Kocia muzyka – projekt zaliczeniowy (*Programowanie*)

maj/czerwiec 2025

Przygotowanie (29 maja – 5 czerwca)

Wstęp

Pewien Pomarańczowy Kot zapragnął czegoś więcej w swoim kocim życiu i poza dziurkowaniem kartek przednimi ząbkami postanowił... nauczyć się grać na skrzypcach. W internecie niektórzy twierdzą, że pomarańczowe koty mają tylko jeden neuron w głowie. Czy mają oni rację, to się jeszcze okaże, aczkolwiek nasz futrzany bohater podjął kontrowersyjną decyzję i postanowił nauczyć się gry na instrumencie samodzielnie, bez nauczyciela. W pewnym momencie Kot zorientował się jednak, że gdy chce zagrać daną nutę, a skrzypce to przecież nie fortepian ani gitara (która ma progi), to trzeba doskonale wiedzieć, gdzie przycisnąć strunę futrzaną łapką, aby otrzymać odpowiedni dźwięk podczas pociągnięcia smyczkiem drugą łapką (bądź ogonkiem). Wprawiło to futrzaka w zakłopotanie i już chciał on zrezygnować ze swej szczytnej idei aktywnego spędzania czasu w napiętym grafiku dnia między jedzeniem a spaniem, gdy przypomniał sobie, że na pomoc może mu przyjść odpowiedni program komputerowy napisany w języku C++.

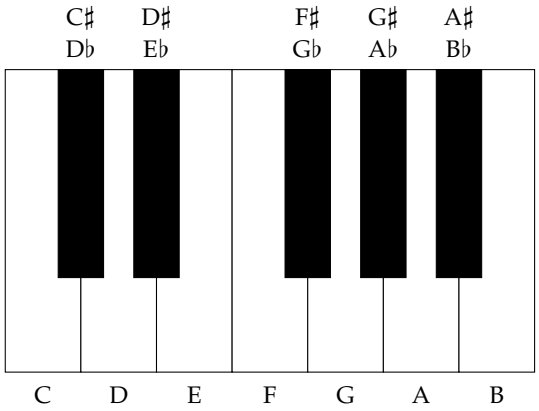
Idea Kota była bardzo prosta: częstotliwość dźwięku rejestrowanego mikrofonem można odczytać z użyciem odpowiednich komend linuksowego wiersza poleceń. Wystarczy już tylko napisać program, który wczytując ze standardowego wejścia częstotliwość w hercach wypisuje na standardowe wyjście odpowiednie informacje o nucie. I właśnie Twoim zadaniem będzie napisanie takiego programu, według kocich wskazówek podanych dalej.

Podstawy teorii muzyki

Pomarańczowy Kot przestudiował nieco notacji muzycznej i choć nie zostanie on raczej wykładowcą żadnej muzycznej uczelni, to może podzielić się swoim skromnym dorobkiem. Na potrzeby tego zadania nie będziemy potrzebować zbyt wielu pojęć.

Nuta (ang. *pitch*) jest właściwością tonu muzycznego określającego jak wysoki bądź niski jest ten ton. Sam ton z kolei jest ciągłym periodycznym dźwiękiem składającego się z jednej (ton czysty) lub wielu sinusoid (ton złożony). Częstotliwość f_p nuty p jest to częstotliwość tonu czystego o tej samej wysokości co ton nuty p. Istnieją dwa powszechnie stosowane systemy notacji nut: MIDI (ang. *Musical Instrument Digital Interface*) oraz nutowa notacja naukowa. Dla uproszczenia zakładamy system równomiernie temperowany opisany dalej i strój koncertowy, gdzie nuta A4 ma częstotliwość 440 Hz. Nie jest to nietypowe ograniczenie — praktycznie cała współczesna zachodnia muzyka jest w ten sposób opisywana.

W notacji naukowej nutę zapisuje się jako klasę dźwięku połączoną z numerem oktawy. Zacznijmy od klasy dźwięku. Z pewnością kojarzysz nazwy solmizacyjne („do", „re", „mi", „fa", „sol", „la", „si"). Nazwy solmizacyjne mają swoje „profesjonalne" odpowiedniki (C, D, E, F, G, A, B). Na fortepianie odpowiadają im białe klawisze — możesz zobaczyć to na rysunku 1. Są to tzw. dźwięki diatoniczne. Jeśli dołożyć tzw. znaki chromatyczne (♯, ♭) to otrzymamy pozostałe nuty — czarne klawisze fortepianu. I niespodzianka: oktawa składa się z 12 nut



Rysunek 1: Oktawa. Widocznych jest 12 klas dźwięków: C, C♯ (lub inaczej D♭), D, D♯, E, F, F♯, G, G♯, A, A♯ oraz B.

(zwanych półtonami). W obranym przez nas systemie niektóre nuty mają podwójną nazwę, np. C♯ jest inaczej nazywane jako D♭. Wynika to z faktu, że ten dźwięk można otrzymać zarówno z dźwięku C jak i D poprzez odpowiednio podwyższenie lub obniżenie częstotliwości o pół tonu.

W ten sposób dochodzimy do ważnego miejsca. Otóż częstotliwość kolejnego półtonu jest o czynnik $2^{1/12}$ większa od poprzedniego. Widać to w tabeli 1. Gdy podzielimy częstotliwość dwóch następujących po sobie nut, np. A♯4 przez A4 to otrzymamy właśnie liczbę $2^{1/12}$. Oznacza to również, że nuta o oktawę wyższa, np. A5 w porównaniu z A4, ma dwa razy wyższą częstotliwość, ponieważ $(2^{1/12})^{12} = 2$. Dlatego też do kompletnego opisu danej nuty potrzebny jest, poza klasą dźwięku, numer oktawy. Numery oktaw są liczbami całkowitymi, niekoniecznie naturalnymi. System MIDI przypisuje kolejnym nutom liczby naturalne w sposób pokazany w tabeli 1. Odległość między dwiema częstotliwościami f_a i f_b to interwał dany wzorem $12 \cdot \log_2(f_a/f_b)$ i mierzony w półtonach. Aby przeliczyć interwał na jednostkę centy należy pamiętać, że interwał jednego półtonu to 100 centów.

Rozporządzenie Pomarańczowego Kota nr 1

> Ja, Pomarańczowy Kot, śpiący prawie cały dzień i jedzący przez niemal resztę dnia, niniejszym tym oto dekretem przykazuję, aby każdy śmiałek, który podejmie się dalej opisanego zadania, wykonał je w prześwietnym systemie operacyjnym GNU/Linux z wykorzystaniem kompilatora GNU C++ oraz programu Emacs lub Vim, lub z użyciem prostego edytora, który nie zawiera narzędzi sztucznej inteligencji, czyli np. z użyciem programu gedit lub nano.
>
> *Kot*

Ciąg dalszy nastąpi...

Tabela 1: Nuty i odpowiadające im częstotliwości. Ciekawostki: A0 (MIDI: 21) – najniższa nuta standardowego fortepianu, G3 (55) – najniższa nuta standardowych skrzypiec (pierwsza otwarta struna skrzypiec), D4 (62) – druga otwarta struna skrzypiec, A4 (69) – nuta referencyjna (trzecia otwarta struna skrzypiec), E5 (76) – czwarta otwarta struna skrzypiec, C7 (96) / D7 (98) – ostatnia używana na zwykłych skrzypcach nuta, C8 (108) – najwyższa nuta standardowego fortepianu.

MIDI	nuta	f [Hz]	MIDI	nuta	f [Hz]	MIDI	nuta	f [Hz]	MIDI	nuta	f [Hz]
0	C−1	8,18	32	G♯1	51,91	64	E4	329,63	96	C7	2093,00
1	C♯−1	8,66	33	A1	55,00	65	F4	349,23	97	C♯7	2217,46
2	D−1	9,18	34	A♯1	58,27	66	F♯4	369,99	98	D7	2349,32
3	D♯−1	9,72	35	B1	61,74	67	G4	392,00	99	D♯7	2489,02
4	E−1	10,30	36	C2	65,41	68	G♯4	415,30	100	E7	2637,02
5	F−1	10,91	37	C♯2	69,30	69	A4	440,00	101	F7	2793,83
6	F♯−1	11,56	38	D2	73,42	70	A♯4	466,16	102	F♯7	2959,96
7	G−1	12,25	39	D♯2	77,78	71	B4	493,88	103	G7	3135,96
8	G♯−1	12,98	40	E2	82,41	72	C5	523,25	104	G♯7	3322,44
9	A−1	13,75	41	F2	87,31	73	C♯5	554,37	105	A7	3520,00
10	A♯−1	14,57	42	F♯2	92,50	74	D5	587,33	106	A♯7	3729,31
11	B−1	15,43	43	G2	98,00	75	D♯5	622,25	107	B7	3951,07
12	C0	16,35	44	G♯2	103,83	76	E5	659,26	108	C8	4186,01
13	C♯0	17,32	45	A2	110,00	77	F5	698,46	109	C♯8	4434,92
14	D0	18,35	46	A♯2	116,54	78	F♯5	739,99	110	D8	4698,64
15	D♯0	19,45	47	B2	123,47	79	G5	783,99	111	D♯8	4978,03
16	E0	20,60	48	C3	130,81	80	G♯5	830,61	112	E8	5274,04
17	F0	21,83	49	C♯3	138,59	81	A5	880,00	113	F8	5587,65
18	F♯0	23,12	50	D3	146,83	82	A♯5	932,33	114	F♯8	5919,91
19	G0	24,50	51	D♯3	155,56	83	B5	987,77	115	G8	6271,93
20	G♯0	25,96	52	E3	164,81	84	C6	1046,50	116	G♯8	6644,88
21	A0	27,50	53	F3	174,61	85	C♯6	1108,73	117	A8	7040,00
22	A♯0	29,14	54	F♯3	185,00	86	D6	1174,66	118	A♯8	7458,62
23	B0	30,87	55	G3	196,00	87	D♯6	1244,51	119	B8	7902,13
24	C1	32,70	56	G♯3	207,65	88	E6	1318,51	120	C9	8372,02
25	C♯1	34,65	57	A3	220,00	89	F6	1396,91	121	C♯9	8869,84
26	D1	36,71	58	A♯3	233,08	90	F♯6	1479,98	122	D9	9397,27
27	D♯1	38,89	59	B3	246,94	91	G6	1567,98	123	D♯9	9956,06
28	E1	41,20	60	C4	261,63	92	G♯6	1661,22	124	E9	10548,08
29	F1	43,65	61	C♯4	277,18	93	A6	1760,00	125	F9	11175,30
30	F♯1	46,25	62	D4	293,66	94	A♯6	1864,66	126	F♯9	11839,82
31	G1	49,00	63	D♯4	311,13	95	B6	1975,53	127	G9	12543,85

Zadanie #1 (5-12 czerwca)

Twoim zadaniem programistycznym na dzisiejsze zajęcia będzie napisanie dwóch funkcji swobodnych oraz jednej klasy. Pamiętaj, aby odpowiednią funkcjonalność zawrzeć w plikach nagłówkowych (o rozszerzeniu *.h*) oraz implementacji (o rozszerzeniu *.cpp*). Dbaj o czytelność kodu (odpowiednie formatowanie, w tym wcięcia). Pamiętaj też o Rozporządzeniu nr 1 ogłoszonym w poprzednim tygodniu oraz nr 2 zawartym niżej.

Rozporządzenie Pomarańczowego Kota nr 2

> Ja, Pomarańczowy Kot, dziurkujący w wolnym czasie ważne kartki leżące na biurku mego właściciela, niniejszym tym oto dekretem przykazuję, aby każde rozwiązanie, przed zakończeniem dzisiejszej pracy na zajęciach, zostało uprzednio spakowane jako archiwum o rozszerzeniu *.zip*, *.tar*, *.tar.gz* lub *.tar.xz* a następnie wysłane przez platformę Kampus do oceny. Jeśliby zaś zaszła potrzeba uzupełnienia czegoś w domu, to przykazuję ponadto, aby najpóźniej na początku następnych zajęć wysłać zaktualizowany stan pracy w formie takiegoż samego archiwum na platformie Kampus.
>
> *Kot*

Funkcja *semitones*

Funkcja przyjmuje dwa argumenty zmiennoprzecinkowe i zwraca liczbę półtonów (bez zaokrąglania) między dwiema częstotliwościami f_a oraz f_b równą $12 \cdot \log_2(f_a/f_b)$, gdzie wartości f_a oraz f_b dane są jako argumenty funkcji *semitones*.

Funkcja *semitones_from_a4*

Funkcja przyjmuje jeden argument zmiennoprzecinkowy i zwraca liczbę półtonów (bez zaokrąglania) między częstotliwością f a częstotliwością referencyjną A4 równą 440 Hz, gdzie częstotliwość f dana jest jako argument funkcji *semitones_from_a4*.

Klasa *pitch*

Klasa opisuje nutę muzyczną i posiada następujące funkcje:

- Konstruktor *pitch*, który przyjmuje jeden argument całkowitoliczbowy z wartością MIDI nuty.
- Metody *get_midi* oraz *set_midi*, które odpowiednio pobierają i ustawiają nutę zapisaną w systemie MIDI.
- Metoda bezargumentowa *frequency*, która zwraca częstotliwość nuty zgodnie ze wzorem $2^{(m-69)/12} \cdot 440\,\text{Hz}$, gdzie m jest nutą zapisaną w systemie MIDI.
- Metoda *interval_cents*, która przyjmuje jeden argument zmiennoprzecinkowy i zwraca interwał w centach między częstotliwością daną jako argument *interval_cents* a częstotliwością nuty opisywanej przez obiekt, na rzecz którego wywoływana jest metoda.
- [Opcjonalnie] Metody *operator+* i *operator−*, które zwracają obiekt klasy *pitch* odpowiednio powiększony i pomniejszony w systemie MIDI o wartość podaną jako drugi argument operacji arytmetycznej.

Pamiętaj, aby dane składowe klasy były prywatne.

Ciąg dalszy nastąpi...

Zadanie #2 (12-19 czerwca)

Tym razem do napisania będą dwie kolejne klasy, jedna funkcja swobodna i program (funkcja *main*) wykorzystujący całość dotychczasowej pracy. Pamiętaj, aby odpowiednią funkcjonalność zawrzeć w plikach nagłówkowych (o rozszerzeniu *.h*) oraz implementacji (o rozszerzeniu *.cpp*). Dbaj o czytelność kodu (odpowiednie formatowanie, w tym wcięcia). Zapoznaj się też z Rozporządzeniem nr 3 umieszczonym dalej.

Klasa *pitch_reading*

Klasa opisuje zagraną na instrumencie nutę muzyczną i posiada następujące funkcje:

- Konstruktor *pitch_reading*, który przyjmuje jeden argument zmiennoprzecinkowy z częstotliwością zagranej nuty. Uwaga: Nuta może zostać zagrana fałszywie, dlatego nie należy zakładać, że podana częstotliwość odpowiada *dokładnie* jakiejś konkretnej nucie.
- Metoda *closest_pitch*, która zwraca obiekt klasy *pitch* reprezentującą nutę najbliższą danej częstotliwości.
- Metoda *deviation_cents*, która zwraca interwał między zagraną częstotliwością a najbliższą nutą w centach.
- Metoda *fidelity*, która zwraca napis typu *std::string* z opisem jakości zagranej nuty zgodnie z tabelą 2.

Tabela 2: Wyjaśnienie dla metody *fidelity*. Zamiast x należy wstawić wartość bezwzględną interwału. (Pomarańczowy Kot nie tylko zna język angielski, ale wie również, że jego aplikacja będzie miała więcej użytkowników, jeśli interfejs będzie anglojęzyczny!)

interwał	napis
powyżej +30	off pitch (x cents sharp)
(+20, +30]	weak (x cents sharp)
(+10, +20]	fair (x cents sharp)
(+5, +10]	good (x cents sharp)
(+2, +5]	very good (x cents sharp)
(+1, +2]	excellent (x cents sharp)
[−1, +1]	perfect lub purrfect
[−2, −1)	excellent (x cents flat)
[−5, −2)	very good (x cents flat)
[−10, −5)	good (x cents flat)
[−20, −10)	fair (x cents flat)
[−30, −20)	weak (x cents flat)
poniżej −30	off pitch (x cents flat)

Funkcja *operator<<*

Funkcja umożliwia wypisanie obiektu *pitch_reading* do strumienia wyjściowego — najpierw powinna być wypisywana najbliższa nuta (w notacji naukowej) a po spacji opis zgodny z tabelą 2.

Klasa *pitch_log*

Klasa przechowuje dotychczas zagrane nuty, tzn. obiekty klasy *pitch_reading*, i posiada następujące funkcje:

- Metoda *push_back*, która przyjmuje obiekt klasy *pitch_reading* i dodaje ten obiekt do kontenera typu *std::vector*, będącego polem prywatnym obiektu, na rzecz którego wywoływana jest opisywana metoda.
- Metoda *save_to_file*, która przyjmuje napis typu *std::string* będącego nazwą pliku i która wypisuje zawartość ww. kontenera *std::vector* do pliku o podanej nazwie.

Program — funkcja *main*

Funkcja *main* powinna wczytywać ze standardowego strumienia wejściowego liczby zmiennoprzecinkowe zakończone znakiem *EOF* symulowanym np. poprzez wciśnięcie klawiszy *Ctrl+D*. Wczytane liczby powinny być interpretowane jako częstotliwości zagranych na skrzypcach nut. Na standardowym strumieniu wyjściowym powinny być wypisywane bezpośrednio po wczytaniu każdej liczby informacje o nucie najbliższej danej częstotliwości (w notacji naukowej) i ocenie jakości dźwięku zgodnym z tabelą 2. Na sam koniec, po wczytaniu wszystkich częstotliwości, program powinien zapisać do pliku o nazwie *pitch.log* te same informacje, które zostały wypisane na standardowym strumieniu wyjściowym. Do osiągnięcia opisanej funkcjonalności wystarczy wykorzystać opisane wcześniej funkcje i klasy. Przykładowa sesja programu wygląda następująco (krojem pogrubionym oznaczono dane wejściowe wpisane przez użytkownika a krojem pochylonym — dane wyjściowe wypisane przez program):

440
A4 perfect
460
A#4 weak (-23 cents flat)
440 460 420
A4 perfect
A#4 weak (-23 cents flat)
G#4 fair (19 cents sharp)

Rozporządzenie Pomarańczowego Kota nr 3

Ja, Pomarańczowy Kot, kodujący w C++ w tajemnicy przed światem, niniejszym tym oto dekretem przykazuję, aby na koniec tych zajęć, wszyscy śmiałkowie, którzy podjęli się realizacji tego projektu, wysłali swe rozwiązania – nawet nieukończone – w formie archiwum do oceny przez platformę Kampus. A gdyby zaszła potrzeba uzupełnienia rozwiązań, to godzina 23:59 za ostateczny termin uznana będzie.

Kot

Koci epilog

Pomarańczowy Kot, wbrew obiegowej opinii, jest całkiem inteligentnym zwierzątkiem. Nie tylko zna wartość dobrego snu i jedzenia, ale również w tajemnicy koduje w C++ i uczy się grać na skrzypcach. Tylko sza! Bo jeszcze ludzie się dowiedzą...

$$\frac{\partial^2 u}{\partial t^2} = \text{🐱}^2 \frac{\partial^2 u}{\partial x^2}$$

(równanie falowe)