**Documentation: Linked Lists, initializations and operations**

This program reads child profile data from the "INPUT.TXT" file, builds a linked list of child profiles, sorts the list, determines gifts for each child based on specific criteria, and writes the child profiles and their assigned gifts to the "OUTPUT.TXT" file.

**Basic Functions of the program**

1. **lastname\_mp3\_classes.py**
   * This file contains the definitions of classes and functions used in the main program.
   * It starts with the import statement **import os** to import the **os** module, which provides functions for interacting with the operating system.

import os

* + The **ChildProfile** class is defined, representing the profile of a child, with attributes for first name, last name, gender, age, good deeds, bad deeds, and gift.

#DO NOT EDIT

class ChildProfile:

    first\_name: str

    last\_name: str

    gender: str

    age: int

    good\_deeds: int

    bad\_deeds: int

    gift: str

#DO NOT EDIT

* + The **Node** class is defined, representing a node in a linked list. It has attributes for data and a reference to the next node.

class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

* + The **ChildProfileList** class is defined, representing a linked list of child profiles.
  + The **children** list and **head** attribute are initialized within the **ChildProfileList** class.

class ChildProfileList:

    children = []

    head = None

1. **build\_list\_from\_file(self)**
   * This method of the **ChildProfileList** class reads the data from the "INPUT.TXT" file and builds the linked list of child profiles.
   * It first gets the directory path of the "lastname\_main.py" file using **os.path.dirname(os.path.realpath(\_\_file\_\_))**.

# Get the directory of the lastname\_main.py file

        dir\_path = os.path.dirname(os.path.realpath(\_\_file\_\_))

* + Then, it constructs the path to the "INPUT.TXT" file using **os.path.join(dir\_path, "INPUT.TXT")**.

input\_file\_path = os.path.join(dir\_path, "INPUT.TXT")

* + The file is opened in read mode using a **with** statement.

with open(input\_file\_path, "r") as file:

* + Each line of the file is processed in a loop.

lines = file.readlines()

for line in lines:

* + The line is split into individual data elements using the **split()** method, and the values are assigned to variables.

first\_name, last\_name, gender, age, good\_deeds, bad\_deeds = line.strip().split()

* + A new **ChildProfile** object is created and initialized with the extracted data.

child = ChildProfile()

                child.first\_name = first\_name

                child.last\_name = last\_name

                child.gender = gender

* + The age, good deeds, and bad deeds are converted to integers and assigned to the corresponding attributes of the child object.

try:

                    child.age = int(age)

                    child.good\_deeds = int(good\_deeds)

                    child.bad\_deeds = int(bad\_deeds)

                except ValueError:

                    print("ValueError: Invalid data in INPUT.TXT file")

                    exit(1)

* + A new **Node** object is created with the **ChildProfile** object as its data.
  + If the linked list is empty (i.e., **self.head** is **None**), the new node becomes the head.
  + Otherwise, the new node is appended to the end of the linked list.

new\_node = Node(child)

                if self.head is None:

                    self.head = new\_node

                else:

                    current = self.head

                    while current.next is not None:

                        current = current.next

                    current.next = new\_node

1. **swap(self, node\_1, node\_2)**
   * This method of the **ChildProfileList** class swaps the data of two nodes.
   * It takes two node objects, **node\_1** and **node\_2**, as arguments.
   * The data of **node\_1** is temporarily stored in the **temp** variable.
   * The data of **node\_2** is assigned to **node\_1.data**.
   * The data stored in **temp** is assigned to **node\_2.data**, effectively swapping the data of the two nodes.

def swap(self, node\_1, node\_2):

        temp = node\_1.data

        node\_1.data = node\_2.data

        node\_2.data = temp

1. **sort\_list(self)**
   * This method of the **ChildProfileList** class sorts the linked list of child profiles in ascending order based on the last name and, if necessary, the first name.
   * It implements the bubble sort algorithm.
   * If the linked list is empty (i.e., **self.head** is **None**), the method returns without performing any sorting.
   * The **swapped** variable is initially set to **True**.
   * A while loop is used to iterate until no more swaps are performed (i.e., **swapped** becomes **False**).

def sort\_list(self):

        if self.head is None:

            return

        swapped = True

* + Inside the loop, a nested while loop is used to iterate through the linked list.
  + If the last name of the current node is greater than the last name of the next node, or if the last names are equal but the first name of the current node is greater than the first name of the next node, the **swap()** method is called to swap the data of the two nodes.
  + After a swap, the **swapped** variable is set to **True** to indicate that a swap has occurred.
  + The outer while loop continues until no more swaps are performed.

while swapped:

            swapped = False

            current = self.head

            while current.next is not None:

                if current.data.last\_name > current.next.data.last\_name or (current.data.last\_name == current.next.data.last\_name and current.data.first\_name > current.next.data.first\_name):

                    self.swap(current, current.next)

                    swapped = True

                current = current.next

1. **determine\_gift(self)**
   * This method of the **ChildProfileList** class assigns a gift to each child profile based on their age, gender, and good/bad deeds.
   * It iterates through the linked list using a **current** variable, which starts from the head node.

def determine\_gift(self):

        current = self.head

        while current is not None:

* + Inside the loop, a series of if-else statements are used to determine the appropriate gift for each child based on their age, gender, and good/bad deeds.
  + The determined gift is assigned to the **gift** attribute of the current child profile.
  + The **current** variable is updated to the next node in the linked list.
  + This process continues until all child profiles have been processed.

if current.data.age <= 5:

                if current.data.gender == 'M':

                    if current.data.age == 1:

                        current.data.gift = 'Blue\_Pacifier'

                    elif current.data.age == 2:

                        current.data.gift = 'Colored\_Shapes'

                    elif current.data.age == 3:

                        current.data.gift = 'Choo\_Choo\_Train'

                    elif current.data.age == 4:

                        current.data.gift = 'Wooden\_Horse'

                    else:

                        current.data.gift = 'Remote\_Controlled\_Car'

                else:

                    if current.data.age == 1:

                        current.data.gift = 'Pink\_Pacifier'

                    elif current.data.age == 2:

                        current.data.gift = 'Colored\_Shapes'

                    elif current.data.age == 3:

                        current.data.gift = 'Teddy\_Bear'

                    elif current.data.age == 4:

                        current.data.gift = 'Doll'

                    else:

                        current.data.gift = 'Pair\_of\_Shoes'

            elif 6 <= current.data.age <= 12:

                deeds = current.data.good\_deeds - current.data.bad\_deeds

                if deeds <= 0:

                    current.data.gift = 'Good\_Manners\_and\_Right\_Conduct\_Book'

                else:

                    if 6 <= current.data.age <= 8:

                        if current.data.gender == 'M':

                            current.data.gift = 'Chess\_Set'

                        else:

                            current.data.gift = 'Disney\_Puzzle'

                    else:

                        if current.data.gender == 'M':

                            current.data.gift = 'Soccer\_Ball'

                        else:

                            current.data.gift = 'Blouse\_Pants'

            else:

                current.data.gift = 'E\_Christmas\_Card'

            current = current.next

1. **write\_gift\_file(self)**
   * This method of the **ChildProfileList** class writes the child profiles and their assigned gifts to the "OUTPUT.TXT" file.
   * It follows a similar process as the **build\_list\_from\_file()** method to construct the output file path and open the file in write mode.

def write\_gift\_file(self):

        # Get the directory of the lastname\_main.py file

        dir\_path = os.path.dirname(os.path.realpath(\_\_file\_\_))

        # Construct the path to the OUTPUT.TXT file

        output\_file\_path = os.path.join(dir\_path, "OUTPUT.TXT")

        with open(output\_file\_path, "w") as file:

* + A **current** variable is used to iterate through the linked list.
  + Inside the loop, each child profile's last name, first name, gender, and gift are written to the file using **file.write()**.
  + A newline character is added after each entry to separate them.

current = self.head

            while current is not None:

                file.write(f"{current.data.last\_name} {current.data.first\_name} {current.data.gender} {current.data.gift}\n")

                current = current.next

* + Finally, a message is printed to the console indicating that the output file has been written.

print("Output file written, check OUTPUT.TXT")

1. **lastname\_main.py**

* This is the main program file that executes the program.
* It starts by importing the **ChildProfileList** class from **lastname\_mp3\_classes.py**.
* The main program logic is executed within the **if \_\_name\_\_ == '\_\_main\_\_':** block, which ensures that the code inside is only executed when the file is run directly and not when imported as a module.
* An instance of the **ChildProfileList** class is created and assigned to the **child\_list** variable.
* The **build\_list\_from\_file()** method is called to build the linked list of child profiles from the "INPUT.TXT" file.
* The **sort\_list()** method is called to sort the linked list.
* The **determine\_gift()** method is called to assign gifts to each child profile.
* The **write\_gift\_file()** method is called to write the child profiles and their assigned gifts to the "OUTPUT.TXT" file.