SPEECH SYNTHESIS FOR THE TESTING OF SENSORY AIDS FOR THE HEARING IMPAIRED

A dissertation submitted in partial fulfillment of
the requirements for the degree of
Master of Technology

By

Shailendra A. Chafekar

Roll No. 883729

Guides

Prof. K.J. Gazder

Dr. P.C. Pandey

School of Biomedical Engineering

Indian Institute of Technology, Bombay

July 1990

DISSERTATION APPROVAL SHEET

Dissertation entitled " **SPEECH SYNTHESIS FOR THE TESTING OF SENSORY AIDS FOR THE HEARING IMPAIRED**" is approved for the award of the degree of MASTER OF TECHNOLOGY in BIOMEDICAL ENGINEERING.

GUIDE	CO-GUIDE	CHAIRMAN	EXAMINERS

ABSTRACT

A software based speech synthesizer with the flexibility for introducing controlled changes in the characteristics of the speech signal, e.g., fundamental frequency, voicing and frication amplitudes, stress patterns, and formant frequencies and amplitudes etc, is a useful tool in testing and calibration of various sensory aids for the hearing impaired. One such synthesizer, the Klatt synthesizer, uses an all-pole model to approximate the vocal tract: vowels are synthesized by employing a cascade model, where as fricatives are synthesized by using a parallel model. Here the anti-resonances in the spectra of fricatives are approximated by adjusting the amplitude controls of the resonators connected in parallel.

In this thesis, a software based speech synthesizer which uses a pole-zero cascade model for the vocal tract has been developed. The cascade model should provide a more natural representation of the vocal tract, and should be better for synthesis in Indian languages, where aspiration is a phonemic contrast feature among many stop and affricate consonants.

A program SPSYNTH has been developed, implementing the scheme of the all-pole cascade/parallel synthesis (of Klatt synthesizer), as well as the new scheme of pole-zero cascade synthesis. Parameter tracks can be specified graphically. The program also offers an in-built set of commonly occurring phonemes, for synthesizing longer speech sequences. This phoneme set can be appended by specifying parameter tracks for user-defined phonemes. This feature would be useful in synthesizing speech of different accents, or pathological disorders.

The program has been tested by generating sample speech segments extending up to three seconds, under informal listening tests.

ACKNOWLEDGEMENTS

I sincerely thank Prof. K.J. Gazder for his able help during the project.

I am extremely thankful to Dr. P.C. Pandey for the constant help and encouragement that he provided during the entire project work.

I would also like to thank Mr. Anil Vartak of the Dept. of Electrical Engineering for his help in sorting out the problems with the D/A converter card, and in taking hard copies of the program listings.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

In the areas of psychoacoustic studies and speech sciences there is often a need for a flexible generator of speech and speech-like stimuli. The speech perception experiments need a source capable of producing the same stimuli repeatedly. Such a source should be flexible enough to allow the experimenter to alter its various parameters in a variety of ways, and to produce sounds which may even defy the normal relationships between the various parameters.

Such a flexible synthesizer would be useful in the testing and calibration of various sensory aids for the hearing impaired like hearing aids, vibro-tactile aids, visual aids, and cochlear prostheses. These devices may be employing techniques like dynamic range compression, frequency selective amplification and a variety of other complex signal processing techniques (Watson & Knudsen, 1940; Vilchur, 1973; Johansson, 1966; Kirman, 1974; Levitt, 1973; Pandey, 1987). Characterization and testing of these devices by the magnitude and phase response measurements using sinusoidal excitation may not be very relevant. Natural speech segments are used for the overall device performance measurement, but most of these results usually do not provide an insight

into the causes for the relative success or failure of the devices (Klatt, 1980; Pandey, 1987). Instead if a synthesized speech stimuli is used, it would be possible to get a detailed assessment of the device performance with respect to the various features of speech signals: the pitch or fundamental frequency, formant frequencies, the shape of the source or glottal spectrum, relative formant amplitudes, etc. The ability to produce sounds with very subtle differences as regards these parameters would be very useful in these applications.

1.2 PROJECT OBJECTIVES

This project is aimed at developing a flexible software based speech synthesizer for the above mentioned applications. The synthesizer will be based on the one developed by Dennis Klatt at MIT (Klatt, 1980). The Klatt synthesizer uses an all pole cascade/ parallel model of the vocal tract. The vowels are simulated through a cascade model, while the fricatives and stops through a parallel model. The parallel model needs careful adjustments to mimic the presence of zeros in the spectra of fricatives.

This work is directed toward developing and testing a "pole-zero cascade" synthesizer. A cascade model represents the vocal tract more precisely, and if the locations of zeros are found out fairly accurately, it should be able to model fricative spectra more accurately. This

is particularly important for synthesis in Indian languages, where aspiration forms one of the major contrast feature for stops. This scheme would also be suitable for hardware realization for real-time applications. By combining pairs of poles and zeros the total number of delays needed can be reduced.

1.3 OUTLINE OF THE REPORT

The second chapter provides an overview of speech synthesis. The Klatt synthesizer and its implementation in the general cascade/parallel and the special all-parallel modes are described in the third chapter. Chapter 4 describes the pole-zero cascade synthesizer and its implementation. It also describes how to find zero locations from the data used on the general Klatt synthesizer. Chapter 5 contains a description of the program. Testing of both the all-pole cascade/parallel and the pole-zero cascade programs may be found in Chapter 6. Chapter 7 provides a summary of the work outlined in this dissertation, and suggestions for further work. Appendix A contains the listings of the programs described in this thesis, and Appendix B describes the power amplifier which was used for listening to the synthesized speech segments.

CHAPTER 2

OVERVIEW OF SPEECH SYNTHESIS

2.1 INTRODUCTION

Speech synthesis is the machine generation of speech. Since long man has been taking an interest in trying out different methods of producing artificial speech. One of the earliest recorded efforts was a mechanical device built by Kratzenstein in 1779. He constructed acoustic resonators similar in shape to the human vocal tract, which were activated with vibrating reeds which represented the vocal cords (Flanagan, 1972).

The developments in electrical technology led to the evolution of techniques for electrical methods of speech synthesis. In this chapter, an overview of electrical speech synthesizers will be provided.

2.2 CLASSIFICATION OF SPEECH SYNTHESIZERS

The first electrical speech synthesizer reported by Dudley in 1939 at the Bell Laboratories (Flanagan, 1972), used a "vocoder" or voice coder approach, employing a bank of filters which estimate the energy of speech signals in different frequency bands. Since then a large number of speech synthesizers for software as well as hardware implementation have been reported (Flanagan et al, 1970; Gold & Rabiner, 1968;

Flanagan, 1972; Rabiner & Gold, 1975; Klatt, 1980; Rabiner & Schafer, 1978; O'Shaughnessy, 1987).

The electrical speech synthesizers can be classified as articulatory and formant synthesizers. Articulatory synthesis is based on describing the vocal tract behaviour in terms of its detailed distributed properties. These synthesizers either employ an approximation of the vocal tract as a set of interconnected tube sections of varying cross-sectional areas and equal length, as shown in Fig. 2.1, or, they employ an articulatory model. Formant synthesizers represent the vocal tract as seen from its input and output and are also known as terminal analog synthesizers. Their approach is to implement a system whose transfer function approximates that of the vocal tract, but which does not necessarily bear any structural similarity with the vocal tract (Oppenheim, 1978; O'Shaughnessy, 1987).

2.2.1 Articulatory Synthesizers

These model the dynamic behaviour and movements of various articulators and try to approximate the resulting vocal tract shapes. The constraints on such a model are that it should represent the vocal tract shape accurately, and that it should model fairly well the continuities of the articulator surfaces and the discontinuities at articulator boundaries (Flanagan et al, 1970). A typical articulatory model is shown in Fig. 2.2. It uses seven parameters to describe the area of the vocal

tract as a function of the distance from the glottis: two coordinates each for position of the lips (W,L), position of the tongue tip (R,B), position of the tongue body (X,Y), and one coordinate for the position of the velum (N).

2.2.2 Formant Synthesizers

A formant synthesizer models the vocal tract resonances or "formants". For different speech sounds the vocal tract assumes different shapes. Thus its resonances occur at different sets of frequencies. Formant synthesizers use the speech production model shown in Fig. 2.3. This model assumes the so called source-tract independence, i.e., it is assumed that the properties of the vocal tract can be specified independently to those of the excitation source, which is typically either a periodic pulse excitation or a noise excitation.

In Fig. 2.3, if the source excitation be $e(t)$, the vocal tract impulse response be $h(t)$, then the lip volume velocity is given as

$$u(t) = e(t) * h(t) \tag{2.1}$$

Or, in frequency domain representation we have

$$U(j\Omega) = E(j\Omega)\, H(j\Omega) \tag{2.2}$$

The speech spectrum is further modified by radiation characteristics of the lips, $R(j\Omega)$. Hence the net output speech is given by

$$S(j\Omega) = E(j\Omega)\, H(j\Omega)\, R(j\Omega) \tag{2.3}$$

The vocal tract transfer function can be represented by a

linear time varying filter as shown in Fig. 2.4.

Generally the vocal tract is represented either by an all-pole cascade model or an all-pole parallel model (Klatt, 1980). The cascade configuration models the vocal tract more accurately. Here the relative amplitudes of the formants will be correct. But it is inadequate to represent the spectral zeros in fricative spectra. For this a parallel model must be used. Locations of spectral zeros are adjusted by carefully changing the formant amplitudes. Since an all-pole cascade model is appropriate for vowels and an all-pole parallel model is needed for fricatives, it becomes imperative to use a cascade/parallel model for speech segments involving both vocalization and fricative excitations (Klatt, 1980).

2.2.3 Articulatory vs Formant Synthesizers

Articulatory synthesizers are more suited to model the finer, controlled movements of the tongue, especially during vowel-stop-vowel sequences. But it is difficult to obtain complete data for articulator movements inside the vocal tract. The only technique available is X-ray imaging (O'Shaughnessy, 1987; Flanagan et al, 1970; Flanagan, 1972). Further the articulatory synthesis is more difficult than the formant synthesis, which is based on a simpler set of rules. The data needed for formant synthesis can be readily obtained by speech analysis.

2.3 HARDWARE VS SOFTWARE SYNTHESIZERS

A number of integrated circuits are currently available for real-time speech synthesis applications (O'Shaughnessy, 1987). But most of these chips lack in flexibility. Usually they have an in-built set of formant frequencies. Hence for psychoacoustic and speech perception studies they lack the flexibility, which is very useful in these applications. A software based speech synthesizer can provide all the flexibility for controlling the parameters as needed for generating test stimuli.

Although the software simulation can not offer real-time speech synthesis, it is not a handicap for applications such as testing of sensory aids for the hearing impaired (hearing aids, vibro-tactile aids, visual aids, cochlear prostheses, etc), and for psychoacoustic studies.

2.4 A SOFTWARE BASED SPEECH SYNTHESIZER FOR INDIAN LANGUAGES

The model which has been developed in this project is a cascade pole-zero model. This would represent the vocal tract more accurately for both vowels as well as fricatives. One more reason why a pole-zero cascade model would be appropriate for Indian languages is that in all Indian languages aspiration plays an important role as a contrast feature. Aspiration is an 'h'-like sound that causes more airflow through the vocal tract. The source of aspiration is at the glottis. Aspiration

occurs after the plosive release in a stop or an affricate. Aspiration can be both voiced or unvoiced. Thus we find both unaspirated and aspirated versions of affricate and stops for the same articulatory configuration. In addition all the Indian languages contain a whole repertoire of retroflex sounds, both stops and fricatives. Table 2.1 shows the stop, affricate, fricative and nasal consonants of Hindi and English, along with the place of articulation.

Table 2.1. Stop, affricate, fricative, and nasal consonants of Hindi and English, along with the place of articulation. Hindi consonants are shown in Devanagari letters and English consonants are shown in IPA (International Phonetic Alphabet) symbols. Pronunciation key to some English consonants is given at the bottom of the table. UV = unvoiced, VO = voiced, UA = unaspirated, AS = aspirated. Adapted from Pandey (1987), Table 2.1.

Place of articulation	Stop				Fricative		Affricate				Nasal
	UV		VO		UV	VO	UV		VO		VO
	UA	AS	UA	AS	UA	UA	UA	AS	UA	AS	UA
Velar	क् k	ख्	ग् g	घ्							ङ् ŋ
Palatal					श् ʃ	ʒ	च् tʃ	छ्	ज् dʒ	झ्	ञ्
Alveolar/ Retroflex	ट्	ठ्	ड्	ढ्	ष्						ण्
Alveolar	t		d		स् s	z					n
Dental	त्	थ्	द्	ध्	θ	ð					न्
Labio-dental					f	v					
Bi-labial	प् p	फ्	ब् b	भ्							म् m

Pronunciation Key

tʃ	chain	ʃ	shin	s	sin	θ	thin	f	fin
dʒ	jail	z	vision	z	zap	ð	this	v	vision
ŋ	rang								

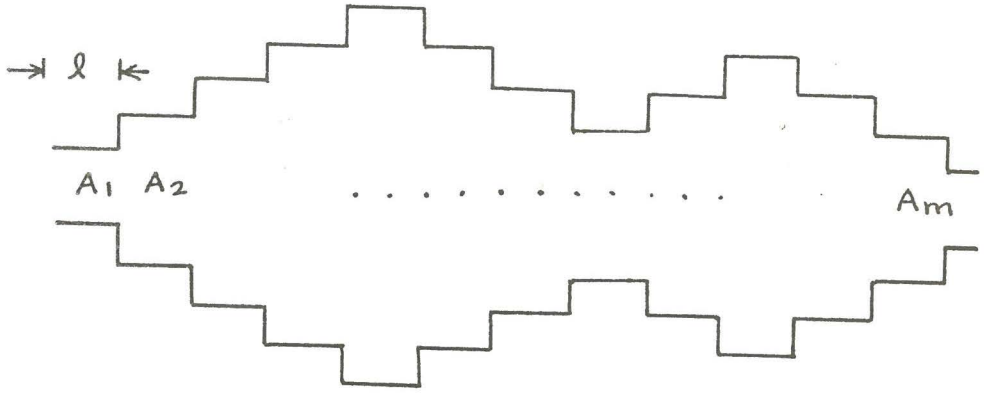


Fig. 2.1. Acoustic tube analog of the vocal tract. Adapted from Oppenheim (1978), Fig. 3.7.

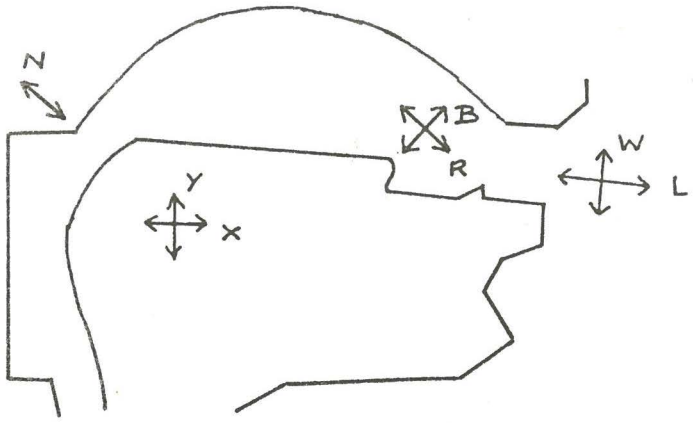


Fig. 2.2. Articulatory model of the vocal tract. Adapted from Flanagan et al (1970), Fig. 21.

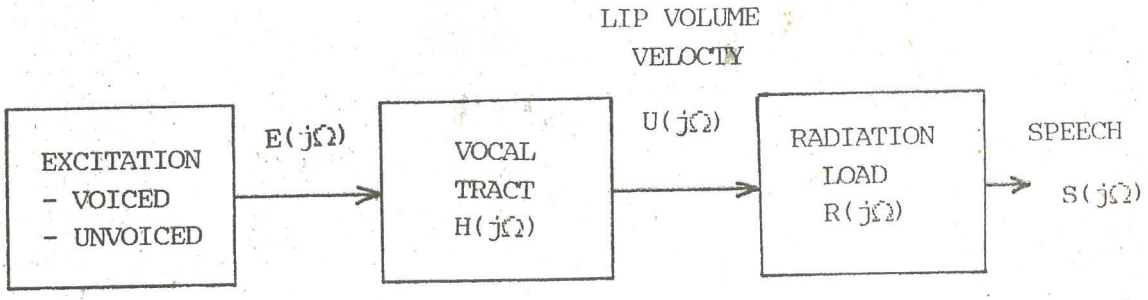


Fig. 2.3. Model of speech production. Adapted from Oppenheim (1978), Fig. 3.5a.

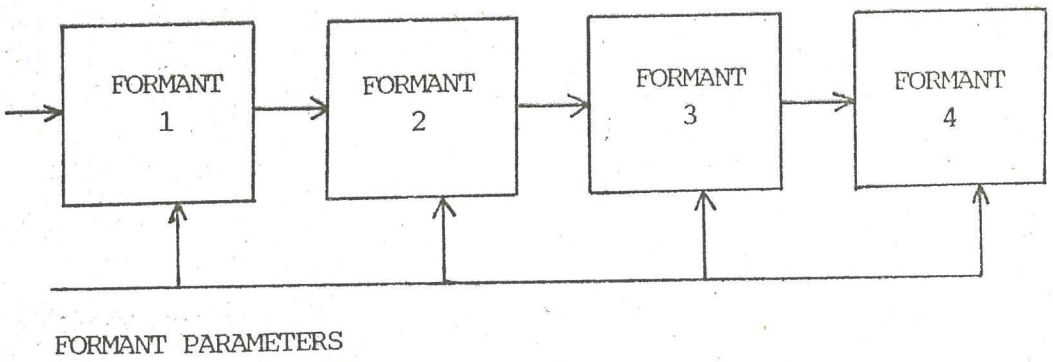


Fig. 2.4. Vocal tract transfer function in a typical formant synthesizer. Adapted from Oppenheim (1978), Fig. 3.5b.

CHAPTER 3

THE KLATT SYNTHESIZER

3.1 INTRODUCTION

The Klatt Synthesizer (Klatt, 1980) is a software based formant synthesizer developed by Dennis Klatt at MIT. The source code is available in FORTRAN. It is a flexible synthesizer: as many as 39 control parameters can be specified of which about 20 can be varied as functions of time. Information about the manner in which the parameters are varied (the *"parameter tracks"*) and values of the constant parameters are stored in a file. The parameters can be updated every 2-20 milliseconds. The output of the program is a file that contains a sequence of speech sample values.

The building block of the synthesizer is a digital resonator as shown Fig. 3.1. The output sequence $y(n)$ is related to the input sequence $x(n)$ as

$$y(n) = A\,x(n) + B\,y(n-1) + C\,y(n-2) \tag{3.1}$$

The difference equation coefficients are given by

$$\begin{aligned} C &= -\exp(2\pi BWT) \\ B &= 2\exp(\pi BWT)\cos(2\pi FT) \\ A &= 1 - B - C \end{aligned} \tag{3.2}$$

In Eqn. 3.2, F and BW are the centre frequency and bandwidth of the

resonator respectively, and T is the sampling interval. The magnitude response of this resonator is shown in Fig. 3.1, where,

$$T(z) = \frac{A}{1 - B z^{-1} - C z^{-2}} \qquad (3.3)$$

A digital antiresonator can be implemented similarly. It is done by replacing A, B, and C by $1/A$, $-1/B$, and $-1/C$ respectively.

3.2 IMPLEMENTATION

The Klatt synthesizer can be used in two ways: either in the general cascade/parallel mode, or in a special all-parallel mode as shown in Fig. 3.2. The figure also indicates the sources of excitation. In the all-parallel mode, vowels are also synthesized with a parallel bank of resonators.

The detailed block diagram of the Klatt Synthesizer is shown in Fig. 3.3. The 39 control parameters of the synthesizers are listed in Table 3.1. The table also shows the minimum and maximum values of the parameters, and their type; whether variable or constant. As can be seen as many as six formants can be employed by the user.

The following sections describe the various aspects of the implementation of the Klatt synthesizer.

3.2.1 Sources of excitation

There are two types of excitation sources. One is a periodic impulse train, which is lowpass filtered by the resonator RGP to produce a typical glottal excitation pulse. The second is a noise source, which represents the noise produced by turbulent air flow through a constriction during the production of fricatives. The same source can be used to produce aspiration. The periodic pulse train is used to modulate the output of the noise source during the production of voiced fricatives and stops.

Sometimes during the production of voiced fricatives a smoothed quasi-sinusoidal voicing is needed. Resonator RGS further filters the glottal pulse to yield such an excitation.

The frication source is typically simulated by a pseudo-random generator. The noise source is an ideal pressure source, and hence its output must be lowpass filtered to give its equivalent volume velocity.

During the production of voiced fricatives the noise is modulated pitch synchronously.

3.2.2 Control of Sources Amplitudes

The amplitudes of voiced and unvoiced sources are controlled by parameters AV and AF respectively. The amplitude of aspiration is controlled by AH. Changes in these parameters take place pitch synchronously when voicing is present. It is also possible to specify sudden

bursts for plosive releases. A value of 60 dB for AV produces strong voicing. Similarly a value of 60 dB for AF and AH gives rise to strong frication and aspiration respectively.

3.2.3 Vocal Tract Transfer Function

This can be represented either as a cascade or parallel combination of resonators, each tuned to a different formant frequency. Each resonator can be represented by

$$T(z) = \frac{A(n)}{1 - B(n)\, z^{-1} - C(n)\, z^{-2}} \qquad (3.4)$$

Where $A(n)$, $B(n)$, and $C(n)$ are calculated from the n^{th} formant frequency as in Eq. 3.2. The transfer function is scaled in such a way that there is a free flow at zero frequency. Fig. 3.4 shows the typical transfer functions for a uniform tube and for the vowels /i/, /a/, and /u/ respectively.

The nasals are simulated by a nasal antiresonator RNZ. During the production of non-nasals its action is nullified by a resonator RNP which is set to the same frequency as RNZ.

In the parallel mode individual formant controls A1 through A6, a nasal pole amplitude control AN, and a bypass path amplitude control are provided.The bypass path is needed for sounds like /v/, /f/, /s/, and /sh/ etc, which show an almost uniform frequency response and no

prominent peaks.

During the production of voiced obstruents two sources excite the vocal tract. The glottal pulses are sent through the cascade model and the noise through the parallel model. In the special all-parallel mode, both the sources excite the parallel model. During the production of vowels the all-parallel model approaches the cascade model if all formant amplitude controls are set to 60 dB.

3.2.4 Formant Bandwidths

The formant bandwidths represent losses in the vocal tract like yielding walls, thermal and friction losses etc. More importantly, the bandwidth of F1 is directly proportional to the interaction of the supraglottal cavity with the subglottal space. For example, during the production of unvoiced stops the bandwidth of the first formant is usually greater than that for the corresponding voiced stops which have the same articulatory configuration.

3.2.5 Radiation Characteristics

The radiation load at the lips acts as a first order highpass filter. Hence the radiation characteristics is a first differnce equation. To make the model computationally more efficient, the radiation characteristics is moved into the source shaping filters. Thus for fricatives, the lowpass characteristics to convert noise pressure into

equivalent volume velocity and the radiation characteristics and the radiation characteristics cancel each other.

3.3 TESTING OF THE KLATT SYNTHESIZER

The Klatt Synthesizer program has been re-written in PASCAL as it was decided to write the program for the pole-zero cascade synthesizer, to be described in the next chapter, also in PASCAL. Both the synthesizer implementations have been integrated. Thus both the synthesizers are accessible through a single program. This would help in comparing the results produced by the two implementations. Different methods of specifying parameter tracks have been provided. The description and testing of this integrated program will be covered in Chapters 5 and 6. The listing of the synthesizer program may be found in Appendix A.

Table 3.1. Control parameters for the all-pole cascade/parallel synthesizer. The list also shows the permitted ranges of values for each parameter, and a typical value. V/C indicates whether the parameter is normally variable (V), or constant (C).
Adapted from Klatt (1980), Table I.

N	V/C	Sym	Name	Min	Max	Typ
1	C	SW	Cascade/parallel switch	0(Cas)	1(Par)	0
2	C	NF	Number of formants	4	6	4
3	V	F0	Fundamental freq. of voicing (Hz)	0	500	0
4	V	AV	Ampl. of voicing (dB)	0	80	0
5	V	AF	Ampl. of frication (dB)	0	80	0
6	V	AS	Ampl. of sinusoidal voicing (dB)	0	80	0
7	V	AH	Ampl. of aspiration (dB)	0	80	0
8	V	F1	First formant freq. (Hz)	150	900	450
9	V	F2	Second formant freq. (Hz)	500	2500	1450
10	V	F3	Third formant freq. (Hz)	1300	3500	2400
11	V	BW1	First formant bandwidth (Hz)	40	500	50
12	V	BW2	Second formant bandwidth (Hz)	40	500	70
13	V	BW3	Third formant bandwidth (Hz)	40	500	110
14	C	A1	First formant amplitude (dB)	0	80	0
15	V	A2	Second formant amplitude (dB)	0	80	0
16	V	A3	Third formant amplitude (dB)	0	80	0
17	V	A4	Fourth formant amplitude (dB)	0	80	0
18	V	A5	Fifth formant amplitude (dB)	0	80	0
19	V	A6	Sixth formant amplitude (dB)	0	80	0
20	V	AB	Bypass path amplitude (dB)	0	80	0
21	C	AN	Nasal formant amplitude (dB)	0	80	0
22	V	FNZ	Nasal zero freq. (Hz)	200	700	250
23	V	FNP	Nasal pole freq. (Hz)	200	500	250
24	C	UPDT	Parameter update int. (ms)	2	20	5
25	C	SR	Sampling rate (Hz)	5000	20000	10000
26	C	G0	Overall gain control (dB)	0	80	0
27	C	F4	Fourth formant frequency (Hz)	2500	4500	3300
28	C	F5	Fifth formant frequency (Hz)	3500	4900	3750
29	C	F6	Sixth formant frequency (Hz)	4000	4999	4900
30	C	BW4	Fourth formant bandwidth (Hz)	100	500	250
31	C	BW5	Fifth formant bandwidth (Hz)	150	700	200
32	C	BW6	Sixth formant frequency (Hz)	200	2000	1000
33	C	BWNZ	Nasal zero bandwidth (Hz)	50	500	100
34	C	BWNP	Nasal pole bandwidth (Hz)	50	500	100
35	C	FGP	Glottal res. 1 freq. (Hz)	0	600	0
36	C	BWGP	Glottal res. 1 bandwidth (Hz)	100	2000	100
37	C	FGZ	Glottal zero freq. (Hz)	0	5000	1500
38	C	BWGZ	Glottal zero bandwidth (Hz)	100	9000	6000
39	C	BWGS	Glottal res. 2 bandwidth (Hz)	100	1000	200

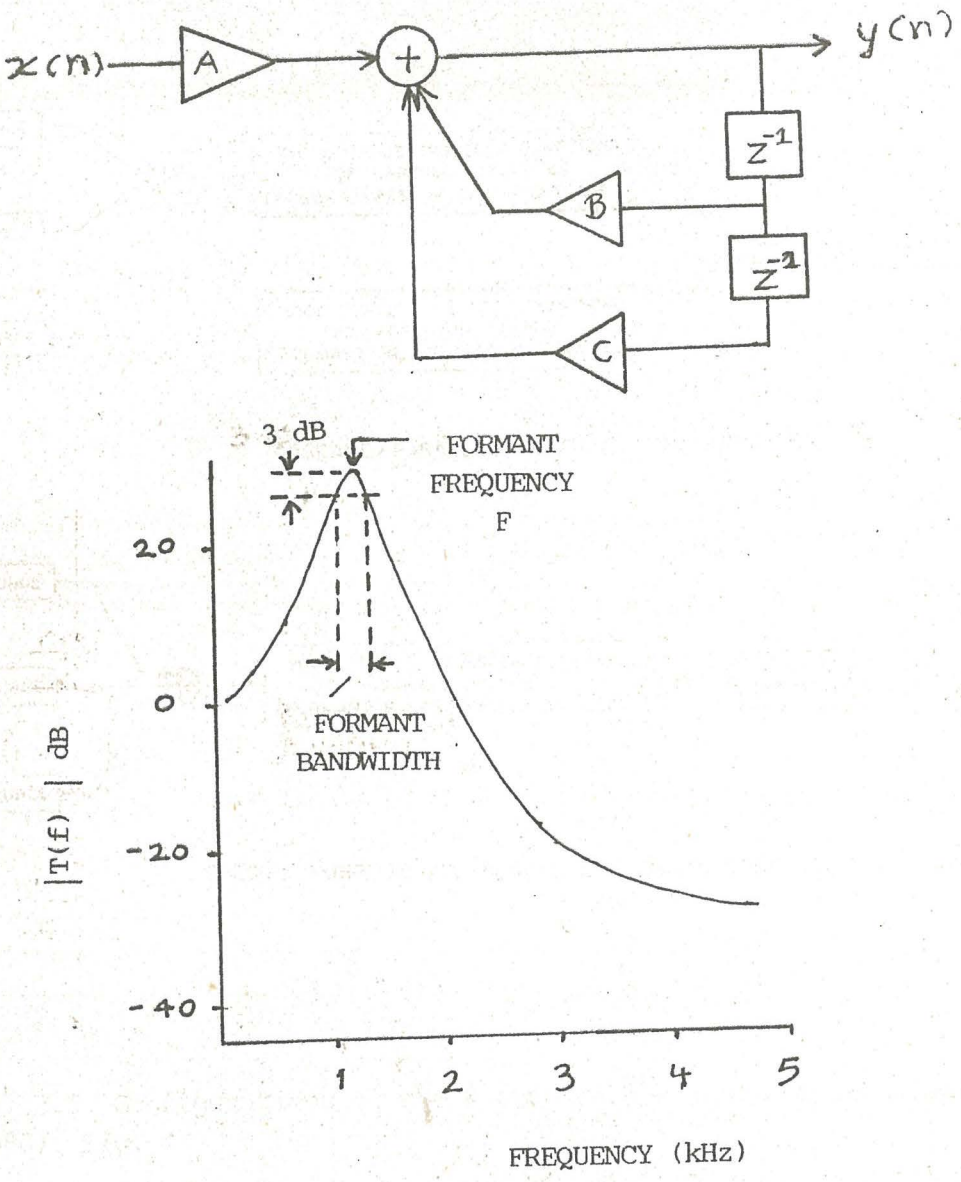
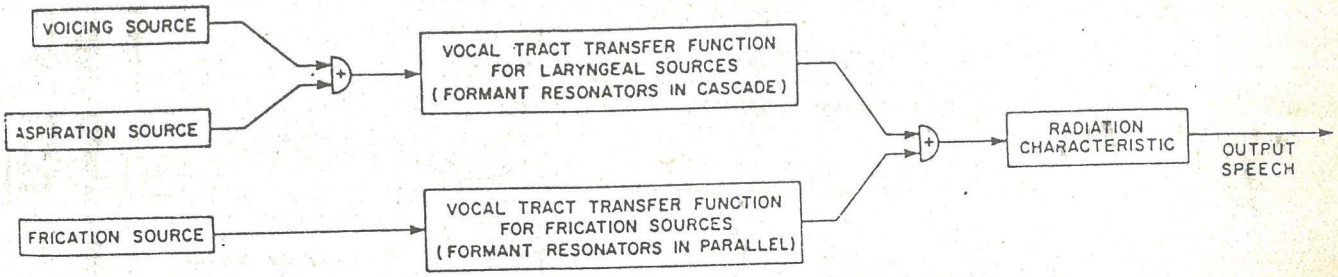
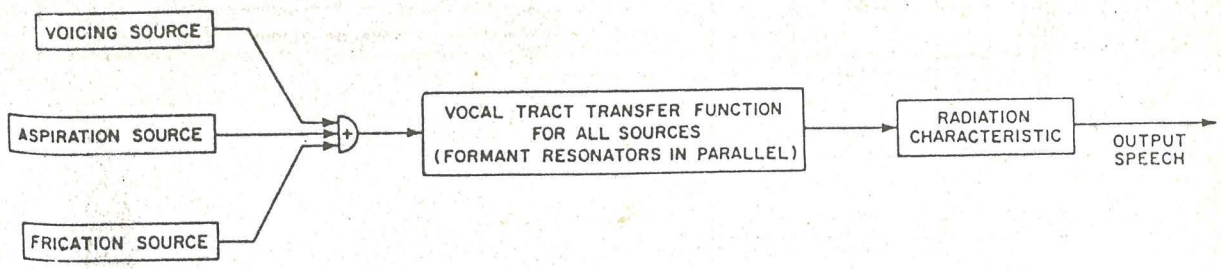


Fig. 3.1. A digital resonator. Adapted from Klatt (1980), Fig. 5.



CASCADE / PARALLEL FORMANT CONFIGURATION



SPECIAL-PURPOSE ALL-PARALLEL FORMANT CONFIGURATION

Fig. 3.2. Configurations of the Klatt synthesizer. Source: Klatt (1980), Fig. 4.

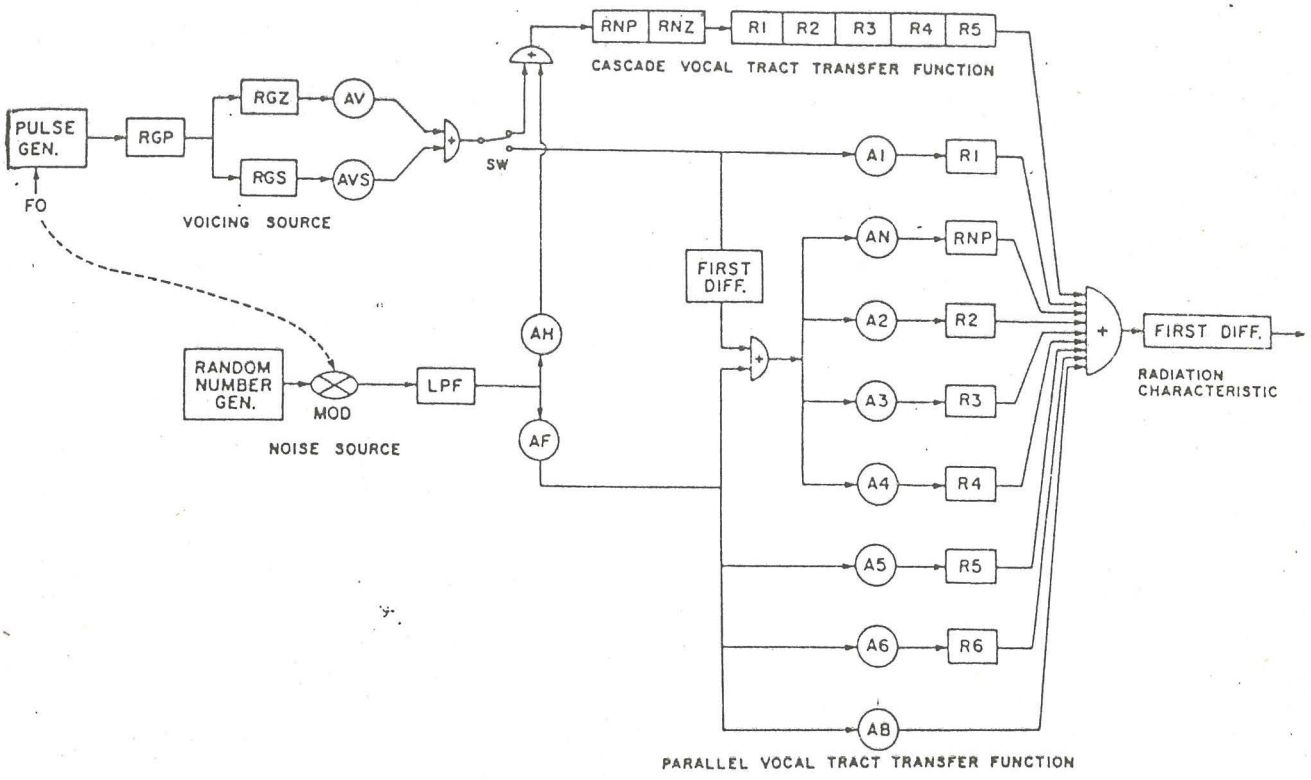


Fig. 3.3. Block diagram of Klatt synthesizer. Source: Klatt (1980), Fig. 6.

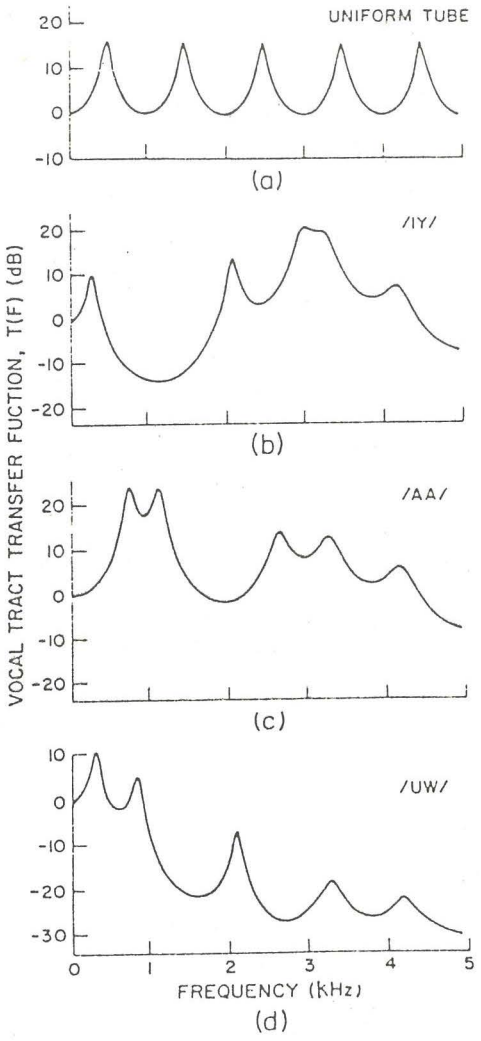


Fig. 3.4. Vocal tract transfer functions for an ideal uniform vocal tract, and for vowels /i/, /a/, and /u/. Source: Klatt (1980), Fig. 9.

CHAPTER 4

THE CASCADE POLE-ZERO SYNTHESIZER

4.1 INTRODUCTION

In Chapter 2, we have suggested the pole-zero cascade synthesizer as a possible improvement over the conventional all-pole cascade/parallel synthesizer. It would be a more natural representation to the vocal tract, as the inter-relationship between various formant amplitudes is represented properly. During the production of vowel-stop-vowel sequences, if a cascade/parallel model is employed, one would have to switch from the cascade to the parallel configuration and vice versa. On the other hand, if a cascade configuration is used throughout and zeros are employed in the production of fricatives and stops there would be no switching, and the resulting speech output therefore would be smoother. Such a synthesizer would also be more relevant in the case of Indian languages, where one can find a whole range of fricatives, stops, and affricates, and where aspiration is one of the phonemic contrast feature among the stops, and affricates. Keeping these things in mind it was decided to develop a pole-zero cascade synthesizer.

The Klatt synthesizer, as already described in the previous chapter, was used as a basis for developing this synthesizer. The locations of zeros in the fricative spectra were calculated using the data pro-

vided by Klatt for the synthesis of phonemes in the speech of a typical American speaker, as described in the following sections.

4.2 CALCULATION OF ZERO LOCATIONS

The zero locations in the spectra of fricatives and stops are calculated by taking into account the amplitudes of the formants of the parallel configuration. The proximity of the formants should also be taken in consideration.

When any system function is realized using a parallel combination of filter sections, zeros are introduced at various points in the system frequency response. The locations, number, and order of these zeros depend on the relative gains of the individual filter sections, their relative locations, and number (Kuc, 1987, pp 218, 219). Consider the following simple example. $H(z)$ is a system function given by

$$H(z) = \frac{A}{1 - az^{-1}} + \frac{B}{1 - bz^{-1}} \qquad (4.1)$$

This will have one first order zero at $\frac{A\,b + a\,B}{A + B}$. The location of the zero will depend on the two gains A and B, and on the location of the poles, a and b. It can be seen that if the gain A is large compared to the gain B, then the zero will be closer to the second pole b. If the reverse is true, then the zero lies more towards the first pole a. If the two amplitudes are comparable then it lies approximately midway

between the two poles.

The procedure for finding the zero frequencies will now be described. The filter coefficients for the second order sections which would otherwise be combined in parallel are calculated from the values of the formant frequencies and bandwidths. The formant amplitudes are properly scaled to take into consideration the proximity of formants. Thus the transfer function of the second order sections which are to be combined in parallel is obtained. These transfer functions are then added. The resulting numerator polynomial is solved for its roots which yield the zero locations and their bandwidths.

A program which accepts the formant frequencies and their bandwidths, and the formant amplitudes, and calculates the zero frequencies and bandwidths has been written in PASCAL, listing of which may be found in Appendix B. The program implements the Lin-Bairstow technique to solve the numerator for its roots (Huelsman, 1986).

4.3 IMPLEMENTATION

The synthesizer block diagram is shown in Fig. 4.1. The model allows up to six resonances and four antiresonances to to be specified. Each resonance (or antiresonance) is in effect a complex conjugate pole (or zero) pair. The number of zeros is a variable parameter, as different fricatives and stops need different number of zeros. Table 4.1 lists the control parameters for the synthesizer.

The program for the pole-zero cascade synthesizer has also been written in PASCAL and integrated with the one written for the all-pole cascade/parallel synthesizer. This allows any one of the two types of synthesizers to be chosen. The integrated program will be described in the next chapter.

There are two sources of excitation, one a periodic impulse generator, and the other, a random number generator. The first serves as a source for voiced sounds, and the other for unvoiced sounds.

The impulse generator outputs an impulse train, whose period is determined by the value of *F0*. For voiced obstruents, the noise is modulated pitch synchronously, by a pulse train, whose period is also determined by *F0*.

The glottal pole-zero pair of RGP and RGZ provide a normal glottal pulse. Additional lowpass filtering for quasi-sinusoidal voicing is achieved by the lowpass filter RGS. Amplitudes of normal and quasi-sinusoidal voicing are controlled by *AV* and *AVS* respectively.

The random number generator acts as the source of both frication and aspiration, the amplitudes of which are controlled by *AF* and *AH* respectively. In this implementation, only one of the two (*AF* and *AH*) should be non-zero at a time. The noise source is a pressure source, hence, the noise is lowpass filtered to obtain the equivalent volume velocity.

The vocal tract is represented by a cascade of nasal pole-zero pair

(RNP and RNZ), six formants (R1-R6), and four antiresonators (RZ1-RZ4). The nasal pole and zero are set to the same frequency during non-nasal sounds, and hence cancel each other's effect. During non-nasal sounds however, RNZ is tuned to a appropriate frequency to simulate nasalization.

The cascade of antiresonators is used only when *AF* is greater than zero. Thus it is bypassed when non-obstruent sounds are produced. The obstruent/non-obstruent switch is employed to achieve this.

At the output, radiation load is simulated by a first difference equation (a highpass filter). This can be moved into the source shaping filters themselves. Thus in the case of the noise source, the effects of the lowpass and highpass filters cancel out.

The parameter tracks for this synthesizer are also generated in two ways, as will be discussed in Chapter 5. The output of the program, a sequence of speech samples, is stored as 12-bit integer values in a sequential file, which can be later used with a D/A converter. The testing of both the all-pole cascade/parallel and pole-zero cascade synthesizers will be covered in Chapter 6. The listings of the synthesizer program, the program for finding antiresonance locations, and the D/A converter driver program have been provided in Appendix A.

Table 4.1. Control parameters for the pole-zero cascade synthesizer. The list also shows the permitted ranges of values for each parameter, and a typical value. V/C indicates whether the parameter is normally variable (V), or constant (C).

N	V/C	Sym	Name	Min	Max	Typ
1	C	NF	Number of formants	4	6	4
2	V	NZ	Number of zeros	0	4	0
3	V	F0	Fundamental freq. of voicing (Hz)	0	500	0
4	V	AV	Ampl. of voicing (dB)	0	80	0
5	V	AF	Ampl. of frication (dB)	0	80	0
6	V	AS	Ampl. of sinusoidal voicing (dB)	0	80	0
7	V	AH	Ampl. of aspiration (dB)	0	80	0
8	V	F1	First formant freq. (Hz)	150	900	450
9	V	F2	Second formant freq. (Hz)	500	2500	1450
10	V	F3	Third formant freq. (Hz)	1300	3500	2400
11	V	BW1	First formant bandwidth (Hz)	40	500	50
12	V	BW2	Second formant bandwidth (Hz)	40	500	70
13	V	BW3	Third formant bandwidth (Hz)	40	500	110
14	V	FZ1	First zero frequency (Hz)	150	5000	-
15	V	FZ2	Second zero frequency (Hz)	150	5000	-
16	V	FZ3	Third zero frequency (Hz)	150	5000	-
17	V	FZ4	Fourth zero frequency (Hz)	150	5000	-
18	V	BZ1	First zero bandwidth (Hz)	-	-	-
19	V	BZ2	Second zero bandwidth (Hz)	-	-	-
20	V	BZ3	Third zero bandwidth (Hz)	-	-	-
21	V	BZ4	Fourth zero bandwidth (Hz)	-	-	-
22	V	FNZ	Nasal zero freq. (Hz)	200	700	250
23	V	FNP	Nasal pole freq. (Hz)	200	500	250
24	C	UPDT	Parameter update int. (ms)	2	20	5
25	C	SR	Sampling rate (Hz)	5000	20000	10000
26	C	G0	Overall gain control (dB)	0	80	0
27	C	F4	Fourth formant frequency (Hz)	2500	4500	3300
28	C	F5	Fifth formant frequency (Hz)	3500	4900	3750
29	C	F6	Sixth formant frequency (Hz)	4000	4999	4900
30	C	BW4	Fourth formant bandwidth (Hz)	100	500	250
31	C	BW5	Fifth formant bandwidth (Hz)	150	700	200
32	C	BW6	Sixth formant frequency (Hz)	200	2000	1000
33	C	BWNZ	Nasal zero bandwidth (Hz)	50	500	100
34	C	BWNP	Nasal pole bandwidth (Hz)	50	500	100
35	C	FGP	Glottal res. 1 freq. (Hz)	0	600	0
36	C	BWGP	Glottal res. 1 bandwidth (Hz)	100	2000	100
37	C	FGZ	Glottal zero freq. (Hz)	0	5000	1500
38	C	BWGZ	Glottal zero bandwidth (Hz)	100	9000	6000
39	C	BWGS	Glottal res. 2 bandwidth (Hz)	100	1000	200

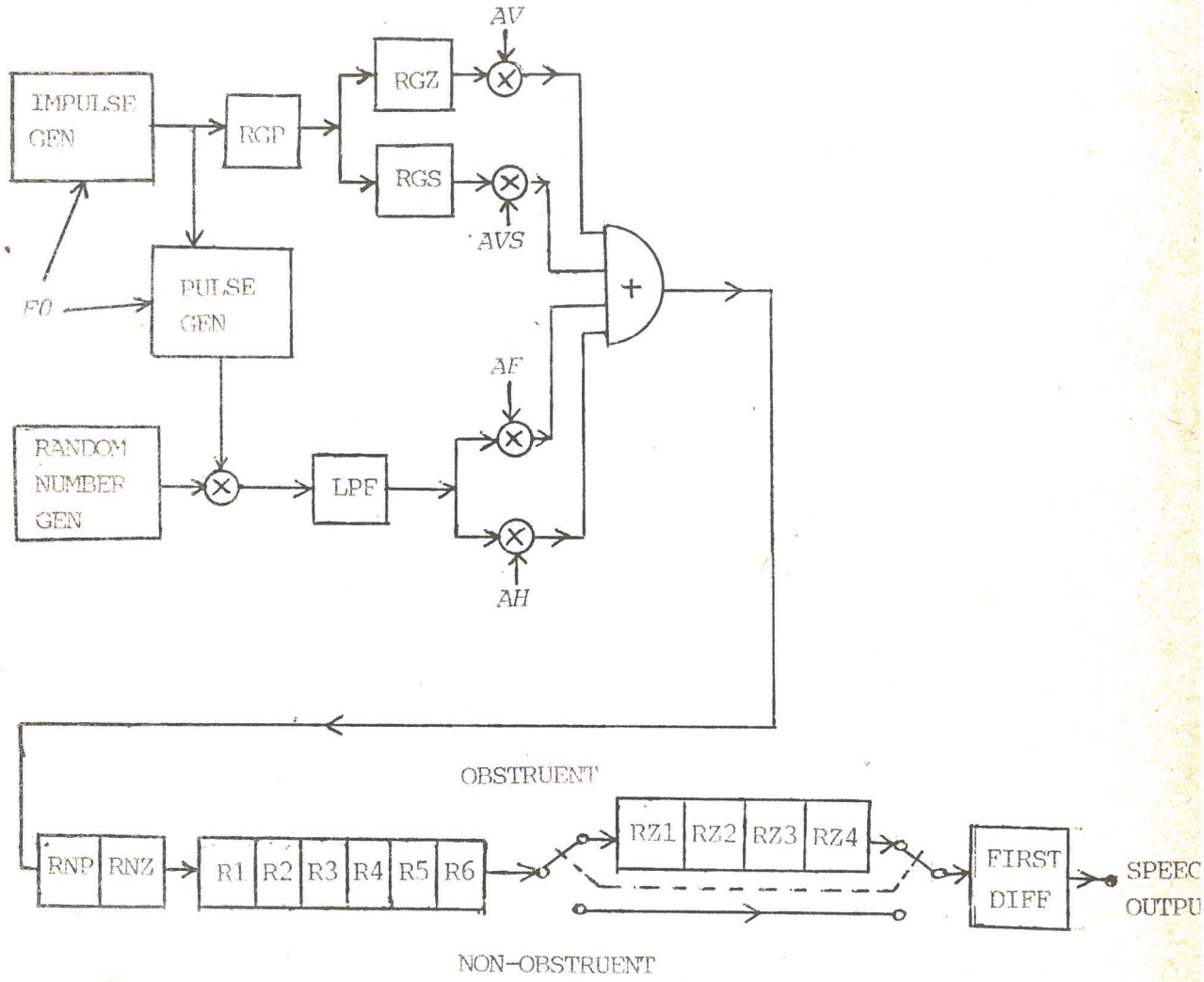


Fig. 4.1. The cascade pole-zero synthesizer.

CHAPTER 5

SPEECH SYNTHESIZER PROGRAM

5.1 INTRODUCTION

A program, SPSYNTH, was written in PASCAL, implementing both the all-pole cascade/parallel, and pole-zero cascade implementations, as described in the two preceding chapters. It runs on an IBM PC/ XT/AT or compatible machine. For listening to the speech output, a D/A converter card is needed.

The operations carried out by the program can be divided into two parts: generation of the parameter tracks, and secondly, synthesis and storing of the speech utterance. The program first asks the user to specify the type of synthesizer he wants to implement; i.e., the all-pole cascade/parallel type, or the pole-zero cascade type. The next screen presents a list of choices which have been put under two main headings: (*i*) Parameter Track Generation, and (*ii*) Synthesis.

The parameter tracks can be generated in one of three ways: starting from a default setting of the synthesizer, reading from an existing parameter track file, or, by using an in-built phoneme set. It is also possible to augment the phoneme set by specifying the parameter values for user defined phonemes in a specified format in a disk file.

Under the start synthesis option there are two alternatives: if

parameter track generation option has already been used, it is possible to use the generated tracks for synthesis once, otherwise, it is possible to read parameter tracks from a file and start the synthesis.

After the completion of every operation, the control is returned to the second screen. Also if the type of synthesizer is to be changed, it is possible to go back to the first screen.

The program can be terminated (Exit to DOS) when in the first and second screens.

The description of the program will be provided in the following sections: setting up the synthesizer, parameter track generation, and synthesis.

5.2 SETTING UP THE SYNTHESIZER

The synthesizer can be set up in two ways. It is possible to start from a default configuration, or, to read from an existing file. When generating the parameter tracks using the in-built phoneme set, always the default setting is used.

The list of parameters is presented in a tabular fashion on the screen. The parameters are presented in three groups. The display includes the parameter names, their minimum, maximum and default values, and, variable/constant indication. When in the setting up mode the V/C (variable/constant) and default value entries of the parameters can be changed. These two entries are highlighted.

The four curser keys can be used to move the curser to the appropriate parameter entry. The V/C entry and default values can then be changed. If the default value of a parameter is outside the allowed range then a beep is emitted. The four parameters, namely, number of formants, parameter update interval, sampling rate, and, overall gain should be kept constant during synthesis. Hence the program does not allow them to be made variables.

After the setting up operation is over it can be indicated by pressing the function key F10. The program responds by asking whether the user wishes to make more modifications. If so he can go back to the setting up mode. Otherwise the user is prompted for the duration of utterance. But if the initial configuration has been read from a file then the duration would already have been read from the same file. The duration of utterance can be up to three seconds and should not be less than the parameter update interval.

5.3 PARAMETER TRACK GENERATION

The parameter tracks can be generated in two ways. When either the "Start with default configuration" or "Modify existing tracks" option is selected from the second screen, the parameter tracks can be specified graphically. If "Use the in-built phoneme set" option has been selected, then the program asks for individual phonemes, and their starting and

ending points in milliseconds. The program then inserts proper values in the parameter tracks. When using this option it is also possible to augment the phoneme set by specifying user defined phonemes.

5.3.1 Graphical Generation of Parameter Tracks

While specifying the parameter tracks graphically, the horizontal axis represents time in milliseconds and the vertical axis represents the parameter chosen in proper units. The value of the parameter at any given point in time can be specified. It is also possible to specify two points and interpolate the parameter values between those two points.

The curser keys are used for moving around the "+" sign that appears on the screen to the desired location that represents the proper time-value combination. If desired the previous store point/join line segment operation can be cancelled.

All the store/erase operations are performed using the function keys. Parameter tracks for the next variable can be accessed by pressing the "PgDn" key. If corrections are to be made to the parameter tracks for the previous variable, the "PgDn" key should be pressed.

An exit from the graphical editing mode is provided through the function key F10. But if the parameter tracks for all variables have not been accessed at least once then the program automatically presents the untouched tracks. This safety precaution is particularly useful when the "Start from default configuration" option is selected from the options

screen.

5.3.2 Using the In-built Phoneme Set

If this option is selected, the program first asks for the number of phonemes to be generated. Then one by one the program prompts for the phonemes and their starting and ending points in milliseconds. The values input are checked so that they fall in the correct range. The program inserts predefined values into the parameter tracks for all the parameters during the time period in which the phonemes are to be generated. In between two phonemes the program generates the parameter tracks by linear interpolation.

It is possible to augment the phoneme set by specifying user defined phonemes. The parameter values for all the control parameters should be supplied in a specified format through a disk file. Thus it is possible to generate speech of any accent or physiological disorder by supplying proper parameter values to the program. This would be very useful in speech science studies.

5.4 SYNTHESIS

The start synthesis heading offers two options: use of the parameter tracks already in computer memory and reading the parameter tracks from a file. If parameter tracks have already been generated, they can be used once to generate the speech sequence. The parameter

tracks can also be read from disk. The "Start Synthesis" option turns control over to the appropriate synthesis routines depending on the synthesizer type selected by the user.

There are two synthesizer routines: one for the all-pole cascade-parallel and the other for the pole-zero cascade synthesizer. Both these routines have their own routines for generating filter coefficients. The filter coefficients are calculated by using the appropriate control parameter values after every parameter update interval. After the synthesis is finished the speech file can be stored on disk. The speech output can be listened to using a D/A converter. Testing of the program, along with sample parameter tracks, will be discussed in the next chapter.

CHAPTER 6

TESTING OF THE SPEECH SYNTHESIZER

6.1 INTRODUCTION

The synthesizer program, SPSYNTH, written in PASCAL, is available as the compiled executable file SPSYNTH.EXE and it runs on an IBM PC/XT/AT or compatible machine. Another program, ZERO_LOC, also written in PASCAL, may be used for finding anti-resonance zero-locations from the corresponding formant frequency and amplitude parameters for the Klatt synthesizer.

The program can be used for speech synthesis in accordance with the method given in the preceding chapter. The speech samples generated by the program are stored in a sequential file as 12-bit integer values (range: -2048 to 2047).

The speech data generated can be output as an analog signal by making use of a D/A converter. In our test setup, we used a D/A converter which is part of MCR-4 data acquisition card from Microcreation. This card fits in one of the PC slots and has the following options: a 12-bit A/D converter with 32 multiplexed inputs, a 12-bit D/A çonverter, 8-bit digital output, 8-bit digital input, and on-board 8253 counter for DMA timing control of A/D AND D/A converters. The card has both unipolar (0 to +5V), and, bipolar (-5 to +5V) outputs.

A driver for this card, DAOUT was written in PASCAL, listing for which can be found in Appendix A. To be able to listen to the speech output properly, an audio power amplifier was built, details of which may be found in Appendix B.

Both the all-pole cascade/parallel and pole-zero cascade synthesizers were tested. Isolated vowels, simple V-C-V sequences, and sentences up to 3 seconds long were synthesized. The synthesizer program runs slowly; it needs about 300 seconds to generate 1 second of speech. But for the applications for which this program is intended, the speed of synthesis is not a major problem.

This chapter will deal with the preparation of parameter tracks, some sample parameter tracks for isolated vowels and simple V-C-V sequences, and the method for generating parameter tracks for synthesis of longer speech sequences using the in-built phoneme set.

6.2 PREPARATION OF PARAMETER TRACKS

The parameter tracks were prepared using both the graphical method, and by using the in-built phoneme set, as outlined earlier in Section 5.3. The in-built phoneme set is the easier option when the sequence duration is long.

6.2.1 Isolated Vowels

The parameter tracks for this were specified using the graphical method. When synthesizing vowels, the variable parameters in both the synthesizer implementations are the same. Vowels /i/, /I/, /e/, /E/, /ɔ/, /ae/, /u/, /U/, /^/, etc with pronunciation keys in Table 6.1, were synthesized. The first three formants and bandwidths were given the appropriate values (Klatt, 1980; O'Shaughnessy, 1987). The last three formants and bandwidths were assigned the default values. Table 6.2 shows the first three formant frequencies and bandwidths for these vowels. During synthesis, the pitch was maintained constant, at 100 Hz. The amplitude of voicing, *AV*, was linearly raised from 0 dB to 60 dB over the initial 100 ms segment. It was maintained constant, and then linearly reduced from 60 dB to 0 dB over the last 100 ms of the utterance, as shown in Fig. 6.1.

6.2.2 V-C-V Sequences

To specify parameter tracks for V-C-V sequences, mostly the in-built parameter set was used. Tests were carried out with many V-C-V sequences, which included the sequences like */apa/*, */aba/*, */ata/*, */asa/*, */aka/*, */aga/*, etc. Vowel-semivowel-vowel sequences like */awa/*, */aya/*, */ala/*, and */ara/* were also generated. During the synthesis of these segments the formant values were varied from those for the vowels to the target values for the middle phoneme of the sequence, and then gradually

changed to those for the vowel once again.

For the pole-zero cascade synthesizer, the antiresonance zero frequencies and their bandwidths were first calculated by the program *ZERO_LOC*. Table 6.3 shows the formant amplitudes and first three formant bandwidths for some fricatives, stops, and affricates. Table 6.4 shows the zero locations for these phonemes. Table 6.5 shows the formant data for the semi-vowels /y/, /w/, /l/, and /r/, and the nasals /m/, and /n/.

Sample parameter tracks for a few sequences are listed in Tables 6.6, and 6.7. Table 6.6 lists the parameter tracks for the all-pole cascade/parallel synthesizer, and Table 6.7 lists those for the pole-zero cascade synthesizer.

6.2.3 Longer Speech Sequences

Synthesis of longer speech sequences was carried out using parameter tracks of the built-in phoneme set. Using this set, it was possible to generate sequences involving as many as 15 phonemes and up to three seconds in duration. It is possible to specify the gap between two adjacent phonemes. Hence after a few trials, satisfactory performance can be obtained. Both the synthesizers were tested, by generating sequences like isolated words, and small sentences.

The in-built phoneme set can be augmented as pointed out earlier, by specifying in a file values for various parameters which are normally varied. These values have to be in a specific format. The first line

should contain the number of phonemes defined in the file and the type of synthesizer; i.e., whether all-pole cascade/parallel or pole-zero cascade. Thereafter the entry for each new phoneme should indicate the symbol to be used for the phoneme in one line. On the subsequent lines, values for variable parameters starting with parameter number 3 (*AV*) up to parameter number 23 (*FNP*) should be specified. The parameters have been deliberately numbered so that in the case of the pole-zero cascade synthesizer the four zeros and their bandwidths replace the *A1-A6*, and *AB*, *AN* entries of the all-pole cascade/parallel synthesizer. Each line should contain values of exactly five parameters, so that it would take five lines to specify 23 parameters.

6.3 INFORMAL LISTENING TESTS

The synthesizer was tested by listening to synthesized speech segments, which consisted of isolated vowels, vowel-consonant-vowel and vowel-semivowel-vowel sequences, and longer speech segments up to three seconds in duration, as has already been described earlier in this chapter. The synthesizer is definitely well suited for the synthesis of longer speech segments, as the parameter tracks can be specified easily. It was found that voiced fricatives and stops were more clear than their unvoiced counterparts. Also the fricatives and stops synthesized with the pole-zero cascade synthesizer were sharper than those synthesized using the all-pole cascade/parallel synthesizer.

Table 6.1. Pronounciation keys for some vowels. Adapted from O'Shaughnessy (1987), Table 3.1.

i	beat	æ	bat	U	book	^	but
I	bit	ɑ	cot	u	boot	ɜ	curt
e	bait	ɔ	caught			ə	about
ɛ	bet	o	coat				

Table 6.2. First three formant frequencies and bandwidths for some vowels. Adapted from Klatt (1980), Table II; O'Shaughnessy (1987), Table 3.2.

Vowel	*F1*	*F2*	*F3*	*BW1*	*BW2*	*BW3*
i	270	2290	3010	60	200	400
I	390	1990	2550	50	100	140
e	330	2020	2600	55	100	200
ɛ	530	1840	2480	60	90	200
æ	660	1720	2410	70	150	320
ɑ	700	1220	2600	130	70	160
ɔ	570	840	2410	90	100	80
U	440	1020	2240	80	100	80
u	300	870	2240	60	80	60
^	640	1190	2390	80	50	140
ə	490	1350	1690	100	60	110

Table 6.3. First three formant frequencies and bandwidths, and formant amplitudes for some fricatives, stops, and affricaties before front vowels. Source: Klatt (1980), Table III.

Consonant	*F1*	*F2*	*F3*	*BW1*	*BW2*	*BW3*	*A2*	*A3*	*A4*	*A5*	*A6*	*AB*
ʃ	300	1840	2750	200	100	300	0	0	0	0	28	48
s	320	1390	2530	200	80	200	0	0	0	0	52	0
z	240	1390	2530	70	60	180	0	0	0	0	52	0
θ	320	1290	2540	200	90	200	0	0	0	0	28	48
ð	270	1290	2540	60	80	170	0	0	0	0	28	48
f	340	1100	2080	200	120	150	0	0	0	0	0	57
v	220	1100	2080	60	90	120	0	0	0	0	0	57
k	300	1990	2850	250	160	330	0	53	43	45	45	0
g	200	1990	2850	60	150	280	0	53	43	45	45	0
t	400	1600	2600	300	120	250	0	30	45	57	63	0
d	200	1600	2600	60	100	170	0	47	60	62	60	0
p	400	1100	2150	300	150	220	0	0	0	0	0	63
b	200	1100	2150	60	110	130	0	0	0	0	0	63
tʃ	350	1800	2820	200	90	300	0	44	60	53	53	0
dʒ	260	1800	2820	60	80	220	0	44	60	53	53	0

Table 6.4. Antiresonance frequencies and bandwidths for some stops, fricatives, and affricates.

Phoneme	*FZ1*	*FZ2*	*FZ3*	*BZ1*	*BZ2*	*BZ3*
k	3180	3673	4465	274	216	929
g	3180	3673	4464	260	214	927
t	2627	3363	3975	256	267	487
d	2693	3497	4232	185	249	755
ʃ	3153	3652	4473	267	218	929
tʃ	2882	3673	4420	295	215	896
dʒ	2882	3673	4420	269	215	896

Table 6.5. First three formant frequencies and bandwidths for the semi-vowels, and nasals. Source: Klatt (1980), Table II.

Phoneme	*F1*	*F2*	*F3*	*BW1*	*BW2*	*BW3*
y	260	2070	3020	40	250	500
w	290	610	2150	50	80	60
r	310	1060	1380	70	100	120
l	310	1050	2880	50	100	280
m	480	1270	2130	40	200	200
n	480	1340	2470	40	300	300

Table 6.6 Control parameters for some vowel-semivowel-vowel sequences synthesized with the all-pole cascade/parallel model. The sequence duration is 500 ms. All the other parameters are given default values as given in Table 3.1.

Syllable	Time	*AV*	*F0*	*F1*	*F2*	*F3*	*BW1*	*BW2*	*BW3*
	0	0	100	700	1220	2600	130	70	160
	100	60	.	.	.	.	.	.	.
	195	.	.	700	1220	2600	130	70	160
aya	245	.	.	260	2070	3020	40	250	500
	255	.	.	260	2070	3020	40	250	500
	305	.	.	700	1220	2600	130	70	160
	395	60	.	.	.	.	.	.	.
	495	0	100	700	1220	2600	130	70	160
	0	0	100	700	1220	2600	130	70	160
	100	60	.	.	.	.	.	.	.
	195	.	.	700	1220	2600	130	70	160
awa	245	.	.	290	610	2050	50	80	60
	255	.	.	290	610	2050	50	80	60
	305	.	.	700	1220	2600	130	70	160
	395	60	.	.	.	.	.	.	.
	495	0	100	700	1220	2600	130	70	160
	0	0	100	700	1220	2600	130	70	160
	100	60	.	.	.	.	.	.	.
	230	60	.	700	1220	2600	130	70	160
ara	240	50	.	310	1060	1380	70	100	120
	270	50	.	310	1060	1380	70	100	120
	305	60	.	700	1220	2600	130	70	160
	395	60	.	.	.	.	.	.	.
	495	0	100	700	1220	2600	130	70	120
	0	0	100	700	1220	2600	130	70	120
	100	.	.	.	.	.	.	.	.
	195	60	.	700	1220	2600	130	70	120
ala	210	50	.	310	1050	2880	50	100	280
	290	50	.	310	1050	2880	50	100	280
	305	60	.	700	1220	2600	130	70	120
	395	.	.	.	.	.	.	.	.
	495	0	100	700	1220	2600	130	70	120

Table 6.7. control parameters for some V-C-V sequences synthesized with the pole-zero cascade synthesizer. The duration of utterence is 500 ms. Only values for *NF*, zero frequencies and bandwidths, *F0*, *AV*, *AF*, and *AVS* are shown. The values for *F1-F3*, and *BW1-BW3* are as given in table 6.3. All the other parameters are given default values as given in Table 4.1. The values for *NF*, the number of zeros, are changed abruptly, and not interpolated linearly. The zero frequency values are used only during the production of stop/affricate/fricative sounds. Hence an 'X' (do not care) sign is shown in place of *NF*, and zero frequency and bandwidth values for vowels.

Syllable	Time	*F0*	*AV*	*AVS*	*AF*	*NZ*	*FZ1*	*FZ2*	*FZ3*	*BZ1*	*BZ2*	*BZ3*
	0	100	0	0	0	X	X	X	X	X	X	X
	100	.	60	.	.	X	X	X	X	X	X	X
	195	100	60	.	0	X	X	X	X	X	X	X
aka	220	0	0	.	60	3	3180	3673	4465	274	216	929
	280	0	0	.	60	3	3180	3673	4465	274	216	929
	330	100	60	.	0	X	X	X	X	X	X	X
	395	.	60	.	.	X	X	X	X	X	X	X
	495	100	0	0	0	X	X	X	X	X	X	X
	0	100	0	0	0	X	X	X	X	X	X	X
	100	.	60	.	.	X	X	X	X	X	X	X
	195	.	60	0	0	X	X	X	X	X	X	X
aga	220	.	47	47	60	3	3180	3673	4464	260	214	927
	280	.	47	47	60	3	3180	3673	4464	260	214	927
	330	.	60	0	0	X	X	X	X	X	X	X
	395	.	60	.	.	X	X	X	X	X	X	X
	495	100	0	0	0	X	X	X	X	X	X	X
	0	100	0	0	0	X	X	X	X	X	X	X
	100	.	60	.	.	X	X	X	X	X	X	X
	195	100	60	.	0	X	X	X	X	X	X	X
asha	220	0	0	.	60	3	3153	3652	4473	267	218	928
(ɑʃɑ)	280	0	0	.	60	3	3153	3652	4474	267	218	928
	330	100	60	.	0	X	X	X	X	X	X	X
	395	.	60	.	.	X	X	X	X	X	X	X
	495	100	0	0	0	X	X	X	X	X	X	X

(Continued on next page)

Table 6.7. (Continued from the previous page)

	0	100	0	0	0	X	X	X	X		X	X
	100	.	60	.	.	X	X	X	X		X	X
	195	100	60	.	0	X	X	X	X		X	X
ata	220	0	0	.	60	3	2627	3363	3975	256	267	487
	280	0	0	.	60	3	2627	3363	3975	256	267	487
	330	100	60	.	0	X	X	X	X		X	X
	395	.	60	.	.	X	X	X	X		X	X
	495	100	0	0	0	X	X	X	X		X	X

	0	100	0	0	0	X	X	X	X	X	X	X
	100	.	60	.	.	X	X	X	X	X	X	X
	195	.	60	0	0	X	X	X	X	X	X	X
ada	220	.	47	47	60	3	2693	3497	4232	185	249	755
	280	.	47	47	60	3	2693	3497	4232	185	249	755
	330	.	60	0	0	X	X	X	X	X	X	X
	395	.	60	.	.	X	X	X	X	X	X	X
	495	100	0	0	0	X	X	X	X	X	X	X

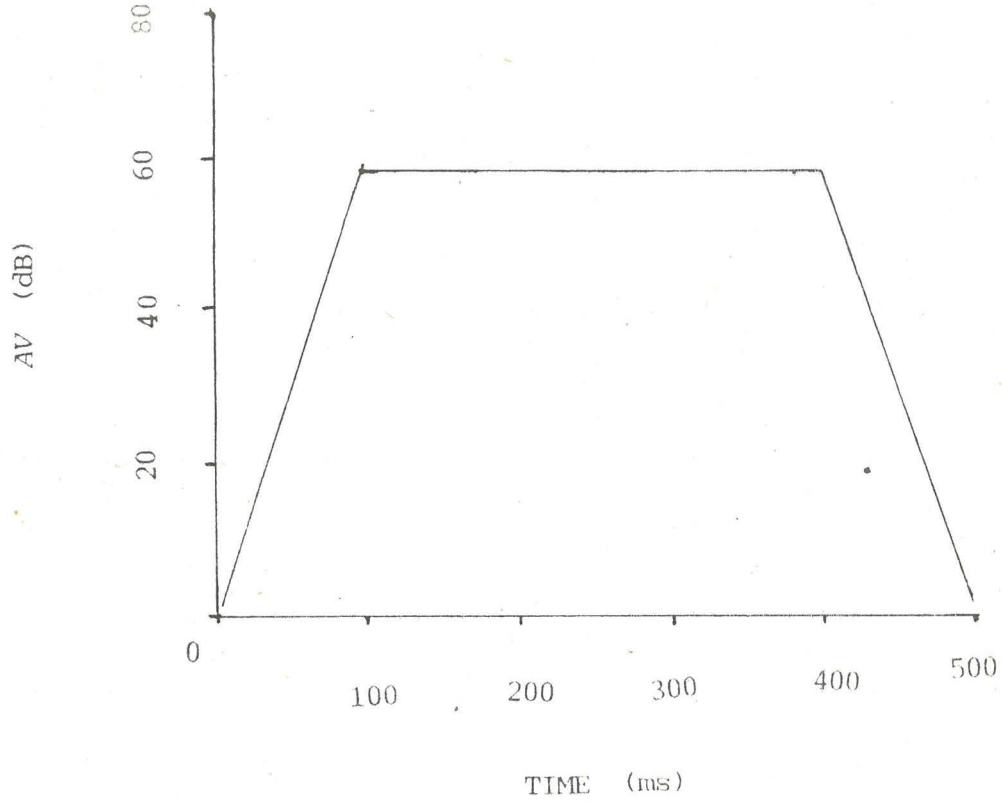


Fig. 6.1. Variation of the amplitude of voicing, *AV*, during the synthesis of isolated vowels.

CHAPTER 7

SUMMARY AND CONCLUSIONS

This thesis has described the development of a cascade pole-zero synthesis scheme. This scheme yields a natural representation for the vocal tract, and particularly so for the Indian languages, where aspiration is a major phonemic contrast feature. Especially, in the case of aspirated stops and affricates, as the source of aspiration is situated at the glottis, the uniform use of a cascade model yields better speech quality.

In this work we have tested the cascade pole-zero synthesis scheme by finding out the zero locations from the formant amplitude data given by Klatt. The program has a set of in-built phonemes, which make the preparation of tracks for longer speech segments easier. This set can be appended by specifying user defined phonemes. Hence it is possible to synthesize speech of any accent, or pathological disorder by specifying appropriate values for the control parameters

We have conducted informal listening tests with the program. The program performance can be improved as described below.

The program could be made more user-friendly. The program could be tested by carrying out extensive listening tests. The synthesizer can also be fine tuned by providing accurate values of different parameters,

obtained through speech analysis. The synthesizer would be more versatile if, while using the phoneme set option, one could also independently specify parameter trajectories for some of the parameters, especially, *F0*, *AV*, etc. This would enable the user to specify intonation, and stress patterns for synthesized segments.

APPENDIX A

PROGRAM LISTINGS

A.1 The SPSYNTH Program

A.2 The ZERO_LOC Program

A.3 The DAOUT Program

The SPSYNTH Program

```
(* Written by Shailendra A. Chafekar; 1989-90  *)

unit glbvars;

(* This unit declares the global variables *)

  interface

  const
    pars=39;
    maxdur=2000;
  type
    strg=string[80];
    arry=array[0..500] of integer;
    trac=array[0..20000] of integer;
    speech=array[0..20000] of integer;
  var
    partrc:^trac;
    wav:^speech;
    ary:arry;

    i,j,k,p,m,n,cons,vars,vari,posn,update,pend,number,code,uptime,dur,
    totup,samprate,samperup,samperpitch,modcount,pitchcount:integer;

    found,over,accept,finish,extfile,endedit,unchg,
    initcoeff,zero,start,pitchpd,
    synthtype,options,quit,cas_par,cas,gentrc,phnset,ramtracks,
    st_synth,useram:boolean;

    key,dig:char;
    num,pt:string[5];
    resp:string[1];
    msg,filename:strg;

    nam:array [1..pars] of string[4];
    conf:array[1..pars] of char;
    min,max,def,par,nvar,lvar:array[1..pars] of integer;

    track,wavefile:text;


    f0,av,af,avs,ah,f1,f2,f3,f4,f5,f6,bw1,bw2,bw3,bw4,bw5,bw6,
    fnp,fnz,bwnp,bwnz,fgp,fgz,fgs,bwgp,bwgz,bwgs,g0,nf:integer;

    sampint,a1,b1,c1,a2,b2,c2,a3,b3,c3,a4,b4,c4,a5,b5,c5,a6,b6,c6,
    anp,bnp,cnp,anz,bnz,cnz,agp,bgp,cgp,agz,bgz,cgz,ags,bgs,cgs:real;

    implse,sinamp,asp,fric,afprev,plrel,plstep:real;

    y1c,y11c,y12c,y2c,y21c,y22c,y3c,y31c,y32c,y4c,y41c,y42c,
    y5c,y51c,y52c,y6c,y61c,y62c,ynpc,ynp1c,ynp2c,ynzc,ynz1c,ynz2c,
```

```
    ygp,ygp1,ygp2,ygz,ygz1,ygz2,ygs,ygs1,ygs2,ygs3,ygs4,
    glotvelAL,glotvelL,aasp,afric,dasp,dfric,glotvel,noise,fricvel,
    glotvel1,vel:real;
    ynp,ynp1,ynp2,inp,insin,lipvel,glotvelA,aspvel,mx:real;
    voiced,aspi,frc,vfrc:boolean;

  implementation

  end.          (* unit glbvars *)
```

```
(*  Written by Shailendra A. Chafekar; 1989-90  *)

unit basicfns;

(* This unit contains some commonly used procedures *)

  interface

  uses
    glbvars;

  var
    trc:array[1..40] of integer;
    header,choice1,choice2:strg;

  procedure findpos;
  procedure error;
  procedure trcread;
  procedure trcwrite;
  procedure readcons;
  procedure readvars;
  procedure getresp(ans:strg;ch1,ch2,ch3,ch4:char);
  procedure filcoef(f,bw:real;var a,b,c:real);
  function linamp(amp:real):real;
  procedure pres_type;
  procedure type_of_synth;
  procedure pres_op;
  procedure operation;
  procedure confset;
  procedure preparetrack;


  implementation

  uses
    crt,edit,graph;

  procedure findpos;
    begin
      vars:=0;
      p:=0;
      for i:=1 to pars do
        begin
        if conf[i]='V' then
          begin
            vars:=vars+1;
            p:=p+1;
            nvar[i]:=p;
            lvar[p]:=i;
          end;
        end;
```

```
  end;            {* procedure findpos *}

procedure error;
  begin
    sound(500);
    delay(50);
    nosound;

  end;            {* procedure error *}

procedure trcread;
  var
    y,z:integer;
  begin
    clrscr;
    gotoxy(10,4);
    write('Parameter tracks to be read from disk.');
    gotoxy(10,6);
    write('Enter file name : ');
    readln(filename);
    assign(track,filename);
    gotoxy(1,24);
    {$I-}
    reset(track);
    {$I+}
    if ioresult=0 then
      begin
        for i:=1 to length(filename) do filename[i]:=upcase(filename[i]);
        write('Reading from ',filename,' ...');
        i:=1;
        for z:=1 to 13 do
          begin
            for y:=1 to 3 do
              begin
                read(track,pt);
                read(track,conf[i]);
                if y=3 then
                  readln(track,def[i])
                else
                  begin
                    read(track,def[i]);
                    read(track,dig);
                  end;
                i:=i+13;
              end;
            i:=i-38;
          end;
        readln(track,msg);
        readln(track,vars,uptime,dur);
        readln(track,msg);
```

```
        totup:=dur div uptime;
        findpos;
        for j:=0 to round(totup-1) do
          begin
            for i:=1 to pars do
             begin
               posn:=(i-1)+(j*pars);
               partrc^[posn]:=min[i];
             end;
          end;
        for j:=0 to (totup-1) do
          begin
            read(track,m);
            for i:=1 to vars do
              begin
                vari:=lvar[i];
                posn:=(vari-1)+(j*pars);
                read(track,partrc^[posn]);
              end;
            readln(track,dig);
          end;
        close(track);
      end
    else
      begin
        found:=false;
        write('File not found ...');
        delay(250);
        error;
      end;
    delline;

  end;          {*  procedure trcread *}

procedure trcwrite;
  var
    y,z:integer;
    trcwr:boolean;
    varlst:string[132];
  begin
    gotoxy(10,8);
    write('Enter file name : ');
    readln(filename);
    assign(track,filename);
    for i:=1 to length(filename) do
      filename[i]:=upcase(filename[i]);
    gotoxy(1,24);
    write('Saving parameter tracks in ',filename,' ...');
    rewrite(track);
    i:=1;
    for z:=1 to 13 do
```

```
begin
  for y:=1 to 3 do
    begin
      str(def[i]:5,msg);
      pt:=nam[i];
      k:=4-length(pt);
      if k=1 then
        pt:=concat(pt,' ')
      else if k=2 then
        pt:=concat(pt,'  ');
      msg:=concat(pt,' ',conf[i],' ',msg,' ');
      if y=3 then
        writeln(track,msg)
      else write(track,msg);
      i:=i+13;
    end;
    i:=i-38;
  end;
write(track,'Variables/');
write(track,'Param updated at (ms)/');
writeln(track,'Duration');
writeln(track,vars:2,'        ',uptime:2,'                    ',dur:4);
varlst:='Time';
for vari:=1 to pars do
  begin
    if conf[vari]='V' then
      begin
        pt:=nam[vari];
        k:=4-length(pt);
        if k=1 then
          pt:=concat('  ',pt)
        else if k=2 then
          pt:=concat('   ',pt);
        varlst:=concat(varlst,pt);
      end;
  end;
writeln(track,varlst);
for j:=0 to (totup-1) do
  begin
    trcwr:=false;
    str(j*uptime:4,msg);
    write(track,msg);
    msg:='';
    for vari:=1 to pars do
      begin
        if conf[vari]='V' then
          begin
            posn:=(vari-1)+(j*pars);
            write(track,partrc^[posn]:6);
            trcwr:=true;
          end;
```

```
        end;
        dig:='/';
        if trcwr then
          writeln(track,' /');
      end;
      close(track);
      delline;

  end;          {*  procedure trcwrite *}

procedure readcons;
  begin
    for i:= 1 to pars do
      begin
        if conf[i]='C'then
          par[i]:=def[i];
      end;

  end;          {*  procedure readcons *}

procedure readvars;
  begin
    for i:= 1 to pars do
      begin
        if conf[i]='V' then
          begin
            posn:=(update * pars) + (i-1);
            par[i]:=partrc^[posn];
          end;
      end;

  end;          {*  procedure readvars *}



procedure getresp(ans:strg;ch1,ch2,ch3,ch4:char);
  begin
    m:=wherex;
    j:=wherey;
    accept:=false;
    over:=false;
    while not over do
      begin
        dig:=readkey;

        if dig<>#0 then
          begin
            if ans='choice' then
              begin
                if (dig=ch1) or (dig=ch2) or (dig=ch3) or (dig=ch4) then
                  begin
```

```
            if accept then
              error;
            if length(resp)<1 then
              begin
                write(dig);
                dig:=upcase(dig);
                resp:=concat(resp,dig);
                accept:=true;
              end;
          end
        else if (length(resp)<1) or (ord(dig)>32) then
          error;
      end;
    if ans='number' then
      begin
        if (ord(dig) >= 48) and (ord(dig) <= 57) then
          begin
          num:=concat(num,dig);
          write(dig);
          end;
      end;
    if dig=#8 then
      begin
        if wherex>m then
          begin
            i:=wherex-1;
            window(i,j,i,j);
            clrscr;
            window(1,1,80,25);
            gotoxy(i,j);
            if ans='choice' then
              begin
                delete(resp,length(resp),1);
                accept:=false;
              end;
            if ans='number' then
              delete(num,length(num),1);
          end;
      end;
    if dig=#13 then
      begin
        over:=true;
        if (ans='choice') and (resp='') then
          over:=false;
        if ans='number' then
          val(num,number,code);
      end;
  end
else dig:=readkey;
end;
```

```
  end;                {*  procedure getresp *}


procedure filcoef(f,bw:real;var a,b,c:real);
  var
    comn:real;
  begin
    comn:=-pi*bw*sampint;
    comn:=exp(comn);
    c:=-comn*comn;
    b:=2*comn*cos(2*pi*f*sampint);
    a:=1-b-c;
    if zero then
      begin
        a:=1.0/a;
        b:=-b*a;
        c:=-c*a;
      end;

  end;          {*  procedure filcoef *}


function linamp(amp:real):real;
  var
    amp1,amp2,s,d:real;
    stab:array[1..28] of real;
    dtab:array[1..11] of real;
  begin
    stab[1]:=65536.0;
    for i:=2 to 28 do
      stab[i]:=stab[i-1]/2;
    dtab[1]:=1.8;dtab[2]:=1.6;dtab[3]:=1.43;dtab[4]:=1.26;
    dtab[5]:=1.12;dtab[6]:=1.0;dtab[7]:=0.89;dtab[8]:=0.792;
    dtab[9]:=0.702;dtab[10]:=0.623;dtab[11]:=0.555;
    if amp>96 then
      amp:=96;
    linamp:=0.0;
    if amp>-72 then
      begin
        amp1:=amp / 6;
        amp2:=amp-(6*(trunc(amp1)));
        s:=stab[17-trunc(amp1)];
        d:=dtab[6-trunc(amp2)];
        linamp:=s*d;
      end;

  end;          {*  procedure linamp *}

procedure pres_type;
  begin
    clrscr;
```

```
    textbackground(white);
    textcolor(black);
    gotoxy(10,8);
    write(' 1 ');
    gotoxy(10,10);
    write(' 2 ');
    gotoxy(10,12);
    write(' Q ');
    textbackground(black);
    textcolor(white);
    gotoxy(7,4);
    write(header);
    gotoxy(13,8);
    write(choice1);
    gotoxy(13,10);
    write(choice2);
    gotoxy(13,12);
    write(' Exit to DOS');
    gotoxy(10,15);
    write('Enter your choice : ');

  end;              {*  procedure pres_type *}

procedure type_of_synth;
  begin
    synthtype:=false;
    cas_par:=false;
    cas:=false;
    options:=false;
    quit:=false;
    rawtracks:=false;
    header:='Select Synthesizer Type';
    choice1:=' Cascade/Parallel Synthesis';
    choice2:=' Cascade Pole-Zero Synthesis';
    pres_type;
    while not (options or quit) do
      begin
        key:=readkey;
        case key of
          '1','2' : begin
                      options:=true;
                      if key='1' then cas_par:=true;
                      if key='2' then cas:=
                      true;
                    end;
          'Q','q' : quit:=true;
           #0     : key:=readkey;
        end;
      end;

  end;              {*  procedure type_of_synth *}
```

```
procedure pres_op;
  begin
    clrscr;
    gotoxy(7,2);
    write('Synthesizer selected : ');
    if cas_par then
      write('Cascade - Parallel');
    if cas then
      write('Cascade Pole - Zero');
    gotoxy(7,4);
    write('Parameter track generation');
    gotoxy(13,6);
    write(' Start with in-built configuration');
    gotoxy(13,8);
    write(' Modify existing tracks');
    gotoxy(13,10);
    write(' Use in-built phoneme set');
    gotoxy(7,13);
    write('Start Synthesis');
    gotoxy(13,15);
    write(' Read parameter tracks from file');
    gotoxy(13,17);
    write(' Use the tracks present in RAM');
    gotoxy(10,20);
    write(' Previous screen');
    gotoxy(33,20);
    write(' Exit to DOS');
    textbackground(white);
    textcolor(black);
    j:=0;
    for i:=1 to 5 do
      begin
        if i<4 then
          gotoxy(10,5+i+j)
        else gotoxy(10,8+i+j);
        write(' ',i,' ');
        j:=j+1;
      end;
    gotoxy(7,20);
    write('Esc');
    gotoxy(30,20);
    write(' Q ');
    textbackground(black);
    textcolor(white);
    gotoxy(7,22);
    write('Enter your choice : ');

  end;              {*  procedure pres_op *}
```

```
procedure operation;
  begin
    pres_op;
    gentrc:=false;
    extfile:=false;
    phnset:=false;
    st_synth:=false;
    useram:=false;
    while not (gentrc or st_synth or synthtype or quit) do
      begin
        key:=readkey;
        case key of

          '1','2','3': begin
                         gentrc:=true;
                         if key='2' then
                           extfile:=true;
                         if key='3' then
                           phnset:=true;
                       end;
          '4','5'    : begin
                         st_synth:=true;
                         if key='4' then
                           extfile:=true;
                         if key='5' then
                           useram:=true;
                       end;
          'Q','q'    : quit:=true;
          #27        : synthtype:=true;
          #0         : key:=readkey;

        end;
      end;

  end;              {*  procedure operation *}

procedure confset;
  begin
    finish:=false;
    while not finish do
      begin
        menu;
        gotoxy(1,20);
        write('    End editing ?   <Y/N> : ');
        resp:='';
        getresp('choice','y','Y','n','N');
        if resp='Y'then
          finish:=true;
      end;
    if not extfile then
      begin
```

```
        accept:=false;
        gotoxy(1,22);
        write('    Duration of utterence in ms < ',def[24],'..',maxdur,' > : ');

        k:=wherex;
        while not accept do
          begin
            num:='';
            getresp('number','j','j','j','j');
            if (number>=def[24]) and (number<=maxdur) then
              accept:=true
            else
              begin
                window(k,22,80,22);
                clrscr;
                window(1,1,80,25);
                gotoxy(k,22);
                error;
              end;
          end;
        dur:=number;
      end;
    dur:=round(dur/def[24])*def[24];
    uptime:=def[24];
    totup:=dur div uptime;
    samperup:= def[24] * (def[25] div 1000);

  end;              {* procedure confset *}

procedure preparetrack;
  begin
    if vars >= 1 then
      begin
        if not extfile then
          begin
            for j:=0 to round(totup-1) do
              begin
                for i:=1 to vars do
                  begin
                    vari:=lvar[i];
                    posn:=(vari-1)+(j*pars);
                    partrc^[posn]:=min[vari];
                  end;
              end;
          end;
        if extfile then
          begin
            gotoxy(36,20);
            write('Change parameter tracks ?   <Y/N> : ');
            resp:='';
            getresp('choice','y','Y','n','N');
```

```
        end;
      if not((extfile) and (resp='N')) then
        begin
          endedit:=false;
          for i:=1 to vars do trc[i]:=0;
          k:=1;
          while not endedit do
            begin
              trc[k]:=1;
              vari:=lvar[k];
              for j:=0 to round(totup)-1 do
                begin
                  posn:=(vari-1)+(j*pars);
                  ary[j]:=partrc^[posn];
                end;
              p:=vari;
              gredit;
              for j:= 0 to round(totup)-1 do
              partrc^[(vari-1)+(j*pars)]:=ary[j];
              sound(500);
              delay(125);
              nosound;
              if endedit=true then
                begin
                  i:=1;
                  unchg:=false;
                  while (not unchg) and (i<=vars) do
                    begin
                      if trc[i]=0 then
                        begin
                          unchg:=true;
                          endedit:=false;
                          k:=i;
                          sound(500);
                          delay(200);
                          nosound;
                        end
                      else i:=i+1;
                    end;
                end;
            end;
        end;
    end;

  end;          {* procedure preparetrack *}

end.            (* unit basicfns *)
```

```
(* Written by Shailendra A. Chafekar; 1989-90 *)

unit edit;

  interface

  uses
    glbvars;

  procedure menu;          {* for setting up the synthesizer *}
  procedure gredit;        {* for graphical prep of parameter tracks *}


  implementation

  uses
    crt,graph;

  procedure menu;
    const
      grp1=1;
      grp2=27;
      grp3=53;
      fld1=1;
      fld2=6;
      fld3=9;
      fld4=14;
      fld5=20;
    var
      x,y,grp,fld,value,relx,pf,xp,yp:integer;
      key1,cf:char;
      val0,val1:string[5];
      df:string[1];

    procedure screen;
      begin
        writeln;
        textbackground(white);
        textcolor(black);
        writeln('Synthesizer Control Parameters');
        writeln('Format is Parameter, V/C, Min, Max & Default Values.');
        writeln;
        gotoxy(1,24);
        write(' ',chr(24),' ',chr(25),' ',chr(26),' ',chr(27),' : ');
        write('Curser Control   Back Sp, Del : Delete char   F10 : End edit
');
        textbackground(black);
        textcolor(white);
        if extfile then
          begin
            gotoxy(1,22);
```

```
        write('    Duration of utterence : ',dur,' ms');
      end;

  end;          { screen }

procedure startup;
  var
    y,z:integer;
  begin
    i:=1;
    j:=5;
    for z:=1 to 13 do
      begin
        k:=0;
        for y:=1 to 3 do
          begin
            textbackground(white);
            textcolor(black);
            gotoxy(k+fld2,j);
            write(conf[i]);
            gotoxy(k+fld5,j);
            write(def[i]);
            textbackground(black);
            textcolor(white);
            gotoxy(k+fld1,j);
            write(nam[i]);
            gotoxy(k+fld3,j);
            write(min[i]);
            gotoxy(k+fld4,j);
            write(max[i]);
            i:=i+13;
            k:=k+26;
          end;
        i:=i-38;
        j:=j+1;
      end;

  end;          {startup}

procedure chkval;
  begin
    if (pf<>p) or (fld=2) then
      begin
        val0:=val1;
        val(val0,value,code);
        accept:=true;

        if (value>=min[pf]) and (value<=max[pf]) then
          begin
            def[pf]:=value;
            str(def[p],val1);
```

```
          end
        else
          begin
            accept:=false;
            if key1=#68 then
              x:=(grp-1)*26+fld5
            else
               begin
                 x:=xp;
                 y:=yp;
               end;
            gotoxy(x,y);
            relx:=1;
            p:=pf;
            sound(500);
            delay(250);
            nosound;
          end;
      end;

  end;            {chkval}

procedure mov;
  begin
    x:=wherex;
    y:=wherey;
    xp:=(grp-1)*26+fld5;
    yp:=y;
    if key1=#77 then
      begin
        if x=6 then
          x:=20
        else if x=24 then
          x:=32
        else if x=32 then
          x:=46
        else if x=50 then
          x:=58
        else if x=58 then
          x:=72
        else x:=x+1;
      end;
    if key1=#75 then
      begin
        if x=72 then
          x:=58
        else if x=58 then
          x:=46
        else if x=46 then
          x:=32
        else if x=32 then
```

```
          x:=20
        else if x=20 then
          x:=6
        else x:=x-1;
      end;
    if key1=#72 then
      y:=y-1;
    if key1=#80 then
      y:=y+1;
    if x<6 then
      x:=72;
    if x>76 then
      x:=6;
    if y<5 then
      y:=17;
    if y>17 then
      y:=5;
    gotoxy(x,y);
    if (x>=1) and (x<grp2) then
      grp:=1;
    if (x>=grp2) and (x<grp3) then
      grp:=2;
    if (x>=grp3) and (x<78) then
      grp:=3;
    pf:=p;
    p:=((grp-1)*13)+(y-4);
    case (x-((grp-1)*26)) of

                    6 : fld:=2;
      20,21,22,23,24 : begin
                         fld:=5;
                         relx:=x-((grp-1)*26)-fld5+1;
                       end;
    end;
    chkval;

  end;          {mov}

procedure chconf;
  begin
    cf:=key;
    cf:=upcase(cf);
    if cas_par then
      i:=2;
    if cas then
      i:=1;
    if ((p>i) and (p<24)) or (p>26) then
      begin
        if (cf='V') or (cf='C') then
          begin
            textbackground(white);
```

```
            textcolor(black);
            write(cf);
            textbackground(black);
            textcolor(white);
            conf[p]:=cf;
            gotoxy(x,y);
          end;
      end
    else if cf='V' then
      begin
        sound(500);
        delay(50);
        nosound;
      end;

  end;          {chconf}

procedure defch;
  begin
    df:='';
    if ((ord(key) >=48) and (ord(key) <=57)) then
      begin
        df:=concat(df,key);
        delete(val1,relx,1);
        insert(df,val1,relx);
        relx:=relx+1;
        textbackground(white);
        textcolor(black);
        write(key);
        if relx=6 then
          begin
            relx:=5;
            gotoxy(x,y);
          end;
        textbackground(black);
        textcolor(white);
      end;

  end;          {defch}

procedure del;
  begin
    x:=wherex;
    y:=wherey;
    window(x,y,x,y);
    clrscr;
    window(1,1,80,25);
    gotoxy(x,y);
    if fld=5 then
      relx:=x-((grp-1)*26)-fld5+1;
    if relx=length(val1) then
```

```
      delete(val1,relx,1);

  end;            {del}

procedure bksp;
  begin
    x:=wherex-1;
    y:=wherey;
    relx:=x-((grp-1)*26)-fld5+1;
    window(x,y,x,y);
    clrscr;
    window(1,1,80,25);
    gotoxy(x,y);
    if relx=length(val1) then
      delete(val1,relx,1);

  end;            {bksp}

procedure exodus;
  begin
    pf:=p;
    p:=p+1;
    chkval;
    if accept then
      over:=true;

  end;            {exodus}

begin                 {* procedure menu *}
  clrscr;
  screen;
  startup;
  grp:=1;fld:=2;
  x:=fld2;y:=5;
  gotoxy(x,y);
  relx:=1;
  p:=1;
  pf:=1;
  str(def[p],val1);
  value:=def[1];
  over:=false;
  while not over do
    begin
      key:=readkey;
      if key=#0 then
        begin
          key1:=readkey;
          case ord(key1) of

            77,75,72,80 : mov;
                     83 : del;
```

```
                    68 : exodus;
            end;
          end;
        if key<>#0 then
          begin
            if fld=2 then
              chconf;
            if fld=5 then
              begin
                if key<>#8 then
                  defch
                else if (relx>1) then
                  bksp;
              end;
          end;
      end;

  end;          {* procedure menu *}


procedure gredit;
  var
  gd,gm,stx,sty,endx,endy,stpx,stpy,stpyp,stp,xpts,ypts:integer;
  x,incx,incyp,xp,yp,tim,value,oldtime,newtime,oldval,newval,incval:integer;
  y,incy,minystp,scx,scy:real;
  stg:string[15];
  axis:char;
  buf,flg:arry;
  bm:pointer;
  size:word;

  procedure plotaxis;
    var
      x,y:integer;
      done:boolean;
    begin
      x:=0;
      y:=0;
      line(stx,sty,endx+8,sty);
      outtextxy(endx-340,sty+30,'Time (ms)');
      line(stx,sty,stx,endy-7);
      for i:=0 to xpts do
        begin
          outtextxy(i*stp*incx+stx-2,sty,'T');
          str(i*stp*round(uptime),stg);
          settextjustify(1,2);
          outtextxy(i*stp*incx+stx,sty+10,stg);
          settextjustify(0,2);
        end;
      ypts:=trunc((max[p]-min[p])/stpyp);
      settextjustify(2,1);
```

```
    for i:=0 to ypts do
      begin
        outtextxy(stx,sty-i*incyp,'-');
        str(min[p]+i*stpyp,stg);
        outtextxy(stx-8,sty-i*incyp,stg);
      end;
    settextjustify(0,2);
    str(round(dur),stg);
    stg:=concat('Dur ',stg,' ms');
    outtextxy(endx-2,sty,'T');
    outtextxy(endx-100,sty+30,stg);
    settextjustify(2,1);
    str(max[p],stg);
    outtextxy(stx,endy,'-');
    outtextxy(stx-8,endy,stg);
    settextjustify(0,2);

  end;            {plotaxis}

procedure saveimg;
  begin
    size:=imagesize(xp-10,yp-12,xp+10,yp+12);
    getmem(bm,size);
    getimage(xp-10,yp-12,xp+10,yp+12,bm^);

  end;            {saveimage}

procedure plottrack;
  var
    xi,yi,xn,yn,i:integer;
  begin
    xi:=stx;yi:=sty;
    for i:=0 to (round(dur) div stpx)-1 do
      begin
        xn:=stx+i*round(stpx*scx);
        yn:=sty-round((ary[i]-min[p])*scy);
        line(xi,yi,xn,yn);
        xi:=xn;yi:=yn;
      end;

  end;            {plottrack}

procedure init;
  var
    i:integer;
  begin
    stpy:=stpyp;
    for i:=0 to round(dur/stpx)-1 do
      buf[i]:=ary[i];
    for i:=0 to round(dur/stpx)-1 do
      flg[i]:=0;
```

```
    x:=stx;
    y:=sty;
    tim:=0;
    value:=min[p];
    outtextxy(getmaxx-100,0,'T=0 ms');
    str(min[p],stg);
    stg:=concat(nam[p],'=',stg);
    outtextxy(getmaxx-100,10,stg);
    plottrack;
    xp:=x;yp:=round(y);
    saveimg;
    outtextxy(x+1,round(y)-6,'+');

  end;          {init}

procedure grscreen;
  begin
    detectgraph(gd,gm);
    initgraph(gd,gm,'');
    stpx:=round(uptime);
    stp:=trunc(dur/10);
    if stp>stpx then
      stp:=trunc(stp/stpx)
    else
      stp:=1;
    if stp=1 then
      xpts:=round(dur/stpx)
    else
      xpts:=trunc(dur/(stp*stpx));
    if frac((max[p]-min[p])/10)=0 then
      stpyp:=10;
    if frac((max[p]-min[p])/20)=0 then
      stpyp:=20;
    if frac((max[p]-min[p])/50)=0 then
      stpyp:=50;
    if frac((max[p]-min[p])/100)=0 then
      stpyp:=100;
    if frac((max[p]-min[p])/150)=0 then
      stpyp:=150;
    if frac((max[p]-min[p])/1000)=0 then
      stpyp:=1000;
    value:=min[p];
    stpy:=stpyp;
    stx:=60;
    sty:=getmaxy-70;
    endx:=getmaxx-40;
    endy:=60;
    scx:=(endx-stx)/dur;
    scy:=(sty-endy)/(max[p]-min[p]);
    incx:=round(scx*stpx);
    incy:=scy*stpy;
```

```
    incyp:=round(scy*stpyp);
    endx:=stx+incx*round(dur/stpx);
    endy:=sty-incyp*round((max[p]-min[p])/stpyp);
    plotaxis;
    init;
    msg:='F1-Store pt F3-Vert step size  F5-Erase F7-Store line ';
    outtextxy(60,sty+50,msg);
    msg:='Pg up-Prev track Pg dn-Next track F10-End edit';
    outtextxy(60,sty+60,msg);

  end;          {grscreen}

procedure readaxs;
  begin
    if x=stx then x:=x+4;
    outtextxy(x-3,round(y)-6,'+');
    x:=xp;
    str(tim,stg);
    outtextxy(getmaxx-100,0,concat('T=',stg,' ms'));
    str(value,stg);
    stg:=concat(nam[p],'=',stg);
    outtextxy(getmaxx-100,10,stg);

  end;          {readaxs}

procedure movcurser;
  var
    buf:arry;
    i,maxx:integer;
  begin
    maxx:=endx-incx;
    setviewport(getmaxx-110,0,getmaxx,20,true);
    clearviewport;
    setviewport(x,round(y),x+8,round(y)+12,true);
    clearviewport;
    setviewport(0,0,getmaxx,getmaxy,true);
    case ord(key) of

      72 : begin
             y:=y-incy;
             value:=value+stpy;
             if y<endy then
               y:=endy;
           end;
      80 : begin
             y:=y+incy;
             value:=value-stpy;
             if y>sty then
               y:=sty;
           end;
      77 : begin
```

```
            x:=x+incx;
            tim:=tim+stpx;
          end;
     75 : begin
            x:=x-incx;
            tim:=tim-stpx;
          end;
    116 : begin
            x:=x+10*incx;
            tim:=tim+10*stpx;
          end;
    115 : begin
            x:=x-10*incx;
            tim:=tim-10*stpx;
          end;
    117 : begin
            x:=x+20*incx;
            tim:=tim+20*stpx;
          end;
    119 : begin
            x:=x-20*incx;
            tim:=tim-20*stpx;
          end;

  end;
  if x>maxx then
    x:=maxx;
  if x<stx then
    x:=stx;

  if tim<0 then
    tim:=0;
  if tim>round(dur-uptime) then
    tim:=round(dur-uptime);
  if value<min[p] then
    value:=min[p];
  if value>max[p] then
    value:=max[p];
  putimage(xp-10,yp-12,bm^,0);
  freemem(bm,size);
  xp:=x;
  yp:=round(y);
  saveimg;
  readaxs;

end;          {movcurser}

procedure storept;
  var
    i:integer;
  begin
```

```
    for i:=0 to (round(dur) div stpx)-1 do
      buf[i]:=ary[i];
    flg[tim div stpx]:=1;
    ary[tim div stpx]:=value;
    oldtime:=tim;
    oldval:=value;
    setviewport(stx+1,endy-8,endx+6,sty,true);
    clearviewport;
    setviewport(0,0,getmaxx,getmaxy,true);
    plotaxis;
    plottrack;
    xp:=x;
    yp:=round(y);
    saveimg;
    readaxs;

  end;          {storept}

procedure erasept;
  var
    i:integer;
  begin
    flg[tim div stpx]:=0;
    for i:=0 to (round(dur) div stpx)-1 do
      ary[i]:=buf[i];
    setviewport(stx+1,endy-8,endx+4,sty,true);
    clearviewport;
    setviewport(0,0,getmaxx,getmaxy,true);
    plotaxis;
    plottrack;
    xp:=x;
    yp:=round(y);
    saveimg;
    readaxs;

  end;          {erasept}

procedure chstpy;
  var
    done:boolean;
  begin
    accept:=false;
    while not accept do
      begin
        str(max[p],msg);
        msg:=concat('< 1..',msg,' >');
        outtextxy(60,0,'New step for y-axis=');
        outtextxy(60,10,msg);
        num:='';
        m:=230;
        j:=0;
```

```
    moveto(m,j);
    over:=false;
    while not over do
      begin
        dig:=readkey;
        if dig<>#0 then
          begin
            if (ord(dig) >= 48) and (ord(dig) <= 57) then
              begin
                num:=concat(num,dig);
                outtext(dig);
              end;
            if dig=#8 then
              begin
                if getx>m then
                  begin
                    n:=getx-8;
                    setviewport(n,j,n+20,10,true);
                    clearviewport;
                    setviewport(0,0,getmaxx,getmaxy,true);
                    moveto(n,j);
                    delete(num,length(num),1);
                  end;
              end;
            if dig=#13 then
              begin
                over:=true;
                val(num,n,code);
              end;
          end
        else dig:=readkey;
      end;
    if (n>=1) and (n<=max[p]) then
      begin
        stpy:=n;
        incy:=scy*stpy;
        accept:=true;
      end
    else
      begin
        setviewport(230,0,340,10,true);
        clearviewport;
        setviewport(0,0,getmaxx,getmaxy,true);
      end;
  end;
delay(300);
setviewport(50,0,340,20,true);
clearviewport;
setviewport(stx+1,endy-8,endx+4,sty,true);
clearviewport;
setviewport(0,0,getmaxx,getmaxy,true);
```

```
      plotaxis;
      plottrack;
      xp:=x;
      yp:=round(y);
      saveimg;
      readaxs;
      over:=false;

    end;          {chstpy}

  procedure firstpt;
    begin
      oldtime:=tim;
      oldval:=value;

    end;          {firstpt}

  procedure endpt;
    var
      pts:integer;
    begin
      for i:=0 to (round(dur) div stpx)-1 do
        buf[i]:=ary[i];
      ary[tim div stpx]:=value;
      newtime:=tim;
      newval:=value;
      if newtime<oldtime then
        begin
          newtime:=oldtime;
          oldtime:=tim;
          newval:=oldval;
          oldval:=value;
        end;
      pts:=(newtime-oldtime) div round(uptime);
      if pts>0 then
        incval:=round((newval-oldval)/pts);
      for j:=1 to pts-1 do
        ary[((oldtime) div stpx)+j]:=oldval+j*incval;
      oldtime:=newtime;
      oldval:=newval;
      setviewport(stx+1,endy-8,endx+6,sty,true);
      clearviewport;
      setviewport(0,0,getmaxx,getmaxy,true);
      plotaxis;
      plottrack;
      xp:=x;
      yp:=round(y);
      saveimg;
      readaxs;

    end;          {endpt}
```

```
begin                {* procedure gredit}
    clrscr;
    grscreen;
    over:=false;
    while not over do
    begin
       key:=readkey;
       if key=#0 then
       begin
          key:=readkey;
          case ord(key) of
             72,80,77,75,115,116,117,119 : movcurser;
                                       59 : storept;
                                       63 : erasept;
                                       61 : chstpy;
                                       65 : endpt;
                                       68 : begin
                                              restorecrtmode;
                                              endedit:=true;
                                              over:=true;
                                            end;
                                       73 : begin
                                              restorecrtmode;
                                              k:=k-1;
                                              if k<1 then
                                                k:=vars;
                                              over:=true;
                                            end;
                                       81 : begin
                                              restorecrtmode;
                                              k:=k+1;
                                              if k>vars then
                                                k:=1;
                                              over:=true;
                                            end;
          end;
       end;
    end;

  end;            {* procedure gredit *}

end.            (* unit edit *)
```

```
(*  Written by Shailendra A. Chafekar; 1989-90  *)

unit trackgen;

(* This unit specifies the in-built phoneme set *)

  interface

  uses
    glbvars,basicfns;

  const
    maxphn=50;

  var
    phns,segst,segen,delt,delv,incv,value:integer;
    st,en,locphn:array[0..40] of integer;
    d,di:array[1..pars] of integer;
    phn:array[1..40] of string[20];
    lstphn:array[1..maxphn] of string[20];

  procedure tracks;          {* parameter tracks for in-built phonemes *)

  implementation

  uses
    crt;

  procedure tracks;

    procedure getphn;
      begin
        k:=wherey;
        if k>=23 then
          begin
            k:=4;
            window(1,k,80,25);
            clrscr;
            window(1,1,80,25);
            gotoxy(1,k);
          end;
        write('Enter phoneme number ',i,' : ');
        j:=wherex;
        k:=wherey;
        while not accept do
          begin
            readln(phn[i]);
            p:=1;
            if phn[i]<>'' then
              begin
                while (p<=maxphn) and (not accept) do
```

```
                begin
                  if phn[i]=lstphn[p] then
                    begin
                      accept:=true;
                    end;
                  p:=p+1;
                end;
            end;
          if not accept then
            begin
              window(j,k,80,k);
              clrscr;
              window(1,1,80,25);
              gotoxy(j,k);
              error;
            end;
          end;

      end;          {getphn}

    procedure gettim;
      begin
        write('Enter the start and end pts in ms : ');
        j:=wherex;
        k:=wherey;
        while not accept do
          begin
            readln(st[i],en[i]);
            if (st[i]<dur-uptime) and (en[i]<=dur-uptime) then
              begin
                if (st[i]>pend) and (st[i]<en[i]) then
                  begin
                    accept:=true;
                    pend:=en[i];
                  end;
              end;
            if not accept then
              begin
                window(j,k,80,k+2);
                clrscr;
                window(1,1,80,25);
                gotoxy(j,k);
                error;
              end;
          end;

      end;          {gettim}

    procedure plztracks;

    { parameter tracks for all-pole cascade/parallel synthesizer }
```

```
procedure bit;
  begin
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=390;d[9]:=1990;d[10]:=2550;d[11]:=50;d[12]:=100;d[13]:=140;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure beet;
  begin
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=270;d[9]:=2290;d[10]:=3010;d[11]:=60;d[12]:=200;d[13]:=400;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure bet;
  begin
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=530;d[9]:=1840;d[10]:=2480;d[11]:=60;d[12]:=90;d[13]:=200;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure bait;
  begin
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=330;d[9]:=2020;d[10]:=2600;d[11]:=55;d[12]:=100;d[13]:=200;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure bat;
  begin
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=660;d[9]:=1720;d[10]:=2410;d[11]:=70;d[12]:=150;d[13]:=320;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure cot;
  begin
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=700;d[9]:=1220;d[10]:=2600;d[11]:=130;d[12]:=70;d[13]:=160;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure caught;
  begin
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=570;d[9]:=840;d[10]:=2410;d[11]:=90;d[12]:=100;d[13]:=80;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure book;
```

```
    begin
      d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
      d[8]:=440;d[9]:=1020;d[10]:=2240;d[11]:=80;d[12]:=100;d[13]:=80;
      d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
      d[22]:=250;d[21]:=0;d[23]:=250;
    end;
  procedure boot;
    begin
      d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
      d[8]:=300;d[9]:=870;d[10]:=2240;d[11]:=60;d[12]:=80;d[13]:=60;
      d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
      d[22]:=250;d[21]:=0;d[23]:=250;
    end;
  procedure but;
    begin
      d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
      d[8]:=640;d[9]:=1190;d[10]:=2390;d[11]:=80;d[12]:=50;d[13]:=140;
      d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
      d[22]:=250;d[21]:=0;d[23]:=250;
    end;
  procedure bird;
    begin
      d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
      d[8]:=490;d[9]:=1350;d[10]:=1690;d[11]:=100;d[12]:=60;d[13]:=110;
      d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
      d[22]:=250;d[21]:=0;d[23]:=250;
    end;

  procedure w;
    begin
      d[3]:=100;d[4]:=50;d[5]:=0;d[6]:=0;d[7]:=0;
      d[8]:=290;d[9]:=610;d[10]:=2150;d[11]:=50;d[12]:=80;d[13]:=60;
      d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
      d[22]:=250;d[21]:=0;d[23]:=250;
    end;
  procedure y;
    begin
      d[3]:=100;d[4]:=50;d[5]:=0;d[6]:=0;d[7]:=0;
      d[8]:=260;d[9]:=2070;d[10]:=3020;d[11]:=40;d[12]:=250;d[13]:=500;
      d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
      d[22]:=250;d[21]:=0;d[23]:=250;
    end;
  procedure r;
    begin
      d[3]:=100;d[4]:=50;d[5]:=0;d[6]:=0;d[7]:=0;
      d[8]:=310;d[9]:=1060;d[10]:=1380;d[11]:=70;d[12]:=100;d[13]:=120;
      d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
      d[22]:=250;d[21]:=0;d[23]:=250;
    end;
  procedure l;
    begin
```

```
    d[3]:=100;d[4]:=50;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=310;d[9]:=1050;d[10]:=2880;d[11]:=50;d[12]:=100;d[13]:=280;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;


procedure ma;
  begin
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=480;d[9]:=1270;d[10]:=2130;d[11]:=40;d[12]:=200;d[13]:=200;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=450;d[21]:=0;d[23]:=270;
  end;
procedure na;
  begin
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=480;d[9]:=1340;d[10]:=2470;d[11]:=40;d[12]:=300;d[13]:=300;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=450;d[21]:=0;d[23]:=270;
  end;


procedure f;
  begin
    d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;
    d[8]:=340;d[9]:=1100;d[10]:=2080;d[11]:=200;d[12]:=120;d[13]:=150;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=57;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure v;
  begin
    d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;
    d[8]:=220;d[9]:=1100;d[10]:=2080;d[11]:=60;d[12]:=90;d[13]:=120;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=57;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure thin;
  begin
    d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;
    d[8]:=320;d[9]:=1290;d[10]:=2540;d[11]:=200;d[12]:=90;d[13]:=200;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=28;d[20]:=48;
    d[22]:=250;d[21]:=0;d[23]:=250;
    end;
procedure that;
  begin
    d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;
    d[8]:=270;d[9]:=1290;d[10]:=2540;d[11]:=60;d[12]:=80;d[13]:=170;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=28;d[20]:=48;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
```

```
procedure sass;
  begin
    d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;
    d[8]:=320;d[9]:=1390;d[10]:=2530;d[11]:=200;d[12]:=80;d[13]:=200;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=52;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure zoos;
  begin
    d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;
    d[8]:=240;d[9]:=1390;d[10]:=2530;d[11]:=70;d[12]:=60;d[13]:=180;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=52;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure shoe;
  begin
    d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;d[8]:=300;
    d[9]:=1840;d[10]:=2750;d[11]:=200;d[12]:=1000;d[13]:=300;
    d[14]:=0;d[15]:=0;d[16]:=57;d[17]:=48;d[18]:=48;d[19]:=46;
    d[20]:=0;d[22]:=250;d[21]:=0;d[23]:=250;
  end;

procedure pa;
  begin
    d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;d[8]:=400;d
    [9]:=1100;d[10]:=2150;d[11]:=300;d[12]:=150;d[13]:=220;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;
     d[20]:=63;d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure ba;
  begin
    d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;
    d[8]:=200;d[9]:=1100;d[10]:=2150;d[11]:=60;d[12]:=110;d[13]:=130;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=63;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure ta;
  begin
    d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;
    d[8]:=400;d[9]:=1600;d[10]:=2600;d[11]:=300;d[12]:=120;d[13]:=250;
    d[14]:=0;d[15]:=0;d[16]:=30;d[17]:=45;d[18]:=57;d[19]:=63;
    d[20]:=0;d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure da;
  begin
    d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;
    d[8]:=200;d[9]:=1600;d[10]:=2600;d[11]:=60;d[12]:=100;d[13]:=170;
    d[14]:=0;d[15]:=0;d[16]:=47;d[17]:=60;d[18]:=62;d[19]:=60;
    d[20]:=0;d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure ka;
```

```
    begin
      d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;d[8]:=300;
      d[9]:=1990;d[10]:=2850;d[11]:=250;d[12]:=160;d[13]:=330;
      d[14]:=0;d[15]:=0;d[16]:=53;d[17]:=43;d[18]:=45;d[19]:=45;
      d[20]:=0;d[22]:=250;d[21]:=0;d[23]:=250;
      if (locphn[i+1]>5) then d[15]:=60;
    end;
  procedure ga;
    begin
      d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;d[8]:=200;
      d[9]:=1990;d[10]:=2850;d[11]:=60;d[12]:=150;d[13]:=280;
      d[14]:=0;d[15]:=0;d[16]:=53;d[17]:=43;d[18]:=45;d[19]:=45;
      d[20]:=0;d[22]:=250;d[21]:=0;d[23]:=250;
      if (locphn[i+1]>5) then d[15]:=60;
    end;

  procedure ch;
    begin
      d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;d[8]:=350;
      d[9]:=1800;d[10]:=2820;d[11]:=200;d[12]:=90;d[13]:=300;
      d[14]:=0;d[15]:=0;d[16]:=44;d[17]:=60;d[18]:=53;d[19]:=53;
      d[20]:=0;d[22]:=250;d[21]:=0;d[23]:=250;
    end;
  procedure judge;
    begin
      d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;d[8]:=260;
      d[9]:=1800;d[10]:=2820;d[11]:=60;d[12]:=80;d[13]:=270;
      d[14]:=0;d[15]:=0;d[16]:=44;d[17]:=60;d[18]:=53;d[19]:=53;
      d[20]:=0;d[22]:=250;d[21]:=0;d[23]:=250;
    end;

  begin       {procedure plztracks}
    case locphn[i] of
      1:bit;2:beet;3:bet;4:bait;
      5:bat;6:cot;7:caught;8:book;
      9:boot;10:but;11:bird;12:w;
      13:y;14:r;15:l;16:ma;
      17:na;18:f;19:v;20:thin;
      21:that;22:sass;23:zoos;24:shoe;
      25:pa;26:ba;27:ta;28:da;
      29:ka;30:ga;31:ch;32:judge;
    end;

  end;          {procedure plztracks}


procedure fricstracks;

{ parameter tracks for pole-zero cascade synthesizer }

  procedure bit;
```

```
    begin
      d[2]:=0;
      d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
      d[8]:=390;d[9]:=1990;d[10]:=2550;d[11]:=50;d[12]:=100;d[13]:=140;
      d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
      d[22]:=250;d[21]:=0;d[23]:=250;
    end;
  procedure beet;
    begin
      d[2]:=0;
      d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
      d[8]:=270;d[9]:=2290;d[10]:=3010;d[11]:=60;d[12]:=200;d[13]:=400;
      d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
      d[22]:=250;d[21]:=0;d[23]:=250;
    end;
  procedure bet;
    begin
      d[2]:=0;
      d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
      d[8]:=530;d[9]:=1840;d[10]:=2480;d[11]:=60;d[12]:=90;d[13]:=200;
      d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
      d[22]:=250;d[21]:=0;d[23]:=250;
    end;
  procedure bait;
    begin
      d[2]:=0;
      d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
      d[8]:=330;d[9]:=2020;d[10]:=2600;d[11]:=55;d[12]:=100;d[13]:=200;
      d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
      d[22]:=250;d[21]:=0;d[23]:=250;
    end;
  procedure bat;
    begin
      d[2]:=0;
      d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
      d[8]:=660;d[9]:=1720;d[10]:=2410;d[11]:=70;d[12]:=150;d[13]:=320;
      d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
      d[22]:=250;d[21]:=0;d[23]:=250;
    end;
  procedure cot;
    begin
      d[2]:=0;
      d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
      d[8]:=700;d[9]:=1220;d[10]:=2600;d[11]:=130;d[12]:=70;d[13]:=160;
      d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
      d[22]:=250;d[21]:=0;d[23]:=250;
    end;
  procedure caught;
    begin
      d[2]:=0;
      d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
```

```
    d[8]:=570;d[9]:=840;d[10]:=2410;d[11]:=90;d[12]:=100;d[13]:=80;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure book;
  begin
    d[2]:=0;
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=440;d[9]:=1020;d[10]:=2240;d[11]:=80;d[12]:=100;d[13]:=80;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure boot;
  begin
    d[2]:=0;
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=300;d[9]:=870;d[10]:=2240;d[11]:=60;d[12]:=80;d[13]:=60;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure but;
  begin
    d[2]:=0;
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=640;d[9]:=1190;d[10]:=2390;d[11]:=80;d[12]:=50;d[13]:=140;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure bird;
  begin
    d[2]:=0;
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=490;d[9]:=1350;d[10]:=1690;d[11]:=100;d[12]:=60;d[13]:=110;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;

procedure w;
  begin
    d[2]:=0;
    d[3]:=100;d[4]:=50;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=290;d[9]:=610;d[10]:=2150;d[11]:=50;d[12]:=80;d[13]:=60;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure y;
  begin
    d[2]:=0;
    d[3]:=100;d[4]:=50;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=260;d[9]:=2070;d[10]:=3020;d[11]:=40;d[12]:=250;d[13]:=500;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
```

```
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure r;
  begin
    d[2]:=0;
    d[3]:=100;d[4]:=50;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=310;d[9]:=1060;d[10]:=1380;d[11]:=70;d[12]:=100;d[13]:=120;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure l;
  begin
    d[2]:=0;
    d[3]:=100;d[4]:=50;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=310;d[9]:=1050;d[10]:=2880;d[11]:=50;d[12]:=100;d[13]:=280;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=250;d[21]:=0;d[23]:=250;
  end;


procedure ma;
  begin
    d[2]:=0;
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=480;d[9]:=1270;d[10]:=2130;d[11]:=40;d[12]:=200;d[13]:=200;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=450;d[21]:=0;d[23]:=270;
  end;
procedure na;
  begin
    d[2]:=0;
    d[3]:=100;d[4]:=60;d[5]:=0;d[6]:=0;d[7]:=0;
    d[8]:=480;d[9]:=1340;d[10]:=2470;d[11]:=40;d[12]:=300;d[13]:=300;
    d[14]:=0;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=0;d[19]:=0;d[20]:=0;
    d[22]:=450;d[21]:=0;d[23]:=270;
  end;


procedure f;
  begin
    d[2]:=1;
    d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;d[8]:=340;
    d[9]:=1100;d[10]:=2080;d[11]:=200;d[12]:=120;d[13]:=150;
    d[14]:=5000;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=9952;d[19]:=0;
    d[20]:=57;d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure v;
  begin
    d[2]:=1;
    d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;d[8]:=220;
    d[9]:=1100;d[10]:=2080;d[11]:=60;d[12]:=90;d[13]:=120;
```

```
    d[14]:=5000;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=9952;d[19]:=0;
    d[20]:=57;d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure thin;
  begin
    d[2]:=1;
    d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;d[8]:=320;
    d[9]:=1290;d[10]:=2540;d[11]:=200;d[12]:=90;d[13]:=200;
    d[14]:=4420;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=1210;d[19]:=28;
    d[20]:=48;d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure that;
  begin
    d[2]:=1;
    d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;d[8]:=270;
    d[9]:=1290;d[10]:=2540;d[11]:=60;d[12]:=80;d[13]:=170;
    d[14]:=4502;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=1152;d[19]:=28;
    d[20]:=48;d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure sass;
  begin
    d[2]:=1;
    d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;d[8]:=320;
    d[9]:=1390;d[10]:=2530;d[11]:=200;d[12]:=80;d[13]:=200;
    d[14]:=5000;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=10170;d[19]:=52;
    d[20]:=0;d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure zoos;
  begin
    d[2]:=1;
    d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;d[8]:=240;
    d[9]:=1390;d[10]:=2530;d[11]:=70;d[12]:=60;d[13]:=180;
    d[14]:=5000;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=12000;d[19]:=52;
    d[20]:=0;d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure shoe;
  begin
    d[2]:=1;
    d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;d[8]:=300;
    d[9]:=1840;d[10]:=2750;d[11]:=200;d[12]:=1000;d[13]:=300;
    d[14]:=4473;d[15]:=3652;d[16]:=3153;d[17]:=48;d[18]:=928;d[19]:=218;
    d[20]:=267;d[22]:=250;d[21]:=0;d[23]:=250;
  end;


procedure pa;
  begin
    d[2]:=1;
    d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;d[8]:=400;
    d[9]:=1100;d[10]:=2150;d[11]:=300;d[12]:=150;d[13]:=220;
    d[14]:=5000;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=7745;d[19]:=0;
```

```
    d[20]:=63;d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure ba;
  begin
    d[2]:=1;
    d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;d[8]:=200;
    d[9]:=1100;d[10]:=2150;d[11]:=60;d[12]:=110;d[13]:=130;
    d[14]:=5000;d[15]:=0;d[16]:=0;d[17]:=0;d[18]:=7745;d[19]:=0;
    d[20]:=63;d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure ta;
  begin
    d[2]:=3;
    d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;d[8]:=400;
    d[9]:=1600;d[10]:=2600;d[11]:=300;d[12]:=120;d[13]:=250;
    d[14]:=3975;d[15]:=3363;d[16]:=2627;d[17]:=45;d[18]:=487;d[19]:=267;
    d[20]:=256;d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure da;
  begin
    d[2]:=3;
    d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;d[8]:=200;
    d[9]:=1600;d[10]:=2600;d[11]:=60;d[12]:=100;d[13]:=170;
    d[14]:=4232;d[15]:=3497;d[16]:=2693;d[17]:=60;d[18]:=755;d[19]:=249;
    d[20]:=185;d[22]:=250;d[21]:=0;d[23]:=250;
  end;
procedure ka;
  begin
    d[2]:=3;
    d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;d[8]:=300;
    d[9]:=1990;d[10]:=2850;d[11]:=250;d[12]:=160;d[13]:=330;
    d[14]:=4465;d[15]:=3673;d[16]:=3180;d[17]:=43;d[18]:=929;d[19]:=267;
    d[20]:=274;d[22]:=250;d[21]:=0;d[23]:=250;
    if (locphn[i+1]>5) then
     begin
       d[2]:=4;
       d[14]:=4520;d[15]:=3687;d[16]:=3199;d[17]:=2343;d[18]:=950;
       d[19]:=211;d[20]:=266;d[21]:=224;
     end;

  end;
procedure ga;
  begin
    d[2]:=3;
    d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;d[8]:=200;
    d[9]:=1990;d[10]:=2850;d[11]:=60;d[12]:=150;d[13]:=280;
    d[14]:=4464;d[15]:=3673;d[16]:=3180;d[17]:=43;d[18]:=927;d[19]:=214;
    d[20]:=260;d[22]:=250;d[21]:=0;d[23]:=250;

    if (locphn[i+1]>5) then
      begin
```

```
            d[2]:=4;
            d[14]:=4520;d[15]:=3687;d[16]:=3199;d[17]:=2343;d[18]:=950;
            d[19]:=211;d[20]:=256;d[21]:=201;
          end;

      end;


    procedure ch;
      begin
        d[2]:=3;
        d[3]:=0;d[4]:=0;d[5]:=60;d[6]:=0;d[7]:=0;d[8]:=350;
        d[9]:=1800;d[10]:=2820;d[11]:=200;d[12]:=90;d[13]:=300;
        d[14]:=4420;d[15]:=3673;d[16]:=2882;d[17]:=60;d[18]:=896;d[19]:=215;
        d[20]:=295;d[22]:=250;d[21]:=0;d[23]:=250;
      end;
    procedure judge;
      begin
        d[2]:=3;
        d[3]:=100;d[4]:=47;d[5]:=60;d[6]:=47;d[7]:=0;d[8]:=260;
        d[9]:=1800;d[10]:=2820;d[11]:=60;d[12]:=80;d[13]:=270;
        d[14]:=4420;d[15]:=3673;d[16]:=2882;d[17]:=60;d[18]:=896;d[19]:=215;
        d[20]:=295;d[22]:=250;d[21]:=0;d[23]:=250;
       end;

    begin          {procedure fricstracks}
      case locphn[i] of
        1:bit;2:beet;3:bet;4:bait;
        5:bat;6:cot;7:caught;8:book;
        9:boot;10:but;11:bird;12:w;
        13:y;14:r;15:l;16:ma;
        17:na;18:f;19:v;20:thin;
        21:that;22:sass;23:zoos;24:shoe;
        25:pa;26:ba;27:ta;28:da;
        29:ka;30:ga;31:ch;32:judge;
       end;

    end;             { procedure fricstracks}

  begin            {* procedure tracks *}
    clrscr;

    lstphn[1]:='bit';lstphn[2]:='beet';lstphn[3]:='bet';
    lstphn[4]:='bait';lstphn[5]:='bat';lstphn[6]:='cot';
    lstphn[7]:='caught';lstphn[8]:='book';lstphn[9]:='boot';
    lstphn[10]:='but';lstphn[11]:='bird';lstphn[12]:='w';
    lstphn[13]:='y';lstphn[14]:='r';lstphn[15]:='l';
    lstphn[16]:='ma';lstphn[17]:='na';lstphn[18]:='f';
    lstphn[19]:='v';lstphn[20]:='thin';lstphn[21]:='that';
    lstphn[22]:='sass';lstphn[23]:='zoos';lstphn[24]:='shoe';
    lstphn[25]:='pa';lstphn[26]:='ba';lstphn[27]:='ta';
```

```
lstphn[28]:='da';lstphn[29]:='ka';lstphn[30]:='ga';
lstphn[31]:='ch';lstphn[32]:='judge';

accept:=false;
write('Number of phonemes   < 1..',maxphn,' > : ');
k:=wherex;
while not accept do
  begin
    num:='';
      getresp('number','j','j','j','j');
      if (number>=1) and (number<=40) then
        accept:=true
      else
        begin
          window(k,1,80,1);
          clrscr;
          window(1,1,80,25);
          gotoxy(k,1);
          error;
        end;
  end;
phns:=number;
writeln;
writeln;
pend:=0;
for i:=1 to phns do
  begin
    writeln;
    accept:=false;
    getphn;
    writeln;
    accept:=false;
    gettim;
    writeln;
  end;
 for i:=1 to phns do
   begin
     for j:=1 to maxphn do
       begin
         if phn[i]=lstphn[j] then
           locphn[i]:=j;
       end;
   end;
   en[0]:=0;
   st[phns+1]:=round(dur-uptime);
   locphn[phns+1]:=0;
   for i:=1 to phns+1 do
     begin
       if cas_par then
         plztracks;
       if cas then
```

```
  fricstracks;
if (i<phns+1) then
  begin
    for j:=round(st[i]/uptime) to round(en[i]/uptime) do
      begin
        for k:=1 to vars do
          begin
            vari:=lvar[k];
            posn:=(vari-1)+round(j*pars);
            partrc^[posn]:=d[vari];
          end;
      end;
  end;
if (i=1) then
  begin
    if cas_par then
      for j:=3 to 23 do
        di[j]:=d[j];
    if cas then
      for j:=2 to 23 do
        di[j]:=d[j];
    di[4]:=0;
    di[5]:=0;
    di[6]:=0;
    di[7]:=0;
    if cas_par then
      for j:=14 to 21 do
        di[j]:=0;
  end;
if (i=phns+1) then
  begin
    if cas_par then
      for j:=3 to 23 do
        d[j]:=di[j];
    if cas then
      for j:=2 to 23 do
        d[j]:=di[j];
    d[4]:=0;
    d[5]:=0;
    d[6]:=0;
    d[7]:=0;
    if cas_par then
      for j:=14 to 21 do
        d[j]:=0;
  end;
  segst:=round(en[i-1]/uptime);
  segen:=round(st[i]/uptime);
  delt:=segen-segst;
  for k:=1 to vars do
    begin
      vari:=lvar[k];
```

```
                delv:=d[vari]-di[vari];
                if delt<>0 then
                  begin
                    incv:=round(delv/delt);
                    for j:=segst to segen do
                      begin
                        posn:=(vari-1)+round(j*pars);
                        value:=di[vari]+(j-segst)*incv;
                        if delv>0 then if value>d[vari] then
                          value:=d[vari];
                        if delv<0 then if value<d[vari] then
                          value:=d[vari];
                        partrc^[posn]:=value;
                        if (cas) and (vari=2) and (j>segst) then
                          begin
                            if locphn[i] < 18 then
                            partrc^[posn]:=d[vari];
                          end;
                      end;
                  end;
              end;
            if cas_par then
              for j:=3 to 23 do
                di[j]:=d[j];
            if cas then
              for j:=2 to 23 do
                di[j]:=d[j];
        end;

  end;          {* procedure tracks *)

end.            (* unit trackgen *)
```

```
(*  Written by Shailendra A. Chafekar; 1989-90  *)

unit plz;

(* This unit implements the all-pole cascade/parallel synthesizer *)

  interface

  uses
    glbvars,basicfns;

  var
    sw:integer;
    a1p,a2p,a3p,a4p,a5p,a6p,an,ab,y1p,y11p,y12p,y2p,y21p,y22p,
    y3p,y31p,y32p,y4p,y41p,y42p,y5p,y51p,y52p,y6p,y61p,y62p,
    velcas,yab,velpar:real;


  procedure plzinit;                {* initialization *}
  procedure plzpars;                {* forms parameter array *}
  procedure plzcoefficients;        {* calculates coeficients *}
  procedure plzwaveform;            {* outputs speech sequence *}


  implementation

  procedure plzinit;
    begin
      nam[1]:='SW';nam[2]:='NF';nam[3]:='F0';nam[4]:='AV';nam[5]:='AF';
      nam[6]:='AVS';nam[7]:='AH';nam[8]:='F1';nam[9]:='F2';
      nam[10]:='F3';nam[11]:='BW1';nam[12]:='BW2';nam[13]:='BW3';
      nam[14]:='A1';nam[15]:='A2';nam[16]:='A3';nam[17]:='A4';
      nam[18]:='A5';nam[19]:='A6';nam[20]:='AB';nam[22]:='FNZ';
      nam[21]:='AN';nam[33]:='BWNZ';nam[24]:='UPDT';nam[25]:='SR';
      nam[26]:='G0';nam[27]:='F4';nam[28]:='F5';nam[29]:='F6';
      nam[30]:='BW4';nam[31]:='BW5';nam[32]:='BW6';nam[23]:='FNP';
      nam[34]:='BWNP';nam[35]:='FGP';nam[36]:='BWGP';
      nam[37]:='FGZ';nam[38]:='BWGZ';nam[39]:='BWGS';

      conf[1]:='C';conf[2]:='C';conf[3]:='C';conf[4]:='C';conf[5]:='C';
      conf[6]:='C';conf[7]:='C';conf[8]:='C';conf[9]:='C';conf[10]:='C';
      conf[11]:='C';conf[12]:='C';conf[13]:='C';conf[14]:='C';
      conf[15]:='C';conf[16]:='C';conf[17]:='C';conf[18]:='C';
      conf[19]:='C';conf[20]:='C';conf[21]:='C';conf[22]:='C';
      conf[23]:='C';conf[24]:='C';conf[25]:='C';conf[26]:='C';
      conf[27]:='C';conf[28]:='C';conf[29]:='C';conf[30]:='C';
      conf[31]:='C';conf[32]:='C';conf[33]:='C';conf[34]:='C';
      conf[35]:='C';conf[36]:='C',conf[37]:='C';

      min[1]:=0;min[2]:=4;min[3]:=0;min[4]:=0;
      min[5]:=0;min[6]:=0;min[7]:=0;min[8]:=150;
```

```
    min[9]:=500;min[10]:=1300;min[11]:=40;min[12]:=40;
    min[13]:=40;min[14]:=0;min[15]:=0;min[16]:=0;
    min[17]:=0;min[18]:=0;min[19]:=0;min[20]:=0;
    min[22]:=200;min[21]:=0;min[33]:=50;min[24]:=2;
    min[25]:=5000;min[26]:=0;min[27]:=2500;min[28]:=3500;
    min[29]:=4000;min[30]:=100;min[31]:=150;min[32]:=200;
    min[23]:=200;min[34]:=50;min[35]:=0;min[36]:=100;
    min[37]:=0;min[38]:=100;min[39]:=100;

    max[1]:=1;max[2]:=6;max[3]:=500;max[4]:=80;
    max[5]:=80;max[6]:=80;max[7]:=80;max[8]:=900;
    max[9]:=2500;max[10]:=3500;max[11]:=500;max[12]:=500;
    max[13]:=500;max[14]:=80;max[15]:=80;max[16]:=80;
    max[17]:=80;max[18]:=80;max[19]:=80;max[20]:=80;
    max[22]:=700;max[21]:=80;max[33]:=500;max[24]:=50;
    max[25]:=20000;max[26]:=80;max[27]:=4500;max[28]:=4900;
    max[29]:=4999;max[30]:=500;max[31]:=700;max[32]:=2000;
    max[23]:=500;max[34]:=500;max[35]:=600;max[36]:=2000;
    max[37]:=5000;max[38]:=9000;max[39]:=1000;

    def[1]:=0;def[2]:=4;def[3]:=0;def[4]:=0;
    def[5]:=0;def[6]:=0;def[7]:=0;def[8]:=450;
    def[9]:=1450;def[10]:=2400;def[11]:=50;def[12]:=70;
    def[13]:=110;def[14]:=0;def[15]:=0;def[16]:=0;
    def[17]:=0;def[18]:=0;def[19]:=0;def[20]:=0;
    def[22]:=250;def[21]:=0;def[33]:=100;def[24]:=5;
    def[25]:=10000;def[26]:=47;def[27]:=3300;def[28]:=3750;
    def[29]:=4900;def[30]:=250;def[31]:=200;def[32]:=1000;
    def[23]:=250;def[34]:=100;def[35]:=0;def[36]:=100;
    def[37]:=1500;def[38]:=6000;def[39]:=200;

  end;                {* procedure plzinit *}

procedure plzpars;
  begin
    sw:=par[1];nf:=par[2];f0:=par[3];av:=par[4];af:=par[5];
    avs:=par[6];ah:=par[7];f1:=par[8];f2:=par[9];f3:=par[10];
    bw1:=par[11];bw2:=par[12];bw3:=par[13];a1p:=par[14];a2p:=par[15];
    a3p:=par[16];a4p:=par[17];a5p:=par[18];a6p:=par[19];ab:=par[20];
    fnz:=par[22];an:=par[21];bwnz:=par[33];uptime:=par[24];
    samprate:=par[25];g0:=par[26];f4:=par[27];f5:=par[28];
    f6:=par[29];bw4:=par[30];bw5:=par[31];bw6:=par[32];
    fnp:=par[23];bwnp:=par[34];fgp:=par[35];bwgp:=par[36];
    fgz:=par[37];bwgz:=par[38];bwgs:=par[39];

  end;          {*  procedure plzpars *}


procedure plzcoefficients;
  const
    {SCALE FACTORS}
```

```
    a1sca:=-58;
    a2sca:=-65;
    a3sca:=-73;
    a4sca:=-78;
    a5sca:=-79;
    a6sca:=-80;
    ansca:=-58;
    absca:=-84;
    avsca:=-72;
    ahsca:=-102;
    afsca:=-72;
    assca:=-44;
  var
    proxcor:array[1..10] of integer;
    cor:integer;
    pi,pi2:real;
    f1dev,f2dev,f3dev,a2cor,a3cor:real;
    f12cor,f23cor,f34cor,f21,f32,f43:real;


  begin
    pi:=4.0*arctan(1.0);
    {COR FOR FMT PROXIMITY}
    for i:= 1 to 10 do
      proxcor[i]:=11-i;
    {INITIALIZE IF BEGINNING OF UTT}
    if initcoeff then
      begin
        afprev:=0;sampint:=1/samprate;
        pi2:=pi*samprate;
        pi2:=2*pi;
        initcoeff:=false;
      end;
    voiced:=false;
    aspi:=false;
    frc:=false;
    vfrc:=false;

    {AMPL OF VOICING}
    if av>0 then
      voiced:=true;
    av:=g0+av+avsca;
    implse:=linamp(av);
    {AMPL OF ASP}
    if ah>0 then
      aspi:=true;
    ah:=g0+ah+ahsca;
    asp:=linamp(ah);
    {AMPL OF FRIC}
    if af>0 then
      frc:=true;
```

```
af:=g0+af+afsca;
{in all par mode, af=max(ah,af).}
if (af<ah) and (sw=1) then
  af:=ah;
fric:=linamp(af);
if voiced and frc then
  vfrc:=true;
{PLOSIVE BURST MARKER}
plrel:=0;
if((af-afprev)<50) then plrel:=linamp(g0+afsca+44);
afprev:=af;
{QU SIN VOICING FOR VOICED OBSTRUENTS}
avs:=g0+avs+assca;
sinamp:=linamp(avs);

{COR FOR FMT DEV}
f1dev:=f1/500;
a2cor:=f1dev*f1dev;
f2dev:=f2/1500;
a3cor:=a2cor*f2dev*f2dev;
{FIRST DIFF FOR F2}
a2cor:=a2cor/f2dev;
{COR FOR FMT PROX}
f12cor:=0;
f23cor:=0;
f34cor:=0;
f21:=f2-f1;
f32:=f3-f2-50;
f43:=f4-f3-150;
cor:=round(f21/ 50);
if cor<1 then
  cor:=1;
if cor >10 then
  cor:=10;
f12cor:=proxcor[cor];
cor:=round(f32/ 50);
if cor<1 then cor:=1;
if cor >10 then cor:=10;
f23cor:=proxcor[cor];
cor:=round(f43/ 50);
if cor<1 then
  cor:=1;
if cor >10 then
  cor:=10;
f34cor:=proxcor[cor];

{PAR FMT AMPLS}
a1p:=a1p+f12cor+a1sca;
a1p:=linamp(a1p);
a2p:=a2p+f12cor+f12cor+f23cor+a2sca;
a2p:=a2cor*linamp(a2p);
```

```
      a3p:=a3p+f23cor+f23cor+f34cor+a3sca;
      a3p:=a3cor*linamp(a3p);
      a4p:=a4p+f34cor+f34cor+a4sca;
      a4p:=a3cor*linamp(a4p);
      a5p:=a5p+a5sca;
      a5p:=a3cor*linamp(a5p);
      a6p:=a6p+a6sca;
      a6p:=a3cor*linamp(a6p);
      {PAR NASAL FMT}
      an:=an+ansca;
      an:= linamp(an);
      {BYPASS FRIC PATH}
      ab:=ab+absca;
      ab:=linamp(ab);

      {RES COEFFICIENTS}
      zero:=false;
      filcoef(f1,bw1,a1,b1,c1);
      filcoef(f2,bw2,a2,b2,c2);
      filcoef(f3,bw3,a3,b3,c3);
      filcoef(f4,bw4,a4,b4,c4);
      filcoef(f5,bw5,a5,b5,c5);
      filcoef(f6,bw6,a6,b6,c6);
      filcoef(fnp,bwnp,anp,bnp,cnp);
      zero:=true;
      filcoef(fnz,bwnz,anz,bnz,cnz);
      samperpitch:=1;
      {If f0,av,as are 0,no glottal pulse}
      if ((voiced) or (avs>0)) and (f0>0) then
        begin
          {if (f0>0) and (av>0) then begin}
          {Glottal wave more sinusoidal at high f0}
          bwgp:=round((bwgp*100) / f0);
          zero:=false;
          filcoef(fgp,bwgp,agp,bgp,cgp);
          fgs:=0;
          filcoef(fgs,bwgs,ags,bgs,cgs);
          zero:=true;
          filcoef(fgz,bwgz,agz,bgz,cgz);
          samperpitch:=round(samprate / f0);  {per pitch period}
          {Set agp constant in its midband}
          agp:=0.007;
          {f0 not to go below 40 Hz}
          if f0<40 then f0:=40;
          {Impulse strength prop to f0}
          implse:=implse*f0;
        end;

    end;            {*  procedure plzcoefficients *}

  procedure plzwaveform;
```

```
begin
  velcas:=0;
  {INITIALIZE AT THE START OF UTT}
  if start then
    begin
      {initialize filter outputs}
      y11c:=0;y12c:=0;          {**cascade model**}
      y21c:=0;y22c:=0;
      y31c:=0;y32c:=0;
      y41c:=0;y42c:=0;
      y51c:=0;y52c:=0;
      y61c:=0;y62c:=0;
      ynp1c:=0;ynp2c:=0;
      ynz1c:=0;ynz2c:=0;
      y11p:=0;y12p:=0;          {**parallel model**}
      y21p:=0;y22p:=0;
      y31p:=0;y32p:=0;
      y41p:=0;y42p:=0;
      y51p:=0;y52p:=0;
      y61p:=0;y62p:=0;
      ynp1:=0;ynp2:=0;
      ygp1:=0;ygp2:=0;          {**source shaping**}
      ygz1:=0;ygz2:=0;
      ygs1:=0;ygs2:=0;
      ygs3:=0;ygs4:=0;
      glotvelAL:=0;glotvelL:=0;
      lipvel:=0;
      aasp:=0;afric:=0;
      pitchcount:=1;
      modcount:=0;   {mod noise for voiced obstruents}
      plstep:=0;
      start:=false;
      pitchpd:=true;
    end;
    dasp:=(asp-aasp)/samperup;       {noise level raised linearly --}
    dfric:=(fric-afric)/samperup;      {--over samples per update time}
    {**MAIN LOOP**}
    for i:=1 to round(samperup) do
      begin
        if (pitchpd=true) and (samperpitch<>1) then
          begin
            inp:=implse;insin:=sinamp;
            modcount:=round(samperpitch / 2);  {set voicing and asp.}
            pitchcount:=round(samperpitch);
            pitchpd:=false;
          end
        else
          begin
            inp:=0;
            insin:=0;    {set input to zero bet impulses}
          end;
```

```
{Resonator RGP}
ygp:=agp*inp+bgp*ygp1+cgp*ygp2;
ygp2:=ygp1;ygp1:=ygp;
{Glottal zero}
ygz:=agz*ygp+bgz*ygz1+cgz*ygz2;
ygz2:=ygz1;ygz1:=ygp;
{quasi sin. voicing}
ygs:=ags*insin+bgs*ygs1+cgs*ygs2;
ygs2:=ygs1;ygs1:=ygs;
ygs:=agp*ygs+bgp*ygs3+cgp*ygs4;
ygs4:=ygs3;ygs3:=ygs;
{Total glottal vol vel}
glotvelA:=ygz+ygs;
{Impose lip radiation;hi-pass}
glotvel:=glotvelA-glotvelAL;
glotvelAL:=glotvelA;
{Turbulence for asp and frication}
noise:=0;
for j:= 1 to 16 do
  begin
    randomize;
    noise:=noise+random;
  end;
noise:=noise-8.;
if modcount<= 0 then noise:=noise/2.;
modcount:=modcount-1;
aasp:=aasp+dasp;
aspvel:=aasp*noise;
if aspi then
  glotvel:=glotvel+aspvel;
afric:=afric+dfric;
{plosive burst}
if plrel>50 then
  begin
    plstep:=-plrel;
    plrel:=0;
  end;
fricvel:=afric*noise;


{**CASCADE SIM**}
if (sw=0) and (voiced or aspi) then
  begin
    y6c:=glotvel;
    if nf=6 then
      begin
        y6c:=a6*glotvel+b6*y61c+c6*y62c;
        y62c:=y61c;y61c:=y6c;
      end;
    y5c:=y6c;
    if nf>=5 then
```

```
      begin
        y5c:=a5*y6c+b5*y51c*y52c;
        y52c:=y51c;y51c:=y5c;
      end;
    y4c:=a4*y5c+b4*y41c+c4*y42c;
    y42c:=y41c;y41c:=y4c;
    y3c:=a3*y4c+b3*y31c+c3*y32c;
    y32c:=y31c;y31c:=y3c;
    y2c:=a2*y3c+b2*y21c+c2*y22c;
    y22c:=y21c;y21c:=y2c;
    y1c:=a1*y2c+b1*y11c+c1*y12c;
    y12c:=y11c;y11c:=y1c;
    {Nasal zero}
    ynzc:=anz*y1c+bnz*ynz1c+cnz*ynz2c;
    ynz2c:=ynz1c;ynz1c:=y1c;
    {Nasal pole}
    ynpc:=anp*ynzc+bnp*ynp1c+cnp*ynp2c;
    ynp2c:=ynp1c;ynp1c:=ynpc;
    velcas:=ynpc;
    lipvel:=velcas;
    glotvel:=0;
    glotvell:=0;
  end;


if (sw=1) or frc or vfrc then
  begin
    {**PARALLEL SIM**}
    {voiced excitn for F1}
    y1p:=a1*a1p*glotvel+b1*y11p+c1*y12p;
    y12p:=y11p;y11p:=y1p;
    {nasal pole excitn by first diff}
    glotvell:=glotvel-glotvell;
    glotvell:=glotvel;
    ynp:=anp*glotvell+bnp*ynp1+cnp*ynp2;
    ynp2:=ynp1;ynp1:=ynp;
    {F2-F4 excitn fric noise and first diff}
    vel:=fricvel+glotvell;
    y2p:=a2*a2p*vel+b2*y21p+c2*y22p;
    y22p:=y21p;y21p:=y2p;
    y3p:=a3*a3p*vel+b3*y31p+c3*y32p;
    y32p:=y31p;y31p:=y3p;
    y4p:=a4*a4p*vel+b4*y41p+c4*y42p;
    y42p:=y41p;y41p:=y4p;
    {F5,F6 excitn fric noise}
    y5p:=a5*a5p*fricvel+b5*y51p+c5*y52p;
    y52p:=y51p;y51p:=y5p;
    y6p:=a6*a6p*fricvel+b6*y61p+c6*y62p;
    y62p:=y61p;y61p:=y6p;
    {Bypass path}
    yab:=ab*fricvel;
```

```
            velpar:=y1p-y2p+y3p-y4p+y5p-y6p+ynp-yab;
            lipvel:=velcas+velpar+plstep;
            plstep:=plstep*0.995;
          end;
        if (not voiced) and (not frc) and (not aspi) then
          lipvel:=0;
        lipvel:=(lipvel/32000)*2048*170;
        if samperpitch<>1 then
          pitchcount:=pitchcount-1;
        if pitchcount=0 then
          pitchpd:=true;
        if lipvel>32767.0 then
          lipvel:=32767.0;
        if lipvel<-32767.0 then
          lipvel:=-32767.0;
        if lipvel>mx then
          mx:=lipvel;
        if -lipvel>mx then
          mx:=-lipvel;
        wav^[i+p]:=round(lipvel);
      end;

  end;          {*  procedure plzwaveform *}

end.          (* unit plz *)
```

```
(*  Written by Shailendra A. Chafekar, 1989-90  *)

unit frics;

{* This unit implements the pole-zero cascade synthesizer *}

  interface

  uses
    glbvars,basicfns;

  var
    nz:integer;
    velvoiced,velfric,fz1,fz2,fz3,fz4,fz5,fz6,fz7,fz8,fz9,fz0,bz1,bz2,bz3,
    bz4,bz5,bz6,bz7,bz8,bz9,bz0,r1,s1,t1,r2,s2,t2,r3,s3,t3,r4,s4,t4,
    r5,s5,t5,r6,s6,t6,r7,s7,t7,r8,s8,t8,r9,s9,t9,r0,s0,t0,
    yz1,yz2,yz3,yz4,yz5,yz6,yz7,yz8,yz9,yz0,velpole,yz11,yz12,
    yz21,yz22,yz31,yz32,yz41,yz42,yz51,yz52,yz61,yz62,yz71,yz72,
    yz81,yz82,yz91,yz92,yz01,yz02 :real;


  procedure  fricsinit;            {* initialization *}
  procedure fricspars;             {* forms parameter array *}
  procedure fricscoefficients;     {* calculates coefficients *}
  procedure fricswaveform;         {* outputs speech sequence *}



  implementation

  procedure fricsinit;
    begin
      nam[2]:='NZ';nam[1]:='NF';nam[3]:='F0';nam[4]:='AV';nam[5]:='AF';
      nam[6]:='AVS';nam[7]:='AH';nam[8]:='F1';nam[9]:='F2';nam[10]:='F3';
      nam[11]:='BW1';nam[12]:='BW2';nam[13]:='BW3';nam[14]:='FZ1';
      nam[15]:='FZ2';nam[16]:='FZ3';nam[17]:='FZ4';nam[18]:='BZ1';
      nam[19]:='BZ2';nam[20]:='BZ3';nam[21]:='BZ4';nam[22]:='FNZ';
      nam[33]:='BWNZ';nam[24]:='UPDT';nam[25]:='SR';nam[26]:='GO';
      nam[27]:='F4';nam[28]:='F5';nam[29]:='F6';nam[30]:='BW4';nam[31]:='BW5';
      nam[32]:='BW6';nam[23]:='FNP';nam[34]:='BWNP';nam[35]:='FGP';
      nam[36]:='BWGP';nam[37]:='FGZ';nam[38]:='BWGZ';nam[39]:='BWGS';

      conf[1]:='C';conf[2]:='C';conf[3]:='C';conf[4]:='C';conf[5]:='C';
      conf[6]:='C';conf[7]:='C';conf[8]:='C';conf[9]:='C';
      conf[10]:='C';conf[11]:='C';conf[12]:='C';conf[13]:='C';
      conf[14]:='C';conf[15]:='C';conf[16]:='C';conf[17]:='C';conf[18]:='C';
      conf[19]:='C';conf[20]:='C';conf[21]:='C';conf[22]:='C';conf[23]:='C';
      conf[24]:='C';conf[25]:='C';conf[26]:='C';conf[27]:='C';conf[28]:='C';
      conf[29]:='C';conf[30]:='C';conf[31]:='C';conf[32]:='C';conf[33]:='C';
      conf[34]:='C';conf[35]:='C';conf[36]:='C';conf[37]:='C';
      conf[38]:='C';conf[39]:='C';
```

```
    min[2]:=0;min[1]:=4;min[3]:=0;min[4]:=0;
    min[5]:=0;min[6]:=0;min[7]:=0;min[8]:=150;
    min[9]:=500;min[10]:=1300;min[11]:=40;min[12]:=40;
    min[13]:=40;min[14]:=0;min[15]:=0;min[16]:=0;
    min[17]:=0;min[18]:=0;min[19]:=0;min[20]:=0;
    min[22]:=200;min[21]:=0;min[33]:=50;min[24]:=2;
    min[25]:=5000;min[26]:=0;min[27]:=2500;min[28]:=3500;
    min[29]:=4000;min[30]:=100;min[31]:=150;min[32]:=200;
    min[23]:=200;min[34]:=50;min[35]:=0;min[36]:=100;
    min[37]:=0;min[38]:=100;min[39]:=100;

    max[2]:=4;max[1]:=6;max[3]:=500;max[4]:=80;
    max[5]:=80;max[6]:=80;max[7]:=80;max[8]:=900;
    max[9]:=2500;max[10]:=3500;max[11]:=500;max[12]:=500;
    max[13]:=500;max[14]:=80;max[15]:=80;max[16]:=80;
    max[17]:=80;max[18]:=800;max[19]:=80;max[20]:=80;
    max[22]:=700;max[21]:=80;max[33]:=500;max[24]:=50;
    max[25]:=20000;max[26]:=80;max[27]:=4500;max[28]:=4900;
    max[29]:=4999;max[30]:=500;max[31]:=700;max[32]:=2000;
    max[23]:=500;max[34]:=500;max[35]:=600;max[36]:=2000;
    max[37]:=5000;max[38]:=9000;max[39]:=1000;

    def[2]:=0;def[1]:=4;def[3]:=0;def[4]:=0;
    def[5]:=0;def[6]:=0;def[7]:=0;def[8]:=450;
    def[9]:=1450;def[10]:=2400;def[11]:=50;def[12]:=70;
    def[13]:=110;def[14]:=0;def[15]:=0;def[16]:=0;
    def[17]:=0;def[18]:=0;def[19]:=0;def[20]:=0;
    def[22]:=250;def[21]:=0;def[33]:=100;def[24]:=5;
    def[25]:=10000;def[26]:=47;def[27]:=3300;def[28]:=3750;
    def[29]:=4900;def[30]:=250;def[31]:=200;def[32]:=1000;
    def[23]:=250;def[34]:=100;def[35]:=0;def[36]:=100;
    def[37]:=1500;def[38]:=6000;def[39]:=200;

  end;          {* procedure fricsinit *}

procedure fricspars;
  begin
    nz:=par[2];nf:=par[1];f0:=par[3];av:=par[4];af:=par[5];
    avs:=par[6];ah:=par[7];f1:=par[8];f2:=par[9];f3:=par[10];
    bw1:=par[11];bw2:=par[12];bw3:=par[13];fz1:=par[14];
    fz2:=par[15];fz3:=par[16];fz4:=par[17];bz1:=par[18];
    bz2:=par[19];bz3:=par[20];bz4:=par[21];fnz:=par[22];
    fnp:=par[23];uptime:=par[24];samprate:=par[25];
    g0:=par[26];f4:=par[27];f5:=par[28];f6:=par[29];
    bw4:=par[30];bw5:=par[31];bw6:=par[32];bwnz:=par[33];
    bwnp:=par[34];fgp:=par[35];bwgp:=par[36];fgz:=par[37];
    bwgz:=par[38];bwgs:=par[39];

  end;          {* procedure fricspars *}
```

```
procedure fricscoefficients;
  const
  {SCALE FACTORS}
    avsca=-72;
    ahsca=-102;
    afsca=-72;
    assca=-44;
  var
    proxcor:array[1..10] of integer;
    cor:integer;
    pi,pi2:real;

  begin
    pi:=4.0*arctan(1.0);
    {INITIALIZE IF BEGINNING OF UTT}
    if initcoeff then
    begin
      afprev:=0;sampint:=1/samprate;
      pi2:=pi*samprate;
      pi2:=2*pi;
      initcoeff:=false;
    end;

   voiced:=false;
   frc:=false;
   vfrc:=false;
   aspi:=false;

   {AMPL OF VOICING}
   if av>0 then
     voiced:=true;
   av:=g0+av+avsca;
   implse:=linamp(av);
   {AMPL OF ASP}
   if ah>0 then
     aspi:=true;
   ah:=g0+ah+ahsca;
   asp:=linamp(ah);
   {AMPL OF FRIC}
   if af>0 then
     frc:=true;
   if (frc=true) and (voiced=true) then
     vfrc:=true;
   af:=g0+af+afsca;
   fric:=linamp(af);
   {PLOSIVE BURST MARKER}
   plrel:=0;
   if((af-afprev)<50) then
     plrel:=linamp(g0+afsca+44);
   afprev:=af;
```

```
    {QU SIN VOICING FOR VOICED OBSTRUENTS}
    avs:=g0+avs+assca;
    sinamp:=linamp(avs);

    {RES COEFFICIENTS}
    zero:=false;
    filcoef(f1,bw1,a1,b1,c1);
    filcoef(f2,bw2,a2,b2,c2);
    filcoef(f3,bw3,a3,b3,c3);
    filcoef(f4,bw4,a4,b4,c4);
    filcoef(f5,bw5,a5,b5,c5);
    filcoef(f6,bw6,a6,b6,c6);
    filcoef(fnp,bwnp,anp,bnp,cnp);
    zero:=true;
    filcoef(fnz,bwnz,anz,bnz,cnz);
    samperpitch:=1;
    {If f0,av,as are 0,no glottal pulse}
    if ((voiced or (avs>0)) and (f0>0) then
      begin
        {Glottal wave more sinusoidal at high f0}
        bwgp:=round((bwgp*100) / f0);
        zero:=false;
        filcoef(fgp,bwgp,agp,bgp,cgp);
        fgs:=0;
        filcoef(fgs,bwgs,ags,bgs,cgs);
        zero:=true;
        filcoef(fgz,bwgz,agz,bgz,cgz);
        {Set agp constant in its midband}
        agp:=0.007;
        {f0 not to go below 40 Hz}
        if f0<40 then
        f0:=40;
        samperpitch:=round(samprate / f0);  {per pitch period}
        {Impulse strength prop to f0}
        implse:=implse*f0;
      end;
    if (frc=true) or (vfrc=true) then
      begin
        zero:=true;
        filcoef(fz1,bz1,r1,s1,t1);
        filcoef(fz2,bz2,r2,s2,t2);
        filcoef(fz3,bz3,r3,s3,t3);
        filcoef(fz4,bz4,r4,s4,t4);
        (* filcoef(fz5,bz5,r5,s5,t5);
        filcoef(fz6,bz6,r6,s6,t6);
        filcoef(fz7,bz7,r7,s7,t7);
        filcoef(fz8,bz8,r8,s8,t8); *)
      end;

  end;              {*  procedure fricscoefficients *)
```

```
procedure fricswaveform;
  var
    i,j:integer;
  begin
    velvoiced:=0;
    {INITIALIZE AT THE START OF UTT}
    if start then
      begin
        {initialize filter outputs}
        y11c:=0;y12c:=0;        {**cascade model**}
        y21c:=0;y22c:=0;
        y31c:=0;y32c:=0;
        y41c:=0;y42c:=0;
        y51c:=0;y52c:=0;
        y61c:=0;y62c:=0;
        ynp1c:=0;ynp2c:=0;
        ynz1c:=0;ynz2c:=0;
        yz11:=0;yz12:=0;        {**parallel model**}
        yz21:=0;yz22:=0;
        yz31:=0;yz32:=0;
        yz41:=0;yz42:=0;
        yz51:=0;yz52:=0;
        yz61:=0;yz62:=0;
        yz71:=0;yz72:=0;
        yz81:=0;yz82:=0;
        yz91:=0;yz92:=0;
        yz01:=0;yz01:=0;
        ygp1:=0;ygp2:=0;        {**source shaping**}
        ygz1:=0;ygz2:=0;
        ygs1:=0;ygs2:=0;
        ygs3:=0;ygs4:=0;
        lipvel:=0;
        glotvelAL:=0;
        aasp:=0;afric:=0;
        pitchcount:=1;
        modcount:=0;   {mod noise for voiced obstruents}
        plstep:=0;
        start:=false;
        pitchpd:=true;
      end;
      dasp:=(asp-aasp)/samperup;       {noise level raised linearly --}
      dfric:=(fric-afric)/samperup;     {--over samples per update time}
      {**MAIN LOOP**}
      for i:=1 to round(samperup) do
        begin
          if (pitchpd=true) and (samperpitch<>1) then
            begin
              inp:=impulse;
              insin:=sinamp;
              modcount:=round(samperpitch / 2); {set voicing and asp.}
              pitchcount:=round(samperpitch);
```

```
    pitchpd:=false;
  end
else
  begin
    inp:=0;
    insin:=0;        {set input to zero bet impulses}
  end;
{Resonator RGP}
 ygp:=agp*inp+bgp*ygp1+cgp*ygp2;
 ygp2:=ygp1;ygp1:=ygp;
 {Glottal zero}
 ygz:=agz*ygp+bgz*ygz1+cgz*ygz2;
 ygz2:=ygz1;ygz1:=ygp;
 {quasi sin. voicing}
 ygs:=ags*insin+bgs*ygs1+cgs*ygs2;
 ygs2:=ygs1;ygs1:=ygs;
 ygs:=agp*ygs+bgp*ygs3+cgp*ygs4;
 ygs4:=ygs3;ygs3:=ygs;
 {Total glottal vol vel}
 glotvelA:=ygz+ygs;
 {Impose lip radiation;hi-pass}
 glotvel:=glotvelA-glotvelAL;
 glotvelAL:=glotvelA;
 {Turbulence for asp and frication}
 noise:=0;
 for j:= 1 to 16 do
   begin
     randomize;
     noise:=noise+random;
   end;
 noise:=noise-8.;
 if modcount<= 0 then noise:=noise/2.;
 modcount:=modcount-1;
 aasp:=aasp+dasp;
 aspvel:=aasp*noise;
 if (aspi=true) then
   glotvel:=glotvel+aspvel;
 afric:=afric+dfric;
 {plosive burst}
 if plrel>50 then
   begin
     plstep:=-plrel;
     plrel:=0;
   end;
 fricvel:=afric*noise;
 if (vfrc=true) then
 glotvel:=glotvel+fricvel;

 {**CASCADE SIM**}

 y6c:=glotvel;
```

```
if nf=6 then
6 begin
    y6c:=a6*glotvel+b6*y61c+c6*y62c;
    y62c:=y61c;y61c:=y6c;
6 end;
y5c:=y6c;
if nf>=5 then
7 begin
    y5c:=a5*y6c+b5*y51c*y52c;
    y52c:=y51c;y51c:=y5c;
7 end;
y4c:=a4*y5c+b4*y41c+c4*y42c;
y42c:=y41c;y41c:=y4c;
y3c:=a3*y4c+b3*y31c+c3*y32c;
y32c:=y31c;y31c:=y3c;
y2c:=a2*y3c+b2*y21c+c2*y22c;
y22c:=y21c;y21c:=y2c;
y1c:=a1*y2c+b1*y11c+c1*y12c;
y12c:=y11c;y11c:=y1c;
{Nasal zero}
ynzc:=anz*y1c+bnz*ynz1c+cnz*ynz2c;
ynz2c:=ynz1c;ynz1c:=y1c;
{Nasal pole}
ynpc:=anp*ynzc+bnp*ynp1c+cnp*ynp2c;
ynp2c:=ynp1c;ynp1c:=ynpc;
velpole:=ynpc;
lipvel:=velpole;
if ((frc) or (vfrc)) and (nz>0) then
8 begin

  {**PARALLEL SIM**}

    yz1:=r1*velpole+s1*yz11+t1*yz12;
     yz12:=yz11;
     yz11:=velpole;

     yz2:=yz1;
     if nz>=2 then
9     begin
        yz2:=r2*yz1+s2*yz21+t2*yz22;
        yz22:=yz21;yz21:=yz1;
9     end;

     yz3:=yz2;
     if nz>=3 then
10    begin
        yz3:=r3*yz2+s3*yz31+t3*yz32;
        yz32:=yz31;yz31:=yz2;
10    end;

     yz4:=yz3;
```

```
      if nz>=4 then
        begin
          yz4:=r4*yz3+s4*yz41+t4*yz42;
          yz42:=yz41;yz41:=yz3;
        end;

      yz5:=yz4;
      if nz>=5 then
        begin
          yz5:=r5*yz4+s5*yz51+t5*yz52;
          yz52:=yz51;yz51:=yz4;
        end;

      yz6:=yz5;
      if nz>=6 then
        begin
          yz6:=r6*yz5+s6*yz61+t6*yz62;
          yz62:=yz61;yz61:=yz5;
        end;

      yz7:=yz6;
      if nz>=7 then
        begin
          yz7:=r7*yz6+s5*yz71+t7*yz72;
          yz72:=yz71;yz71:=yz6;
        end;

      yz8:=yz7;
      if nz>=8 then
        begin
          yz8:=r8*yz7+s8*yz81+t8*yz82;
          yz82:=yz81;yz81:=yz7;
        end;

      lipvel:=yz8+plstep;
      plstep:=plstep*0.995;

  end;

  if (not voiced) and (not frc) and (not aspi) then
    lipvel:=0;
  lipvel:=(lipvel/32000)*2048*170;
  if samperpitch<>1 then
    pitchcount:=pitchcount-1;
  if pitchcount=0 then
    pitchpd:=true;
  if lipvel>32767.0
    then lipvel:=32767.0;
  if lipvel<-32767.0 then
    lipvel:=-32767.0;
  if lipvel>mx then
```

```
            mx:=lipvel;
          if -lipvel>mx then
            mx:=-lipvel;
          wav^[i+p]:=round(lipvel);
      end;

  end;          {* procedure fricswaveform *}

end.          (* unit frics*)
```

```
{   Written by Shailendra A. Chafekar, 1989-90   *)

program spsynth;

(* Main program; needs a suitable graphics driver :
   CGA.BGI/ EGAVGA.BGI/ HERC.BGI/ any other *.bgi driver
   specified by Borland *)

  uses
    crt,glbvars,basicfns,plz,frics,trackgen;

  procedure synth_rtn;
    begin
      clrscr;
      gotoxy(10,2);
      writeln('Parameter tracks now in RAM.');
      gotoxy(10,3);
      writeln('Synthesis starts.');
      gotoxy(10,4);
      write('Press Enter. ');
      readln;
      mx:=1;
      initcoeff:=true;
      start:=true;
      p:=0;
      p:=round((dur/1000)*def[25]);
      gotoxy(10,6);
      write('Duration=',dur,'   Total samples=',p);
      p:=0;
      gotoxy(10,7);
      write('Samples  generated : ',p);
      samperup:=round((def[24]/1000)*def[25]);
      for update:= 0 to round(totup-1) do
        begin
          readvars;
          if cas_par then
            begin
              plzpars;
              plzcoefficients;
              plzwaveform;
            end;
          if cas then
            begin
              fricspars;
              fricscoefficients;
              fricswaveform;
            end;
          p:=p+round(samperup);
          gotoxy(10,7);
          write('Samples  generated : ',p);
          write('   Max  : ',(20*ln(mx))/ln(10):1:3,'  dB');
```

```
    end;
   mx:=(20*ln(mx))/ln(10);
   gotoxy(10,7);
   write('Samples  generated : ',p);
   write('    Max Value  : ',mx:1:3,'  dB');
   gotoxy(10,10);
   write('Save speech file ?  <Y/N> : ');
   resp:='';
   getresp('choice','y','Y','n','N');
   if resp='Y' then
    .begin
       gotoxy(10,12);
       write('Name of the speech file ? ');
       readln(filename);
       assign(wavefile,filename);
       gotoxy(1,24);
       write('Saving speech file ...');
       rewrite(wavefile);
       for i:=1 to p do
         writeln(wavefile,wav^[i]);
       close(wavefile);
       delline;
     end;
   gotoxy(10,16);
   write('Press Enter. ');
   readln;

 end;                {* procedure synth_rtn *}

begin               (* main program spsynth *)
  new(partrc);
  new(wav);
  textbackground(black);
  textcolor(white);
  clrscr;
  quit:=false;
  synthtype:=true;
  options:=false;
  ramtracks:=false;
  while not quit do
    begin
      if synthtype then
        type_of_synth;
      if options then
       begin
         operation;
         if not (ramtracks) then
           begin
             if cas_par then
               plzinit;
             if cas then
```

```
                fricsinit;
              end;
            found:=true;
            if extfile then
             trcread;
            if found then
              begin
                if gentrc then
                  begin
                    confset;
                    findpos;
                    if not phnset then
                      preparetrack;
                    if phnset then
                      tracks;
                    ramtracks:=true;
                   clrscr;
                   gotoxy(10,4);
                   write('Parameter tracks prepared.');
                   gotoxy(10,6);
                   write('Save parameter tracks?   <Y/N> : ');
                   resp:='';
                   getresp('choice','y','n','Y','N');
                   if resp='Y' then
                    trcwrite;
                  end;
                if st_synth then
                  begin
                    if extfile or ramtracks then
                     begin
                       readcons;
                       synth_rtn;
                       ramtracks:=false;
                     end
                    else
                     begin
                       gotoxy(1,24);
                       write('Tracks not yet in RAM...');
                       delay(250);
                       error;
                       delline;
                     end;
                  end;
              end;
          end;
      end;
  end;
dispose(partrc);
dispose(wav);
gotoxy(1,24);
write('Quitting to DOS ...');
delay(600);
```

```
  clrscr;

end.                (* main program spsynth *)
```

The ZERO_LOC Program

```
(* Written by Shailendra A. Chafekar; 1989-90 *)
program zero_loc;

  (* This program finds out the zeros in the fricative spectra *)

  uses
    crt;
  const
    m=12;
  type
    complex = record
                re,im:real;
              end;
    arraym = array[0..m] of real;
    cArraym = array[1..m] of complex;
  var
    deg:integer;
    f,bw:array[1..6] of integer;
    af,bf,cf,db:array[1..7] of real;
    x:arraym;
    pk:cArraym;

  procedure flt;
    const
      T=0.0001;
      a1s=-58;
      a2s=-65;
      a3s=-73;
      a4s=-78;
      a5s=-79;
      a6s=-80;
      abs=-84;
    var
      i,j,k,cor,f21,f32,f43,f12cor,f23cor,f34cor :integer;
      ab,cc,db1,db2,f1dev,f2dev,a2cor,a3cor:real;
      proxcor:array[1..10] of integer;
      s:array[1..28] of real;
      d:array[1..11] of real;
    begin
      clrscr;
      writeln('This procedure returns the filter coefficients');
      writeln('of parallel second order resonators.');
      writeln;
      for i:=1 to 10 do
        proxcor[i]:=11-i;
      for i:=1 to 6 do
        begin
          write('Enter F',i,' : ');
          readln(f[i]);
          writeln;
          write('Enter BW',i,' : ');
```

```
    readln(bw[i]);
    writeln;
    write('Enter gain in dB , A',i,' : ');
    readln(db[i]);
    writeln;
  end;
write('Enter bypass path gain AB : ');
readln(ab);
s[1]:=65536.0;
for k:=2 to 28 do
  s[k]:=s[k-1]/2;
d[1]:=1.8;d[2]:=1.6;d[3]:=1.43;d[4]:=1.26;
d[5]:=1.12;d[6]:=1.0;d[7]:=0.89;d[8]:=0.792;
d[9]:=0.702;d[10]:=0.623;d[11]:=0.555;
f21:=0;f32:=0;f43:=0;
f1dev:=f[1]/500;f2dev:=f[2]/1500;
a2cor:=f1dev*f1dev;
a3cor:=a2cor*f2dev*f2dev;
if f2dev<>0 then
  a2cor:=a2cor/f2dev;
f21:=f[2]-f[1];
f32:=f[3]-f[2]-50;
f43:=f[4]-f[3]-150;
cor:=round(f21/50);
if cor>10 then
  cor :=10;
if cor<1 then
  cor:=1;
f12cor:=proxcor[cor];
cor:=round(f32/50);
if cor>10 then
  cor :=10;
if cor<1 then
  cor:=1;
f23cor:=proxcor[cor];
cor:=round(f43/50);
if cor>10 then
  cor :=10;
if cor<1 then
  cor:=1;
f34cor:=proxcor[cor];
db[1]:=db[1]+f12cor+a1s;
db[2]:=db[2]+f12cor+f12cor+f23cor+a2s;
db[3]:=db[3]+f23cor+f23cor+f34cor+a3s;
db[4]:=db[4]+f34cor+f34cor+a4s;
db[5]:=db[5]+a5s;
db[6]:=db[6]+a6s;
ab:=ab+abs;
for i:=1 to 6 do
  begin
    if db[i]<-72 then
```

```
      db[i]:=-72;
    if db[i]>96 then
      db[i]:=96;
  end;
if ab<-72 then
  ab:=-72;
if ab>96 then
  ab:=96;
for i:=1 to 6 do
  begin
    cc:=-pi*bw[i]*T;
    cc:=exp(cc);
    cf[i]:=-cc*cc;
    bf[i]:=2*cc*Cos(2*pi*f[i]*T);
    af[i]:=1-bf[i]-cf[i];
    db1:=int(db[i]/6);
    db2:=db[i]-6*db1;
    db1:=s[17-round(db1)];
    db2:=d[6-round(db2)];
    db[i]:=db1*db2;
    af[i]:=db[i]*af[i];
  end;
af[2]:=a2cor*af[2];
for i:=3 to 6 do
  af[i]:=a3cor*af[i];
db1:=int(ab/6);
db2:=ab-6*db1;
db1:=s[17-round(db1)];
db2:=d[6-round(db2)];
ab:=db1*db2;
af[7]:=ab;
bf[7]:=0;
cf[7]:=0;
for i:=1 to 7 do
  begin
    if af[i]<=0.001 then
      begin
        af[i]:=0;
        bf[i]:=0;
        cf[i]:=0;
      end;
  end;
clrscr;
writeln('    **** The filter coefficients are ****');
writeln('     *The A coefficient includes dB gain*');
writeln;
writeln('    Freq      BW         A        B        C');
writeln;
for j:=1 to 6 do
    writeln(f[j]:8,'    ',bw[j]:4,af[j]:12:3,bf[j]:10:5,cf[j]:10:5);
writeln;
```

```
    writeln('The bypass path amplitude is : ',ab:2:3);
    writeln;
    write('Press ENTER.');
    readln;

  end;          {*procedure flt*}

procedure zeros;
  var
    y,u,v,n,d:array[0..12] of real;
    t,r:real;
    i,j,k,p:integer;
  begin
    clrscr;
    writeln('This procedure finds the coefficients of the numerator');
    writeln('polynomial arising from the addition of a ');
    writeln('maximum of seven second order polynomials.');
    writeln;
    for i:=0 to 12 do
      begin
        x[i]:=0;y[i]:=0;
        u[i]:=0;v[i]:=0;
        n[i]:=0;d[i]:=0;
      end;
    x[0]:=af[1];
    y[0]:=1;
    y[1]:=-bf[1];
    y[2]:=-cf[1];
    p:=2;
    for k:=2 to 7 do
      begin
        u[0]:=af[k];
        v[0]:=1;
        v[1]:=-bf[k];
        v[2]:=-cf[k];
        for j:=0 to p do
          begin
            for i:=0 to p do
              begin
                t:=x[i]*v[j];
                r:=u[i]*y[i];
                n[i+j]:=n[i+j]+t+r;
                t:=y[i]*v[j];
                d[i+j]:=d[i+j]+t;
              end;
          end;
        p:=p+2;
        for i:=0 to 12 do
          begin
            x[i]:=n[i];
            y[i]:=d[i];
```

```
          end;
        for i:=0 to 12 do
          begin
            n[i]:=0;
            d[i]:=0;
          end;
      end;
    writeln('The coefficients of the numerator polynomial');
    writeln('For power of z ranging from 0 to -12 are :');
    writeln;
    writeln('   z-index    Coefficient');
    writeln;
    for i:=0 to 12 do
      writeln('     ',-i:3,'       ', x[i]:10:5);
    writeln;
    write('Press ENTER.');
    readln;

  end;            {*procedure zeros*}

procedure rootspoly(ak:arraym;var ck:cArraym);
  var
    j:integer;
    noconv:boolean;

  procedure root(n:integer;a:arraym;var p:cArraym);

    (* This procedure implements the Lin-Bairstow Algorithm *)

    const
      errmax = 1E-5;
      maxiter = 100;
    var
      i,iter,sign:integer;
      det,discr,q1,q2,dq1,dq2,q1i,q2i:real;
      exit:boolean;
      b,c:arraym;

    begin  {prcedure root}
      exit:=false;
      while (n>2) and (abs(a[0]) < errmax) do
        begin
          p[n].re:=0;
          p[n].im:=0;
          for i:=0 to (n-1) do
            a[i]:=a[i+1];
          n:=n-1;
        end;
      if n>2 then
        begin
          if abs(a[2])<errmax then
```

```
            begin
              q1:=1;
              q2:=1;
            end
          else
            begin
              for i:=0 to n do
                a[i]:=a[i]/a[n];
              q1:=a[0]/a[2];
              q2:=a[1]/a[2];
              q1i:=0;q2i:=0;
            end;
          iter:=0;
          b[n]:=1;
          c[n]:=1;
          repeat
            b[n-1]:=a[n-1]-q2;
            c[n-1]:=b[n-1]-q2;
            for i:=(n-2) downto 0 do
              begin
                b[i]:=a[i]-q2*b[i+1]-q1*b[i+2];
                c[i]:=b[i]-q2*c[i+1]-q1*c[i+2];
              end;
            det:=c[2]*c[2]+c[3]*b[1]-c[3]*c[1];
            if det=0 then det:=errmax;
            dq1:=(b[0]*c[2]+b[1]*(b[1]-c[1]))/det;
            dq2:=(b[1]*c[2]-b[0]*c[3])/det;
            q1i:=q1;q2i:=q2;
            q1:=q1+dq1;
            q2:=q2+dq2;
            iter:=iter+1;
          until (iter=maxiter) or
((abs(q1-q1i)<errmax)and(abs(q2-q2i)<errmax));
          (*until (iter=maxiter) or ((abs(dq1)+abs(dq2))<errmax);*)
          if iter=maxiter then
            begin
              clrscr;
              writeln;writeln;writeln;writeln;
              write('                    **** Root Procedure ');
              writeln('Does Not Converge ****');
              noconv:=true;
              exit:=true;
            end;
        end
      else
        case n of
          0 : exit:=true;
          1 : begin
                p[1].im:=0;
                p[1].re:=-a[0];
                exit:=true;
```

```
            end;
          2 : begin
                q1:=a[0]/a[2];
                q2:=a[1]/a[2];
              end;
        end;
      if exit=false then
        begin
          discr:=q2*q2-4*q1;
          sign:=1;
          for i:=n downto (n-1) do
            with p[i] do
             begin
               if discr<0 then
                 begin
                   im:=sign*sqrt(-discr)/2;
                   re:=-q2/2;
                 end
               else
                 begin
                   im:=0;
                   re:=(-q2+sign*sqrt(discr))/2;
                 end;
               sign:=-1;
             end;
          n:=n-2;
          for i:=0 to n do
            a[i]:=b[i+2];
          root(n,a,p);
        end;

    end;   {procedure root}

  begin     {*procedure rootspoly*}
    clrscr;
    writeln('This procedure returns the roots of a polynomial of the form :
');
    writeln;
    writeln('   a[0] + a[1]*x +....+ a[n-1]*x**(n-1) + a[n-2]*x**n');
    writeln;
    writeln('The maximum number of coefficients is ',m+1,'.');
    writeln;
    deg:=1;
    for j:=1 to m do
      if abs(ak[j])>0 then
        deg:=j;
      noconv:=false;
      root(deg,ak,ck);
      if not noconv then
        begin
          writeln('***** Roots of the polynomial are *****');
```

```
        writeln;
        writeln('    Root      Re Part        Im Part');
        for j:=1 to deg do
          begin
            with ck[j] do
              writeln(j:6,re:15:5,im:15:5);
          end;
      end;
    writeln;
    write('Press ENTER.');
    readln;

  end;          {*rootspoly*}

procedure zerofreq(dk:cArraym);
  var
    i,j:integer;
    r,sn,cs,tn:real;
    a,b,bcof,ccof,fz,bw:array[1..m] of real;
  begin
    clrscr;
    writeln('This procedure calculates the antiresonance frequency ');
    writeln('and bandwidth of a second order antiresonator.');
    writeln;
    with dk[1] do
      begin
        a[1]:=re;
        b[1]:=im;
      end;
    j:=2;
    for i:=2 to deg do
      begin
        with dk[i] do
          begin
            a[j]:=re;
            b[j]:=im;
            if (a[j]=a[j-1]) and (b[j]=-b[j-1]) then
              j:=j-1;
            j:=j+1;
          end;
      end;
    for i:=1 to j-1 do
      begin
        bcof[i]:=a[i];
        ccof[i]:=sqr(a[i])+sqr(b[i]);
        r:=sqrt(ccof[i]);
        bw[i]:=(abs(ln(1/r)))/(pi*1E-4);
        cs:=a[i];
        sn:=abs(b[i]);
        tn:=arctan(sn/cs);
        if (cs<0) then
```

```
            tn:=tn+pi;
          fz[i]:=tn*5000/pi;
        end;
      writeln('    ** The antiresonance frequencies and bandwidths are **');
      writeln;
      writeln('    Section      Freq     BW');
      writeln;
      for i:=1 to (j-1) do
        writeln(i:8,round(fz[i]):13,round(abs(bw[i])):7);
      writeln;
      write('Press ENTER.');
      readln;

    end;            {*procedure zerofreq*}

  begin   (* main program zero_loc *)
    flt;
    zeros;
    rootspoly(x,pk);
    zerofreq(pk);

  end.    (* program zero_loc *)
```

The DAOUT Program

```
(* Written by Shailendra A. Chafekar; 1989-90 *)
program daout;

  (* This program is the driver for the 12-bit D/A converter
     of the MCR-4 card *)

  uses
    crt;
  type
    arr=array[1..20000] of integer;
    bytarr=array[1..20000] of byte;
  var
    i,m,min,max,sms:integer;
    x:arr;
    y,z:^bytarr;
    finis,over:boolean;
    p:char;
    name:string[20];
    fil:text;

  begin
    finis:=false;
    while not finis do
      begin
        clrscr;
        port[$308]:=0;
        new(y);
        new(z);
        write('File Name ? ');
        readln(name);
        assign(fil,name);
        min:=0;
        max:=0;
        sms:=0;
        reset(fil);
        while not eof(fil) do
          begin
            sms:=sms+1;
            readln(fil,x[sms]);
            if x[sms]<min then
              min:=x[sms];
            if x[sms]>max then
              max:=x[sms];
          end;
        close(fil);
        for i:=1 to sms do
          begin
            x[i]:=x[i]-min;
            x[i]:=trunc((x[i]/(max-min))*4095);
          end;
```

```
      for i:=1 to sms do
        begin
          y^[i]:=trunc(x[i]/16);
          z^[i]:=16*(x[i]-(16*y^[i]));
        end;
      writeln;
      write('Hit any key to start the playback.');
      p:=readkey;
      over:=false;
      while not over do
        begin
          for i:=1 to sms do
            begin
              port[$30e]:=z^[i];
              port[$30e]:=y^[i];
              for m:=1 to 3 do
                begin
                end;
            end;
          clrscr;
          write('Hit any key to play back ,"n" for next file ,');
          write('"." to end the session.');
          p:=readkey;
          if (p='n') or (p='N') or(p='.') then
            over:=true;
          if p='.' then
            finis:=true;
        end;
      dispose(y);
      dispose(z);
    end;

end.          (* program daout *)
```

APPENDIX B

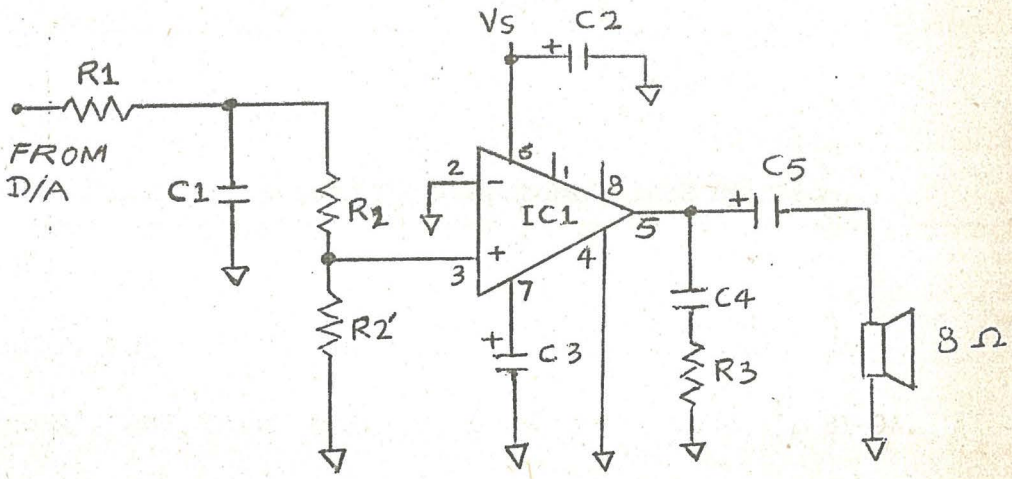
THE AUDIO POWER AMPLIFIER

The D/A converter used to output speech sequences under program control is part of a data acquisition system, MCR-4. To obtain a properly audible speech output a simple audio power amplifier was built using IC LM386.

LM386 is a low voltage audio power amplifier, and is capable of supplying up to 325 mW of power at the output. It can be operated with a wide range of supply voltages, typically 4-12 V. The voltage gain can be varied from 20 to 200. The inputs are ground referenced, and the input resistance is about 50kΩ.

The circuit diagram of the amplifier (National Semiconductor Linear Data Book, 1980) is shown in Fig. B.1.

The output from the D/A converter (range: -5 to +5 V), is lowpass filtered by the first order filter (R1, C1) with cutoff frequency of 5kHz, and then it is amplified by the LM386 stage. Output is capacitively coupled to the 8 Ω speaker.



$R_1 = 3.2\,k\Omega$ $R_2 = 100\,k\Omega$

$C_1 = 0.01\,\mu F$ $R_2' = 2\,k\Omega$

$C_3 = 10\,\mu F$ $C_5 = 250\,\mu F$

$R_3 = 10\,\Omega$ $C_4 = 0.1\,\mu F$

IC1 = LM386 $C_2 = 220\,\mu F$

Fig. B.1. Circuit diagram of the audio power amplifier.

REFERENCES

Flanagan, J.L. (1972). *Speech Analysis, Synthesis and Perception,* 2nd Edn. New York: Springer-Verlag.

Flanagan, J.L., Coker, C.H., Rabiner, L.R., Schafer, R.W., & Umeda, N. (1970). "Synthetic voices for computers", IEEE Spectrum, 7(10), pp 22-45.

Gold, B., & Rabiner, L.R. (1968). "Analysis of digital and analog formant synthesizers", IEEE Trans. Audio Electroacoust., AU-16, pp 81-94.

Huelsman, L.P. (1986). *Engineering and Scientific Computations in PASCAL.* New York: Harper & Row.

Johansson, B. (1966). "The use of the transposer for the management of the deaf child", Int. Audiol., 5, pp 362-372. Reprinted in Levitt, Pickette, & Houde (1980), pp 195-204.

Kirman, J.H. (1974). "Tactile communication of speech: a review and an analysis", Psychol. Bull., 80, pp 54-74. Reprinted in Levitt, Pickette, & Houde (1980), pp, 297-317.

Klatt, D.H. (1980). "Software for a cascade/parallel formant synthesizer", J. Acoust. Soc. Am., 67(3), pp 971-995.

Kuc, R. (1988). *Introduction to Digital Signal Processing.* McGraw-Hill: New York.

Levitt, H. (1973). "Speech processing aids for the deaf: an overview", IEEE Trans. Audio Electroacoust., AU-21, pp 269-273. Reprinted in Levitt, Pickette, & Houde (1980), pp 419-423.

Levitt, H., Pickette, J.M., & Houde, R.A., Eds., (1980). *Sensory Aids for the Hearing Impaired.* New York: IEEE Press.

Oppenheim, A.V. (1978). *Applications of Digital Signal Processing.* Englewood Cliffs, New Jersey: Prentice-Hall.

O'Shaughnessy, D. (1987). *Speech Communication: Human and Machine.* Reading, Massachusets: Addison-Wesley.

Pandey, P.C. (1987). *Speech Processing for Cochlear Prostheses.* Ph.D. Thesis, Dept. of Elect. Engg., University of Toronto.

Rabiner, L.R., & Gold, B. (1978). *Theory and Applications of Digital Signal Processing.* Englewood Cliffs, New Jersey: Prentice-Hall.

Rabiner, L.R., & Schafer, R.W. (1978). *Digital Processing of Speech Signals.* Englewood Cliffs, New Jersey: Prentice-Hall.

Vilchur, E. (1973). "Signal processing to improve speech intelligibilty in perceptive deafness", J. Acoust. Soc. Am., 53, pp 1646-1657. Reprinted in Levitt, Pickette, & Houde (1980), pp 410-423.

Watson, N.A., & Knudsen, V.O. (1940). "Selective amplification in hearing aids", J. Acoust. Soc. Am., 2, pp 406-419. Reprinted in Levitt, Pickette, & Houde (1980), pp 39-52.