Chapter 3  \textbf{PROGRAMMING EXERCISE}

Chords in C++

Write one of the two following programs in C or C++.

\textbf{Version A}
Your program should have the following functionality:
- The user can indicate a key in which chords are to be played, including whether the key is major or minor.
- The user then selects the type of chord from
  - major triad
  - minor triad
  - augmented triad
  - diminished triad
  - minor seventh
  - dominant seventh
  - major seventh
- The above activities should be in a loop that continues until the user chooses “quit.”

\textbf{Version B}
Your program should have the following functionality:
- The program randomly determines a key, including whether the key is major or minor.
- The program then randomly generates chords of one of the following types:
  - major triad
  - minor triad
  - augmented triad
  - diminished triad
  - minor seventh
  - dominant seventh
  - major seventh
- The user is given the opportunity to guess the key and chord that is being played, and the program tells the user if the answer is correct. If the answer is not correct, the correct answer is given.
- The above activities should be in a loop that continues until the user chooses “quit.”

In both version A and version B, your program should make chords by writing sound samples directly to the sound device.
Solution:

Below is a solution for Version A.

```c
#include <sys/ioctl.h> //for ioctl()
#include <math.h> //sin(), floor()
#include <stdio.h> //perror
#include <fcntl.h> //open, O_WRONLY
#include <linux/soundcard.h> //SOUND_PCM*
#include <iostream>
#include <stdlib.h>

using namespace std;

#define LENGTH 1 //number of seconds
#define RATE 44100 //sampling rate
#define SIZE sizeof(short) //size of sample, in bytes
#define CHANNELS 1 // number of stereo channels
#define PI 3.14159
#define SAMPLE_MAX 32767 // should this end in 8?
#define MIDDLE_C 262
#define SEMITONE 1.05946
#define SCALE 1
#define CHORD 2
#define QUIT "quit"

eenum chordTypes {BACK = 0, MAJOR, MINOR, DIMINISHED, AUGMENTED, MAJ_7, MIN_7, DOM_7};
eenum intervals {ROOT = 0, THIRD, FIFTH, SEVENTH};

double getFreq(int index)
{
    return MIDDLE_C * pow(SEMITONE, index - 1);
}

int getInput()
{
    char c;
    string str;
    int i;
    while ((c = getchar()) != '\n' && c != EOF)
    {
        str += c;
        for (i = 0; i < str.length(); ++i)
            str.at(i) = tolower(str.at(i));
        if (c == EOF || str == QUIT)
            exit(0);
    }
    return atoi(str.c_str());
}

void writeToSoundDevice(short buf[], int buffSize, int deviceID) {
    int status;
    status = write(deviceID, buf, buffSize);
    if (status != buffSize)
```

This material is based on work supported by the National Science Foundation under CCLI Grant DUE 0717743, Jennifer Burg PI, Jason Romney, Co-PI.
perror("Wrote wrong number of bytes\n");
status = ioctl(deviceID, SOUND_PCM_SYNC, 0);
if (status == -1)
perror("SOUND_PCM_SYNC failed\n");
}

int mainMenu()
{
cout << "Choose one of the following:\n"
<< "t" << SCALE << " to hear a scale\n"
<< "t" << CHORD << " to hear a chord\n"
<< "tor type \n" << QUIT << " to quit\n";
return getInput();
}

int getIndex()
{
int input;
cout << "Choose one of the following:\n"
<< "t1) C\n"
<< "t2) C sharp/D flat\n"
<< "t3) D\n"
<< "t4) D sharp/E flat\n"
<< "t5) E\n"
<< "t6) F\n"
<< "t7) F sharp/G flat\n"
<< "t8) G\n"
<< "t9) G sharp/A flat\n"
<< "t10) A\n"
<< "t11) A sharp/B flat\n"
<< "t12) B\n"
<< "t" << BACK << " to back up\n"
<< "tor type \n" << QUIT << " to quit\n";
input = getInput();
if (! (input >= BACK && input <= 12))
return -1;
return input;
}

int getChordType()
{
int input;
cout << "Choose one of the following:\n"
<< "t" << MAJOR << " major triad\n"
<< "t" << MINOR << " minor triad\n"
<< "t" << DIMINISHED << " diminished triad\n"
<< "t" << AUGMENTED << " augmented triad\n"
<< "t" << MAJ_7 << " major 7th\n"
<< "t" << MIN_7 << " minor 7th\n"
<< "t" << DOM_7 << " dominant 7th\n"
<< "t" << BACK << " to back up\n"
<< "tor type \n" << QUIT << " to quit\n";
input = getInput();
if (! (input == BACK || (input >= MAJOR && input <= DOM_7)))
    return -1;
return input;
}

int getScaleType()
{
    int input;
    cout << "Choose one of the following:
        \"t\" << MAJOR << ",\n        \"t\" << MINOR << ",\n        \"t\" << BACK << ",\n        \"tor type \"n" << QUIT << ",\n    input = getInput();
    if (! (input == BACK || (input >= MAJOR && input <= MINOR)))
        return -1;
    return input;
}

void playChord(int deviceID)
{
    int index,
        arraySize,
        note,
        chordType,
        f,
        t;
    double freq[4];
    short *buf;
    while ((index = getIndex()) < 0)
        cout << "Input out of bounds. Try again.\n"
    if (index == BACK)
        return;
    f = getFreq(index);
    freq[ROOT] = 0;
    while ((chordType = getChordType()) < 0) {
        cout << "chordType = " << chordType << "<" chordType << endl;
        cout << "Input out of bounds. Please try again.\n"
    }
    switch (chordType) {
    case MAJOR :
        freq[THIRD] = 4;
        freq[FIFTH] = 7;
        freq[SEVENTH] = -1;
        break;
    case MINOR :
        freq[THIRD] = 3;
        freq[FIFTH] = 7;
        freq[SEVENTH] = -1;
        break;
    case DIMINISHED :
        freq[THIRD] = 3;
freq[FIFTH] = 6;
freq[SEVENTH] = -1;
break;
case AUGMENTED :
    freq[THIRD] = 4;
    freq[FIFTH] = 8;
    freq[SEVENTH] = -1;
    break;
case MAJ_7 :
    freq[THIRD] = 4;
    freq[FIFTH] = 7;
    freq[SEVENTH] = 11;
    break;
case MIN_7 :
    freq[THIRD] = 3;
    freq[FIFTH] = 7;
    freq[SEVENTH] = 10;
    break;
case DOM_7 :
    freq[THIRD] = 4;
    freq[FIFTH] = 7;
    freq[SEVENTH] = 10;
    break;
case BACK :
    playChord(deviceID);
default:
    cout << "wat
";
}
for (note = ROOT; note <= SEVENTH; ++note) {
    if (freq[note] < 0)
        continue;
    freq[note] = f * pow(SEMITONE, freq[note]);
}
arraySize = LENGTH*RATE*CHANNELS;
buf = new short[arraySize];
for (t = 0; t < arraySize; ++t) {
    buf[t] = 0;
    for (note = 0; note <= SEVENTH; ++note) {
        f = freq[note];
        if (f < 0)
            continue;
        buf[t] += floor((SAMPLE_MAX/4) * (sin(2*PI*f*t/RATE)));
    }
}
writeToSoundDevice(buf, arraySize*SIZE, deviceID);
delete buf;
return;

void playScale(int deviceID) {
    int
        arraySize,
        note,
        steps,
index,
break1, // only one half step to here
break2, // only one half step to here
scaleType,
t,
off;
double
f;
short
*buf;
arraySize = 8 * LENGTH * RATE * CHANNELS;
buf = new short[arraySize];
while ((index = getIndex()) < 0)
cout << "Input out of bounds. Please try again.\n";
if (index == BACK)
return;
f = getFreq(index);
while ((scaleType = getScaleType()) < 0)
cout << "Input out of bounds. Please try again.\n";
switch (scaleType) {
case MAJOR :
break1 = 3;
break2 = 7;
break;
case MINOR :
break1 = 2;
break2 = 5;
break;
case BACK :
playScale(deviceID);
break;
default :
cout << "Input out of bounds. Please try again.\n";
playScale(deviceID);
}
arraySize = LENGTH * RATE * CHANNELS;
for (note = off = 0; note < 8; ++note, off += t) {
if (note == 0)
steps = 0;
else if (note == break1 || note == break2)
steps = 1;
else
steps = 2;
f *= pow(SEMITONE, steps);
for (t = 0; t < arraySize; ++t)
buf[t + off] = floor(SAMPLE_MAX*sin(2*PI*f*t/RATE));
}
arraySize = 8 * LENGTH * RATE * SIZE * CHANNELS;
writeToSoundDevice(buf, arraySize, deviceID);
delete buf;
return;
}

int main() {


int deviceID, arg, status, index;
deviceID = open("/dev/dsp", O_WRONLY, 0);
if (deviceID < 0)
    perror("Opening /dev/dsp failed\n");
arg = SIZE * 8;
status = ioctl(deviceID, SOUND_PCM_WRITE_BITS, &arg);
if (status == -1)
    perror("Unable to set sample size\n");
arg = CHANNELS;
status = ioctl(deviceID, SOUND_PCM_WRITE_CHANNELS, &arg);
if (status == -1)
    perror("Unable to set number of channels\n");
arg = RATE;
status = ioctl(deviceID, SOUND_PCM_WRITE_RATE, &arg);
if (status == -1)
    perror("Unable to set sampling rate\n");

while (true) {
    switch(mainMenu()) {
    case SCALE :
        playScale(deviceID);
        break;
    case CHORD :
        playChord(deviceID);
        break;
    case BACK :
        break;
    default :
        cout << "Input out of bounds. Please try again.\n"
        break;
    }
}