leq f(n) \leq c \cdot g(n) \text{ para todo } n \geq n_0\}$$

Esto significa que una función $f(n)$ pertenece a $O(g(n))$ si existe una constante positiva c y un valor de n_0 a partir del cual $f(n)$ siempre es menor o igual a $c \cdot g(n)$. Es decir, $f(n)$ no crece más rápido que una constante múltiplo de $g(n)$.

Análisis de algoritmos

Para demostrar que $f(n)$ pertenece a $O(g(n))$, a menudo se utiliza el límite:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = k$$

donde k es una constante real, que puede ser 0 o un número real finito pero no infinito. Si el límite existe con esas condiciones, entonces $f(n)$ pertenece a $O(g(n))$.

Análisis de algoritmos: Verdadero o Falso

Análisis de algoritmos: Verdadero o Falso

Demostrar la veracidad de las siguientes afirmaciones mediante la definición de pertenencia a la notación O grande.

- $f(n) \in O(g(n))$:
  - $\lim_{n \to \infty} \frac{f(n)}{g(n)} = k, \quad k \in \mathbb{R}^+ \cup \{0\}, \quad k \neq \infty$

Ejercicios:

1. $n^2 + 3n + 10 \in O(n^2)$
2. $3n \in O(n \log(n))$
3. $n^2 \in O(\sqrt{n})$

Análisis de algoritmos: Verdadero o Falso

Verdadero o Falso: $n^2 + 3n + 10 \in O(n^2)$

$$\lim_{n \to \infty} \frac{n^2 + 3n + 10}{n^2} = \lim_{n \to \infty} \left(1 + \frac{3}{n} + \frac{10}{n^2}\right) = 1$$

Es verdadero porque su límite tiende a 1 y no a infinito, por lo tanto, pertenece a $O(n^2)$.

Análisis de algoritmos: Verdadero o Falso

Verdadero o Falso: $3n \in O(n \cdot \log(n))$

$$\lim_{n \to \infty} \frac{3n}{n \log(n)} = \lim_{n \to \infty} \frac{3}{\log(n)} = 0$$

Es verdadero ya que su límite tiende a 0 y no es infinito, por lo tanto, pertenece a $O(n \log(n))$.

Análisis de algoritmos: Verdadero o Falso

Verdadero o Falso: $n^2 \in O(\sqrt{n})$

$$\lim_{n\to\infty} \frac{n^2}{\sqrt{n}} = \lim_{n\to\infty} \frac{n^2}{n^{\frac{1}{2}}} =$$

$$\lim_{n\to\infty} n^{\left(2-\frac{1}{2}\right)} = \lim_{n\to\infty} n^{\frac{3}{2}} = \infty$$

Es falso, ya que para que pertenezca a $O(\sqrt{n})$, el límite debe tender a un valor diferente de ∞.

Análisis de algoritmos: Codigos

Análisis de algoritmos: Codigos

```
bool es_primo(int  n){
  int i = 2;
  bool primo = false;
  while(i < n){
    if(n % i == 0){
      primo = true;
      break;
    }
    i++;
  }
  return primo;
}
```

Responder:

1. ¿Qué hace el algoritmo anterior?
2. ¿Cuál es la cantidad de iteraciones en el mejor caso y en el peor caso?
3. ¿A qué notación de O grande pertenece?

Análisis de algoritmos: Codigos

```
void ordenar(int *arreglo, int largo){
  for(int i = 0; i < largo; i++){
    for(int j = 0; i < largo - 1; j++){
      if(arreglo[j] > arreglo[j + 1]){
        int auxiliar = arreglo[j];
        arreglo[j] = arreglo[j + 1];
        arreglo[j + 1] = auxiliar;
      }
    }
  }
}
```

Responder:

1. ¿Qué hace el algoritmo anterior?
2. ¿Cuál es la cantidad de iteraciones en el mejor caso y en el peor caso?
3. ¿A qué notación de O grande pertenece?