```
// VirtualPet.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include "chrono"
#include "windows.h"
#include "conio.h"
#include "PetFunctions.h"

int main()
{
#pragma region Pet setup

    HANDLE Cons = GetStdHandle(STD_OUTPUT_HANDLE);// honestly have no idea what this actually does but from my understanding if I have read the documentation correctly it's a pointer for the output but that's only a guess at this point

    TAMA CurrentPet{ "" }; // creates an instance of the pet. The brackets at the end get rid of an annoying warning of the "variables not initialzed" which for the most part doesn't matter as they get initialized in the next line
    Init(&CurrentPet);//initialises the pet variables
    NamingPet(&CurrentPet);// allows the pet to be named
#pragma endregion
#pragma region Prototype functions
    // declares a prototype of all the functions in this file here

    int Hung_Sleep_Hydrate();
    void HappinessCalc(TAMA*);
    void Print(TAMA*, int, HANDLE);
    void sleep(TAMA*, int, const clock_t);
    void Update(TAMA*, char, int, clock_t, clock_t, double, HANDLE);
#pragma endregion

#pragma region variables
    // this set up the variables and allows them to be used in the functions properly
    double time = 1;// this is used to check in the while statement if one second has passed
    int N = sizeof(CurrentPet.Levels) / sizeof(CurrentPet.Levels[0]); // this gets the size of the list for anything that requires it and will only ever get called
    clock_t Timer = clock();
    clock_t BackgroundTimer = clock();
    char key = '1';
#pragma endregion

#pragma region Update
    // this is to make it easier to find the Update function
    Update(&CurrentPet, key, N, Timer, BackgroundTimer, time, Cons);
#pragma endregion

#pragma region Dead
    // this is to find the bit that prints the string for when the pet is dead quicker
```

```
45
46      system("CLS");
47      Print(&CurrentPet, N, Cons);
48  #pragma endregion
49  }
50
51  int Hung_Sleep_Hydrate() // this will return a value to whatever one needs it. If the Bool is true it will be a positive number otherwise it will be negative. this should technically be in the petFunctions but since it doesn't use the TAMA struct it's fine here
52  {
53      int value = 0;
54
55      //positive number
56      value = rand() % 15 + 10;
57      return value;
58
59  }
60  void HappinessCalc(TAMA* Pet)
61  {
62      Pet->Levels[2] = (Pet->Levels[1] + Pet->Levels[0] + Pet->Levels[3]) / 3; // this gives the average happiness
63      Pet->Levels[2] += Pet->HAP;
64  }
65
66  void Print(TAMA* CurrentPet, int N, HANDLE Cons) // this just contains the majority of the functions that will print something in it and the stuff that will print when the pet
67  {
68      if (CurrentPet->isDead == false)
69      {
70          system("CLS");
71          HappinessCalc(CurrentPet);
72          StatCap(CurrentPet, N);
73          DisplayStats(CurrentPet, Cons);
74      }
75      else
76      {
77          SetConsoleTextAttribute(Cons, 4);//red
78
79          cout << "\n\n The Pet is dead.\n\n It survived " << (clock() - CurrentPet->Start) / (double)CLOCKS_PER_SEC << "Seconds";
80
81      }
82  }
83
84  void Sleep(TAMA* CurrentPet, int N, const clock_t& Timer, HANDLE Cons) // this will constantly print whilst the pet is asleep and
85  {
86      Print(CurrentPet, N, Cons);
87      cout << "The pet will sleep for " << CurrentPet->duration << " second/s";
88      if (CurrentPet->isAsleep)
89          cout << "\n\n Time Passed: " << ((double)clock() - Timer) / (double) CLOCKS_PER_SEC; // this will only print if the bool isAsleep is false
90      Decrease(CurrentPet, N);
```

```
 91      CurrentPet->Levels[3]++;
 92  }
 93
 94  void Update(TAMA* CurrentPet, char key, int N, clock_t Timer, clock_t
       BackgroundTimer, double time, HANDLE Cons)
 95  {
 96
 97
 98      Print(CurrentPet, N, Cons);
 99      while (!CurrentPet->isDead)
100      {
101          if (CurrentPet->Levels[0] <= 0 || CurrentPet->Levels[1] <= 0 ||
              CurrentPet->Levels[2] <= 0) // if the hunger,hydration or happiness
              hit 0 or less then the pet dies
102          {
103              CurrentPet->isDead = true;
104              break;
105          }
106          while (time > (double)((double)clock() - BackgroundTimer) / (double)
              CLOCKS_PER_SEC)// checks if a second has passed and if not then it
              will do what's inside this statement
107          {
108
109
110              if (_kbhit() && CurrentPet->isAsleep == false && CurrentPet-
                  >PassedOut == false)// this checks if the keyboard has been hit
                  and if it hasn't it will set it to a a default value of 1 which
                  will never be used and allows the background timer to work
                  properly. This also checks that the pet is asleep so you can't
                  feed or water it when it's asleep
111              {
112                  key = _getch();
113              }
114              else
115              {
116                  key = '1';
117              }
118              if (CurrentPet->Levels[3] == 0)
119              {
120                  CurrentPet->PassedOut = true;
121              }
122              if (CurrentPet->PassedOut)// this will force the pet to collapse
                  and fall asleep
123              {
124                  while (CurrentPet->Levels[3] < 50)
125                  {
126                      Sleep(CurrentPet, N, Timer, Cons);
127
128                  }
129                  CurrentPet->PassedOut = false;
130              }
131              if (key == 'f' || key == 'F')// the hunger stat is satisfied by a
                  random amount
132              {
133                  CurrentPet->Levels[0] += Hung_Sleep_Hydrate();
134                  Print(CurrentPet, N, Cons);
```

```
135
136
137             }
138             if (key == 'h' || key == 'H') // the hydration stat is quenched by a random amount
139             {
140                 CurrentPet->Levels[1] += Hung_Sleep_Hydrate();
141                 Print(CurrentPet, N, Cons);
142
143             }
144             if (key == 'p' || key == 'P') // this was
145             {
146                 CurrentPet->HAP++; // this increases the
147                 Print(CurrentPet, N, Cons);
148
149             }
150             if (key == 's' || key == 'S') // puts it to sleep
151             {
152                 CurrentPet->duration = rand() % 3 + 0b1;// randomly decides how long the pet will sleep
153                 Timer = clock();// sets Timer to the current time
154                 while (CurrentPet->duration > (double)((double)clock() - Timer) / (double)CLOCKS_PER_SEC)// this essentially is checking that the duration has passed and if it hasn't it will loop the inside of this till it has
155                 {
156                     CurrentPet->isAsleep = true; // used for painting the pet asleep in the console
157                     Sleep(CurrentPet, N, Timer, Cons);
158                 }
159                 CurrentPet->isAsleep = false;//used for painting the pet awake in the console
160                 Print(CurrentPet, N, Cons);
161             }
162             if (key == 'z')
163             {
164                 CurrentPet->isAsleep = true;
165             }
166             if (key == 'x' || key == 'X')
167             {
168                 CurrentPet->isDead = true;
169             }
170         }
171         //once a second has passed it will do this stuff bellow
172         for (int i = 0; i < N; i++)
173         {
174             CurrentPet->Levels[i] -= rand() % 2 + 1; // this will decrease all the values by a random value when the background timer hits
175         }
176         CurrentPet->HAP -= rand() % 2; // this will randomly decrease the petting happiness
177
178         Print(CurrentPet, N, Cons);
179
180         BackgroundTimer = clock(); // this resets the background timer and restarts the count
```

```
181
182
183
184         }
185     }
186
```