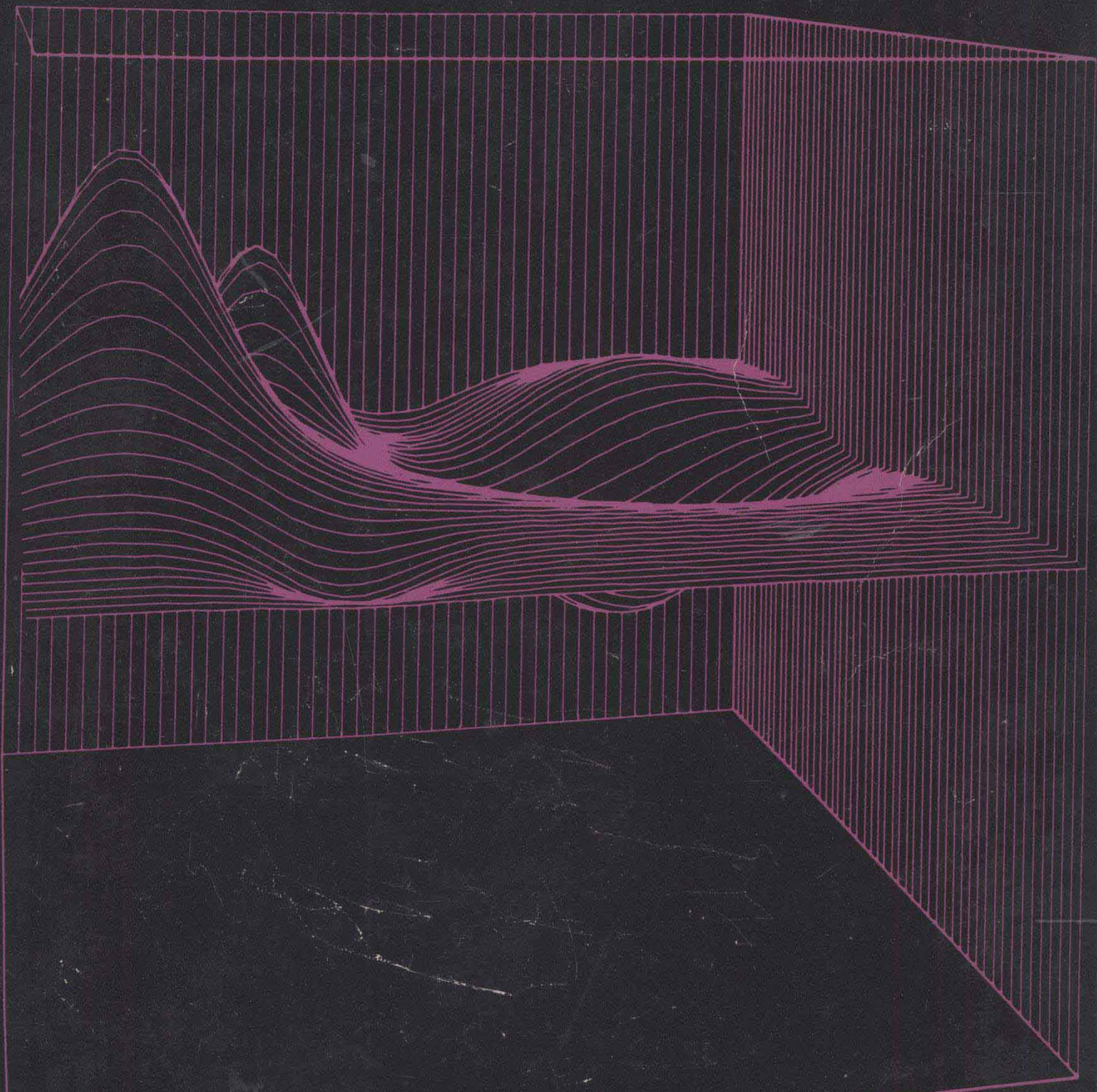
OSBORNE/McGraw-Hill

Some Common Pascal Programs



Based on the book
Some Common BASIC Programs

Some Common Pascal Programs

Based on the book
Some Common BASIC Programs

Published by
Osborne/McGraw-Hill
630 Bancroft Way
Berkeley, California 94710
U. S. A.

For information on translations and book distributors outside of the U.S.A., please write Osborne/McGraw-Hill at the above address.

SOME COMMON PASCAL PROGRAMS

1234567890 DLDL 8765432

ISBN 0-931988-73-X

Cover design by Peter Kunz

Some Common Pascal Programs

Introduction

These 76 programs solve common problems in the areas of finance, business, mathematics, statistics, and home budgeting. All programs are ready to be typed into your computer and run.

You don't have to be a programmer to use this book, but you must understand the subject matter of the programs. It is beyond the scope of this book to explain in detail where, when, how, or why you would use any of them. Of course, this does not mean that you must be a financial analyst to the use the Discount Commercial Paper program or a mathematician to use the Poisson Distribution program. There are sample runs and practice problems for each program. If you understand the applications well enough to know that the program may fit your needs, but you would like more information, you will find suggestions for further reading with most programs.

This book's secondary purpose is to show by example the wide range of subjects that lend themselves to computerization. All too often, computer users who have cut their teeth on entertainment computing have trouble coming up with ideas for practical computing. So, even if you don't see a program in this book that is exactly what you need, you may find it easier to invent your own practical applications after studying some of these.

As you look through the programs in this book, you may discover that you can use parts of the programs or some of the programming techniques in your own work. For example, this book includes functions for manipulating dates and character strings which can be amalgamated into other programs. You may even use an entire program as a component part of you own larger, more complex program. Some of these programs share code with programs in the book *Some Practical Pascal Programs,* also published by Osborne/McGraw-Hill.

Organization

Each program is accompanied by a discussion of the subject matter, the program information, the content and form of the output, and Program Notes. This material is followed by examples of how the program might be used in more or less real-life situations. The point of these examples is to help you imagine potential uses for the program. The examples demonstrate as many of the program features as they can in a moderate-sized problem. The sample run is next, showing the dialogue between the computer and the user when the program is used to solve the problem posed in the example. Compare the user's inputs and the computer's outputs in the sample run with the problem stated in the example. You should understand how you would use the program to solve a similar problem.

The text of the Pascal program comes next. To save typing, and to accommodate differences in the way that different implementations of Pascal receive interactive input, some procedures, functions, and type declarations (called Include files) used by several of the programs are printed only once in Appendices A and B. If you type them into a file you have created for this purpose, you can simply copy them into each program where they are needed, using your system text editor. Alternatively, most Pascal implementations allow you to tell the compiler where to find files that will be "Included" at specific places in your programs. See Appendix A for more information on Include files.

Lastly, we list references for most programs. Investigate these books and articles if you wish to read more about the subject matter of the program.

Pascal Compatibility

These programs have been written in a very conservative Pascal, acceptable to any implementation. To remove the worst potential problem—interactive input—all input to these programs is done through routines to be copied from Appendix B. This appendix contains suitable routines for the most common solutions to the problem of interactive input in Pascal. One set of routines is suitable for use with UCSD Pascal, including Apple Pascal. Another set of routines will work for any implementation using the Lazy-IO convention, in which characters are not read until the first time the program attempts to inspect them. By merely selecting the appropriate routine to type in, all the programs should run without modification on your system. See Appendix A for more information on the different implementations of Pascal.

None of these programs requires a mass storage device (disk or tape) for storing data. Thus, the widely varying methods for accessing data files in Pascal are not a problem. Of course, you will want to store the programs themselves on a tape or disk once you have typed them in. This is a fairly straightforward procedure that should be described in the manual for your computer system.

How to Use These Programs

Follow these procedures when you want to use one of the programs in this book:

1. Read the program write-up and familiarize yourself with how the program works. Consult the suggested reference material if you need a better understanding of the subject matter that the program addresses. Be sure that the program does what you need it to do before going any further.
2. Type the program listing into your computer. Make a note of any Include files you need to have available, and if you are directing your compiler to find them in separate files, be sure you use the syntax specified by your compiler. (See Appendix A for more information on this.) Look for any lines containing comments that you should know about. If a comment line says to omit the line unless you have a particular Pascal implementation (such as Apple Pascal) and you are not using that implementation, do not type in the line.
3. Check your program listing carefully for correctness. Not all typing errors are caught by the compiler.
4. Save the program on tape or disk. Do it *now,* before you run the program. If you do, you can retrieve your work if anything happens while the program is running. Remember, unless your editor keeps an audit trail or has some other unusual protective feature, you are always in danger of losing all the work until you save the program. With longer programs you may wish to save it after typing it in.
5. Determine whether the current program requires any Include files that have not been typed in for another program. Type in these new Include files and save them in separate files. If you have directed your compiler to Include from those files, be sure you give it the correct file names. If your text editor copies the Include files into your main program file, you may still want to keep separate copies of the Include files for each use in later programs.
6. Compile the program. If the compiler indicates any errors, double-check your typing and check that you used the correct Include files.
7. Run the example exactly as shown in the sample run. If you have done everything right up to this point, the results should be similar to those published in the example. Your answers will differ slightly from those in the book if your computer has a different level of internal numerical precision than ours.
   NOTE: Most versions of UCSD Pascal have only six digits of precision for real numbers. This may lead to slight inaccuracies (in the cents columns) in financial calculations. If absolute accuracy is required, you might consider learning how to use the non-standard Long Integer feature and keeping monetary amounts in cents or mills.
8. If your answers differ markedly from ours, or your program does not run at all (that is, you get some sort of error message), it is time for some detective work. First, double-check your listing. It may be useful to count the number of lines, just to make sure that you have not duplicated or eliminated any lines, a common error. If you are still having trouble, read Appendix A for information on potential problems with implementation of Pascal.

9. By now your program should be running correctly. If not, have someone else look at your program. Often another pair of eyes can see something that you repeatedly miss. Try putting the program aside for a while and come back to it after a short break. Errors you didn't see before may be obvious later.

Acknowledgments

These programs are Pascal versions of BASIC programs originally published in *Some Common BASIC Programs* (Osborne/McGraw-Hill, 1977). Gregory Davidson converted 41 of the original BASIC programs. The remaining 35 were written by Osborne/McGraw-Hill staff programmers Brad Hellman, Brian Williamson, and Vicki Marney-Petix, and the book was edited by Vicki Marney-Petix. The original *Some Common BASIC Programs* was edited by Lon Poole and Mary Borchers.

Other Osborne/McGraw-Hill Publications

An Introduction to Microcomputers: Volume 0—The Beginner's Book, 3rd Edition
An Introduction to Microcomputers: Volume 1—Basic Concepts, 2nd Edition
An Introduction to Microcomputers: Volume 3—Some Real Support Devices
Osborne 4 & 8-Bit Microprocessor Handbook
Osborne 16-Bit Microprocessor Handbook
8089 I/O Processor Handbook
CRT Controller Handbook
68000 Microprocessor Handbook
8080A/8085 Assembly Language Programming
6800 Assembly Language Programming
Z80 Assembly Language Programming
6502 Assembly Language Programming
Z8000 Assembly Language Programming
6809 Assembly Language Programming
Running Wild—The Next Industrial Revolution
The 8086 Book
PET® and the IEEE 488 Bus (GPIB), 2nd Edition
PET™/CBM™ Personal Computer Guide, 2nd Edition
CBM™ Professional Computer Guide
Business System Buyer's Guide
Osborne CP/M® User Guide
Apple II® User's Guide
Microprocessors for Measurement and Control
Some Common BASIC Programs
Some Common BASIC Programs—PET™/CBM™ Edition
Some Common BASIC Programs—Atari® Edition
Some Common BASIC Programs—TRS-80™ Level II® Edition
Some Common BASIC Programs—Apple II Edition
Practical BASIC Programs
Practical BASIC Programs—TRS-80™ Level II Edition
Practical BASIC Programs—Apple II® Edition
Practical BASIC Programs—IBM® Personal Computer Edition
Practical Pascal Programs
Payroll with Cost Accounting
Accounts Payable and Accounts Receivable
General Ledger
CBASIC™ User Guide
Science and Engineering Programs—Apple II® Edition
Interfacing to S-100/IEEE 696 Microcomputer
A User Guide to the UNIX™ System
PET™ Fun and Games
Trade Secrets: How to Protect Your Ideas and Assets
Assembly Language Programming for the Apple II®
VisiCalc®: Home and Office Companion
Discover FORTH
6502 Assembly Language Subroutines
Your ATARI® Computer
CBM™ Professional Computer Guide

Contents

1
Future Value of an Investment

This program calculates the future value of an investment which earns interest. You must know the amount of the initial investment, the nominal interest rate, the number of compounding periods per year, and the amount of time in months and years that the money is invested.

A financial situation may act like a compound interest calculation even when it is called something else. A steady rise in property values is a good example. Please note that if there is only one compounding period per year, you must specify a term in whole years to obtain an accurate answer.

Assuming there are no additional deposits and no withdrawals, the future value is based on the formula

$$T = P(1 + i/N)^{N \cdot Y}$$

where: T = total value after Y years (future value)
P = initial investment
i = nominal interest rate
N = number of compounding periods per year
Y = number of years

Examples:

Carl makes an investment of $6800.00 at 9.5%. If interest is compounded quarterly, what will be the value of Carl's investment in 10 years and 6 months?

Valerie purchases a piece of property for $16,050.00. Property values are rising at an average annual rate of 7%. What may Valerie expect her property to be worth in five years?

Run:

```
Future value of an investment

Initial investment: $6800
Nominal interest rate (%) 9.5
Number of compounding periods per year: 4
Number of whole years: 10
Number of periods past last whole year: 0
Future value = $ 17388.6

Would you like another run? (y/n) y

Initial investment: $16050
Nominal interest rate (%) 7
Number of compounding periods per year: 1
Number of whole years: 5.5
Number of periods past last whole year: 0
Future value = $ 22511.0

Would you like another run? (y/n) n
```

Program Listing:

```
program FutureVal(input, output);
var
  NumPeriods, XtraPeriods, NumYears: integer;
  investment, percent, rate: real;

{$I IntRaise}
{$I NotAgain}

begin { main }
  writeln('Future value of an investment');
  repeat
    writeln;
    write('Initial investment: $');
    readln(investment);
    write('Nominal interest rate (%) ');
    readln(percent);
    write('Number of compounding periods per year: ');
    readln(NumPeriods);
    write('Number of whole years: ');
    readln(NumYears);
    write('Number of periods past last whole year: ');
    readln(XtraPeriods);
    rate := percent / NumPeriods / 100;
    writeln('Future value = $', investment
        * IntRaise(1 + rate, NumPeriods * NumYears + XtraPeriods):9:2);
    writeln
  until NotAgain
end.
```

2
Future Value of Regular Deposits (Annuity)

This program calculates a future value when regular deposits are made. You must provide the amount of each deposit, the number of deposits per year, the amount of time the future value is calculated for, and the nominal interest rate.

Assuming that interest is compounded with each deposit, the calculation is based on the formula

$$T = R \cdot \left(\frac{(1 + i/n)^{N \cdot Y} - 1}{i/N}\right)$$

where: T = total value after Y years (future value)
R = amount of regular deposits
N = number of deposits per year
Y = number of years
i = nominal interest rate

Examples:

Michel makes annuity payments of $175.00. The interest is 5.5%. What amount will Michel have accumulated in 15 years?

Each month, Tanya transfers $50 from her checking account to a Christmas Club savings account with 5% interest. How much can Tanya expect to have saved at the end of the year?

Run:

```
Future Value of Regular Deposits (Annuity)

Amount of regular deposits: $50
Nominal interest rate: (%) 5
Number of deposits per year: 12
Number of years: 1
Future value = $    613.95

Would you like another run? (y/n) y

Amount of regular deposits: $175
Nominal interest rate: (%) 5.5
Number of deposits per year: 1
Number of years: 15
Future value = $   3921.51

Would you like another run? (y/n) n
```

Program Listing:

```
program annuity(input, output);
var
  DepsPerYear, NumYears: integer;
  AmtDep, percent, RatePerDep: real;

{$I IntRaise}
{$I NotAgain}

begin { main }
  writeln('Future Value of Regular Deposits (Annuity)');
  repeat
    writeln;
    write('Amount of regular deposits: $');
    readln(AmtDep);
    write('Nominal interest rate: (%) ');
    readln(percent);
    write('Number of deposits per year: ');
    readln(DepsPerYear);
    write('Number of years: ');
    readln(NumYears);
    RatePerDep := percent / DepsPerYear / 100;
    writeln('Future value = $', AmtDep
            * (IntRaise(1 + RatePerDep, DepsPerYear * NumYears) - 1
            / RatePerDep:9:2);
    writeln
  until NotAgain
end.
```

3
Regular Deposits

This program calculates the regular deposit amount required to provide a stated future value in a specified time period. All deposits are equal, and the number of deposits per year must be at least one. You must know the future value, the nominal interest rate, the number of deposits per year, and the term in years and months.

You must be careful to input only terms that are "reasonable" for the specified problem. For example, if deposits are quarterly and you specify a term of two years and two months, the answer will be prorated on the basis of the next quarterly deposit. But, financial institutions do not prorate. A term of two years and two months would be reasonable if deposits were monthly, however.

The calculation for regular deposits is based on the formula

$$R = T\left(\frac{i/N}{(1 + i/N)^{N \cdot Y} - 1}\right)$$

where: R = amount of regular deposit
T = future value
i = nominal interest rate
N = number of deposits per year
Y = number of years

Examples:

Karen would like to have $1000 in her savings account at the end of the year. How much must she deposit each month to reach her goal, if she is receiving 8% interest on her savings?

Roman has opened an Individual Retirement Account (IRA) which he hopes will have $15,000 in 10 years and 3 months. The nominal interest rate for IRAs at his bank is 12.5% and he will make quarterly deposits. How large must each deposit be?

Run:

```
Regular Deposits

Desired future value: $1000
Nominal interest rate: (%) 8
Number of deposits per year? 12
Number of whole years: 1
Number of additional months(0-12):'0

Regular deposits = $  80.32

Would you like another run? (y/n) y

Desired future value: $15000
Nominal interest rate: (%) 12.5
Number of deposits per year? 4
Number of whole years: 10
Number of additional months(0-12): 3

Regular deposits = $ 185.19

Would you like another run? (y/n) n
```

Program Listing:

```
program RegularDeposits(input, output);
uses transcendentals;{omit this line if not Apple Pascal}
var
  NumYears, NumMonths, DepsPerYear: integer;
  value, percent, RatePerDep, TotalTime: real;

{$I RealRaise}
{$I ReadInt}
{$I NotAgain}

begin { main }
  writeln('Regular Deposits');
  repeat
    writeln;
    write('Desired future value: $');
    readln(value);
    write('Nominal interest rate: (%) ');
    readln(percent);
    repeat
      write('Number of deposits per year? ')
    until ReadInt(DepsPerYear, 1, maxint);
    repeat
      write('Number of whole years: ')
    until ReadInt(NumYears, 1, maxint);
    repeat
      write('Number of additional months(0-12): ')
    until ReadInt(NumMonths,0,12);
    RatePerDep := percent / DepsPerYear / 100;
    TotalTime:=NumYears+NumMonths/12;
    writeln;
    writeln('Regular deposits = $',
           value * RatePerDep
           /(RealRaise(RatePerDep + 1, DepsPerYear * TotalTime) - 1):7:2);
    writeln
  until NotAgain
end.
```

4
Regular Withdrawals from an Investment

This program calculates the maximum amount that may be withdrawn regularly from an investment over a specified time period, leaving a zero balance in the account. All withdrawals are equal. If less than the maximum amount is withdrawn, a balance will remain in the account at the end of the time period. You must know the amount of the initial investment, the nominal interest rate, the number of withdrawals per year, and the term in years and months.

You must be careful to input only terms that are "reasonable" for the specified problem. For example, if withdrawals are quarterly and you specify a term of two years and two months, the answer will be prorated on the basis of the next quarterly withdrawal. But, financial institutions do not prorate. A term of two years and two months would be perfectly reasonable if withdrawals were monthly, however.

The maximum amount of the withdrawals is calculated using the formula

$$R = P\left(\frac{i/N}{(1 + i/N)^{N \cdot Y} - 1} + \frac{i}{N}\right)$$

where: R = amount of regular withdrawal
P = initial investment
i = nominal interest rate
N = number of withdrawals per year
Y = number of years

Examples:

The twins, David and Daniel, each received legacies of $8000 from their aunt's estate. They invested their money with a nominal interest rate of 9.5%. David wants to make regular monthly withdrawals for ten years. What is the maximum he can withdraw each month?

Daniel wants to make weekly withdrawals from his account for ten years and six months. What is the maximum amount he can withdraw each week?

Run:

```
Regular Withdrawals from an Investment

Initial investment: $8000
Nominal interest rate: (%) 9.5
Number of withdrawals per year: 12
Number of whole years: 10
Number of additional months(0-12): 0
Amount of each withdrawal = $ 103.52

Would you like another run? (y/n) n
```

Program Listing:

```
program RegularWithdrawals(input, output);
uses transcend;  { Omit this line if not using Apple Pascal }
var
  WithsPerYear, NumYears, NumMonths: integer;
  invest, percent, RatePerWith, TotalTime: real;
```