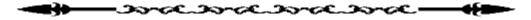
Pascal and C++ Side by Side



Copyright © 1997 By Maria Litvin and Skylight Publishing All rights reserved.

For permission to make multiple copies please contact Maria Litvin, Phillips Academy, Andover, MA 01810 e-mail: mlitvin@andover.edu

You can download an MS Word file PASCPP.DOC for this document.

It contains true side-by-side tables formatted for 8 1/2 by 11 paper (in landscape orientation).

Brief Contents

- 1. Example: BMI Program
- 2. Program Layout, Names, Cosmetics
- 3. Declarations of Constants, Variables, and Arrays
- 4. Procedures and Functions
- 5. Arithmetic Expressions
- 6. Conditions and if-else Statements
- 7. <u>Iterations</u>
- 8. case / switch
- 9. Input and Output
- 10. Strings
- 11. Sets
- 12. Pointers, References, Dynamic Memory Allocation
- 13. Records / Structures
- 14. Classes
- 15. Bit-Wise Logical Operators

Skylight Publishing Home Page

Acknowledgements

My sincere thanks to Owen Astrachan whose valuable comments helped to improve this text. I am grateful to all the visitors of this web site and to my students who commented on "Pascal and C++ Side by Side."

Contents

- 1. Example: BMI Program
- 2. Program Layout, Names, Cosmetics
 - o Comments
 - o <u>Upper and Lower Case</u>

- o Reserved Words
- o Names
- o Main Program
- o Blocks, Semicolons
- 3. Declarations of Constants, Variables, and Arrays
 - o Built-In Data Types
 - o Enumerated Types
 - o Constants
 - o Variables
 - o Arrays
 - o Type / typedef
 - o sizeof(...) Operator
- 4. Procedures and Functions
 - o Procedures vs. Functions
 - o Placement of Procedures / Functions in the Source Module
 - o Passing Parameters (Arguments) by Reference
- 5. Arithmetic Expressions
 - Assignment and Arithmetic Operators
 - o Compound Arithmetic Operators
 - Explicit Type Conversions / Casts
 - o Built-In / Library Math Functions
- 6. Conditions and if-else Statements
 - o Boolean Variables and Values
 - o Relational Operators
 - o Logical Operators
 - o if-else Statements
 - o Short-Circuit Evaluation
- 7. <u>Iterations</u>
 - o while, for, repeat / do
 - o break and continue
- 8. case / switch
- 9. Input and Output
 - Standard Input / Output
 - o Files
- 10. Strings
- 11. <u>Sets</u>
- 12. Pointers, References, Dynamic Memory Allocation
 - o Pointers, new and dispose / delete
 - o Pointers and Arrays
 - Reference Variables
- 13. Records / Structures
 - Declarations and Member Access
 - o Member Access with Pointers
- 14. Classes

15. Bit-Wise Logical Operators



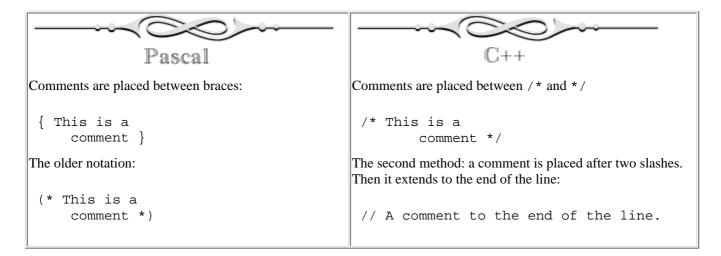
Example: BMI Program

```
Pasca1
{ BMI.PAS }
Program CheckWeight (input, output);
  This program prompts the user for his weight in pounds and
  height in inches, computes the Body Mass Index (BMI) and displays
  "Underweight" if BMI is less than 18, "Normal" if BMI is between
  18 and 25, and "Overweight" if BMI is greater than 25.
  BMI is defined as weight, in kilograms, divided over the squared
  height, in meters.
}
const
    kgInPound = 0.4536;
    metersInInch = 0.0254;
var
    weight,
    height : real;
    BMI
          : integer;
function BodyMassIndex(weight, height : real) : integer;
    { Takes weight in kilograms and height in meters.
      Returns the value of BMI rounded to the nearest integer. }
    var
        bmIndex : real;
        bmIndex := weight / (height * height);
        BodyMassIndex := round(bmIndex);
    end;
begin { main program }
    write ('Enter your height in inches ==> ');
    readln (height);
    write ('Enter your weight in pounds ==> ');
    readln (weight);
    weight := weight * kgInPound;
    height := height * metersInInch;
    BMI := BodyMassIndex(weight, height);
    writeln ('Your BMI = ', BMI);
    if BMI < 18 then
        writeln ('Underweight')
    else if BMI <= 25 then
        writeln ('Normal')
    else
```

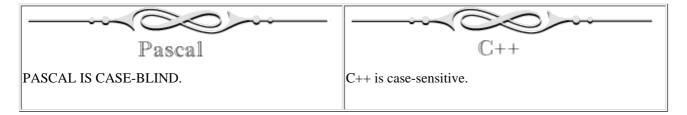
```
writeln ('Overweight');
end.
               \mathbb{C}^{++}
  BMI.CPP
  This program prompts the user for his weight in pounds and
  height in inches, computes the Body Mass Index (BMI) and displays
  "Underweight" if BMI is less than 18, "Normal" if BMI is between
  18 and 25, and "Overweight" if BMI is greater than 25.
 BMI is defined as weight, in kilograms, divided over the squared
 height, in meters.
#include <iostream.h>
int BodyMassIndex(double weight, double height)
// Takes weight in kilograms and height in meters.
// Returns the value of BMI rounded to the nearest integer.
{
    double bmIndex;
    bmIndex = weight / (height * height);
    return int(bmIndex + .5); // round to the nearest integer
}
int main()
{
    const double kgInPound = 0.4536, metersInInch = 0.0254;
    double weight, height;
    int BMI;
    cout << "Enter your height in inches ==> ";
    cin >> height;
    cout << "Enter your weight in pounds ==> ";
    cin >> weight;
    weight = weight * kgInPound;
                                     // or: weight *= kgInPounds;
    height = height * metersInInch; // or: height *= metersInInch;
    BMI = BodyMassIndex(weight, height);
    cout << "Your BMI = " << BMI << endl;</pre>
    if (BMI < 18)
        cout << "Underweight" << endl;</pre>
    else if (BMI <= 25)
        cout << "Normal" << endl;</pre>
        cout << "Overweight" << endl;</pre>
    return 0;
}
```

Program Layout, Names, Cosmetics

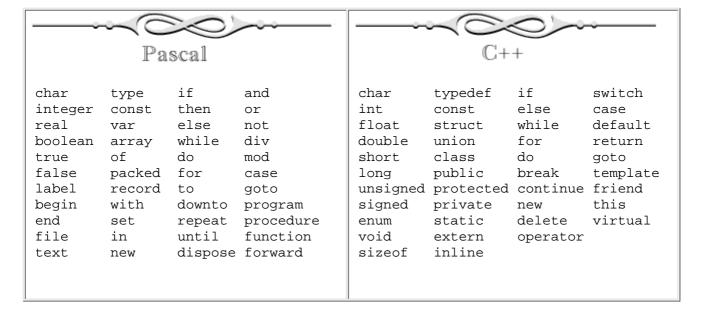
Comments



Upper and Lower Case



Reserved Words (A Partial List)



Names



Names can use letters and digits but must begin with a letter, e.g.:

amount, x1, str3a



Names can use letters, digits and the underscore character, but must begin with a letter or the underscore, e.g.:

amount, x1_, _str3a

Main Program



program MyProg(input, output);
begin

writeln ('Hello, World!');
end.



#include <iostream.h>

int main()
{
 cout << "Hello, World!" << endl;
 return 0;
}</pre>

The "main program" is implemented as the function main(). The return value 0 indicates to the operating system that the program finished successfully.

The *include* (or *header*) file iostream.h contains the definitions of the standard input and output streams cin and cout.

Blocks, Semicolons



A compound statement is placed between begin and end:

begin
 <statement1> ;
 <statement2>
end;

Semicolon is optional before end and is usually required after end, unless followed by another end.



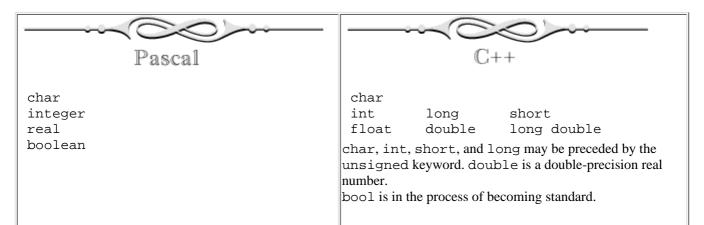
A compound statement is placed between braces:

{
 <statement1> ;
 <statement2> ;
}

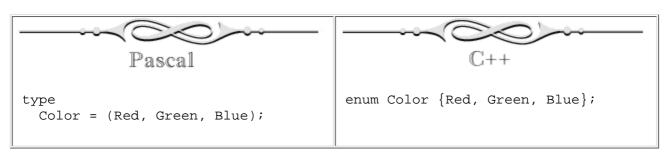
Semicolon is required before the closing brace, and usually omitted after it.

Declarations of Constants, Variables, and Arrays

Built-In Data Types



Enumerated Types



Constants



All declarations of constants in the main program or in a procedure or a function are grouped together under the keyword const:

```
const
  Pi = 3.14;
  Rate = 0.05;
     { .05 not allowed }
  Hour = 3600;
  Dollar = '$';
  Greeting = 'Hello, World!';
```

There are no "escape" characters. Two single quotes in a row in a literal string represent one single quote character:

```
writeln ('Let''s have fun!');
```



Declarations of constants are the same as declarations of variables with initialization, but they are preceded by the keyword const:

```
const double Pi = 3.14,
    Rate = .05; // or 0.05;
const int Hour = 3600;
const char Dollar = '$';
const char Greeting[] =
    "Hello, World!";

// Also allowed:
const double R = 5., Pi = 3.14,
    Area = Pi * R * R;
```

C++ recognizes so-called "escape characters" for special char constants. These are written as a backslash (which serves as the "escape" character) followed by some mnemonic char. For example:

```
'\n'
       newline
 '\'' single quote
 '\"'
       double quote
 backslash
 '\a' alarm (bell)
 '\t'
      tab
 '\r' carriage return
 '\f' form feed
etc.
Character constants with escape chars are used the same
way as regular char constants. For example:
 const char newline = '\n';
 cout << "Hello, World\n";</pre>
```

Variables



All declarations of variables in the main program or in a procedure or a function are grouped together under the keyword var:

```
SomeProcedure (...);

...
var
   r : real;
   i, j : integer;
   star : char;
   match : boolean;
   ...
begin
   ...
end;
```

No initialization is allowed in declarations.



Declarations of variables (or constants) may be placed more or less anywhere in the code, before they are used. Beginners are advised to place them at the top of main() or at the top of a function to avoid complications with the scope rules. Global variables, declared outside any function (and outside main()), are allowed, but should be avoided. Values of variables may be initialized to constants or previously defined variables or expressions:

```
SomeFunction (...)
{
   double r = 5.;
   int i = 0, j = i+1;
   char star = '*';
   ...
}
```

Arrays



var
 str : packed array [1..80] of char;
 grid : array [1..32, 1..25]
 of integer;

The packed keyword is recommended for an array of characters to save space. The range of subscripts can start from any number, but usually starts from 1. Here str[1] refers to the first element of the array str. Pascal compilers normally report an error if a subscript value is out of range.

char str[80];
int grid[32][25];

The subscript for the first element of the array is 0. Here str[0] refers to the first element of the array str and str[79] to the last element. C++ compilers do not verify that a subscript value is within the legal range.

Arrays can be initialized in declarations. For example:

```
int fiboNums[5] = {1,1,2,3,5};
char phrase[80] =
   "Hello, World!";
```

Type / typedef



The type keyword is used to define enumerated and *subrange* types, array types, and records:



The typedef keyword is used to define aliases for built-in (and, if desired, userdefined) types:

```
typedef unsigned char BYTE;
// e.g. BYTE pixel;

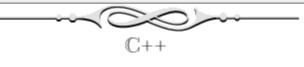
typedef double MONEY;
// e.g. MONEY price = 9.95;

typedef int BOARD[8][8];
// e.g. BOARD board;
```

sizeof(...) Operator



No such thing.



Returns the size in bytes of a constant, a variable, or a data type on your computer system. For example, sizeof(char) returns 1, sizeof(int) may be 2 or 4.

Procedures and Functions

Procedures vs. Functions



Procedures and functions take arguments of specified types. Procedures do not explicitly return a value. Functions return a value of the specified type.

```
procedure DoSomething
    (n : integer; ch : char);
  begin
    . . .
  end;
function ComputeSomething
    (m, n : integer) : real;
  begin
    . . .
    ComputeSomething :=
            <expression>;
```

The return value in a function is indicated by using the assignment statement.



There are no procedures, everything is a function. Functions take arguments of specified types and return a value of the specified type. Functions that do not explicitly return a value are designated as void functions.

```
void DoSomething (int n, char ch)
double ComputeSomething
              (int m, int n)
    return <expression>;
}
```

Functions of the type other than void return a value of the specified type. The return value is indicated by using the return statement. A function can have multiple return statements. A void function can have return statements without any value to return.

```
if ( <condition)> )
 return;
```

This is used to quit early and return to the calling statement.

Placement of Procedures / Functions in the Source Module



call to it:

```
program ...
. . .
procedure DoSomething (...);
  begin
    . . .
```



A procedure or a function is usually defined above the first A function must be declared above the first call to it. The function's definition (heading and body) may be placed above the first call, or the function's *prototype* (heading only) is placed above the first call, usually near the top of the source module (or in a header file). A prototype is similar to Pascal's forward declaration: it declares the function's type and arguments:

```
// Function prototype:
```

```
end;
...
begin { main }
...
DoSomething(...);
...
end.
```

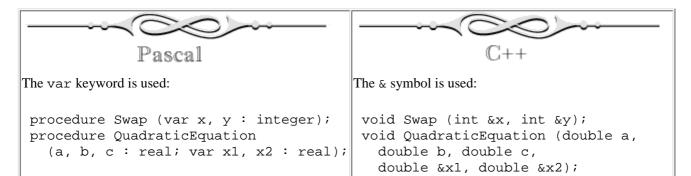
Occasionally the forward keyword is used to define the heading of a procedure or a function and allow the placement of its definition later in the source code.

Nested procedures or functions <u>are allowed</u>. All procedures and functions are nested inside the main program.

```
double MyFunc(int arg1, int arg2);
int main()
{
   double x;
   ...
   x = MyFunc(1999, 3);
   ...
}
...
// Function definition:
double MyFunc(int arg1, int arg2)
{
   ...
}
```

Note: semicolon terminates the prototype but not allowed in the definition header. Nested functions are <u>not allowed</u>.

Passing Parameters (Arguments) by Reference



Arithmetic Expressions

Assignment and Arithmetic Operators

```
Pascal

:= { assignment }

+
-
*
/ { "real" division }
div { "integer" division }
mod { modulo division }
```

Arithmetic operations are allowed only for integer and real operands. div and mod are used only with integer operands. No arithmetic operation are allowed for variables of the char or boolean types.

```
C++

= // assignment
+
-
*
//
% // modulo division
```

Arithmetic operations are allowed for all built-in types, including char, although % makes sense only for integral types (char, int, long, short, etc.). char operands use the actual binary value stored in that byte and have a

The result of an arithmetic operation has integer type when both operands have integer type and real when at least one of the operands is real. The "real" division / is an exception: the result is always a real value, even if operands are integers.

The result of div is the quotient truncated to an integer (in the direction of 0). Examples:

```
var
  x : real;
  n : integer;
...
  x := 2 / 3;
  { x gets the value of 0.66.. }

n := 2 div 3;
  { n gets the value of 0 }
```

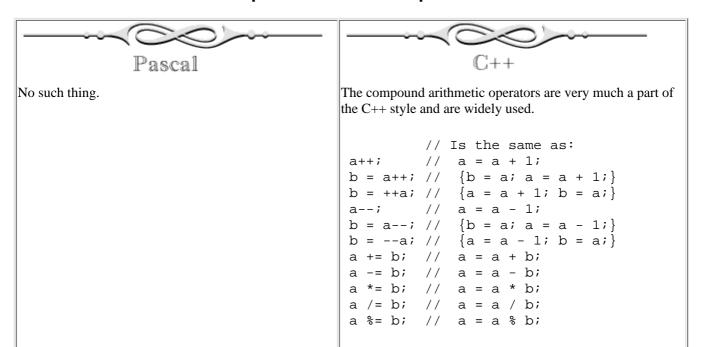
range from -127 to 127. They are first automatically converted to int in arithmetic operations.

The intermediate type of the result is always the same as the type of the operands. If the operands have different types, the "shorter" operand is first *promoted* to the type of the "longer" operand (e.g. int may be promoted to long; or long to double). Examples:

```
double x;
...
    x = 2. / 3;
    // x gets the value of 0.66..

x = 2 / 3;
    // x gets the value of 0
```

Compound Arithmetic Operators



Explicit Type Conversions / Casts



Assignment automatically converts an integer value into a real.

Built-in functions convert real to integer and char to integer:

```
var
  x : real;
  n : integer;
  ch : char;
n := round(x)
{ rounds x to an integer }
n := trunc(x)
{ truncates x to an integer }
n := ord(ch)
{ converts ch into its
  integer ASCII code }
ch := chr(n)
 { converts n into a char
  with ASCII code n }
ch := succ(ch)
{ returns the ASCII char
  that follows ch }
ch := pred(ch)
  { returns the ASCII char
    that precedes ch }
Example:
procedure ToUpper(var ch : char);
  begin
    ch := chr(ord(ch)
        + ord('A') - ord('a'));
   end;
```

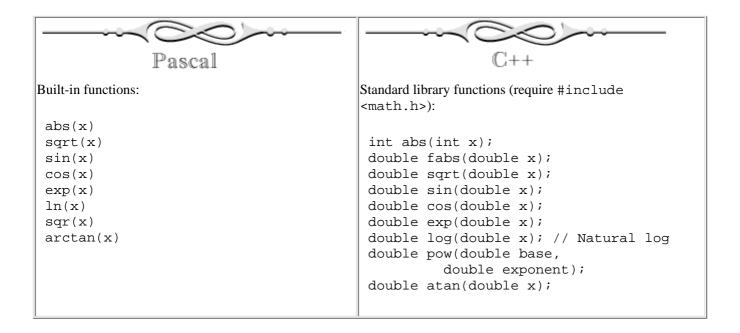


Assignment automatically converts the right-side value into the type of the left-side variable. A compiler warning may be generated if a "longer" type is implicitly converted into a "shorter" type.

A cast operator is provided (and recommended) for explicit type conversions. For example:

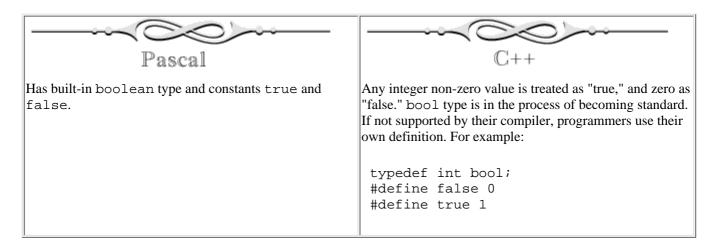
```
int n, p, q;
double x;
char ch;
x = double(p) / double(q);
// sets x to the actual quotient p/q.
n = int(x);
// truncates x to an integer
n = int(ch);
// converts ch to its
// ASCII code value
ch = char(n);
 // converts n to a char
// with ASCII code n
ch = ch + 1;
// sets ch to the next
// ASCII char
ch = ch - 1;
// sets ch to the
// previous ASCII char
Example:
void ToUpper(char ch)
  ch += 'A' - 'a';
```

Built-In / Library Math Functions

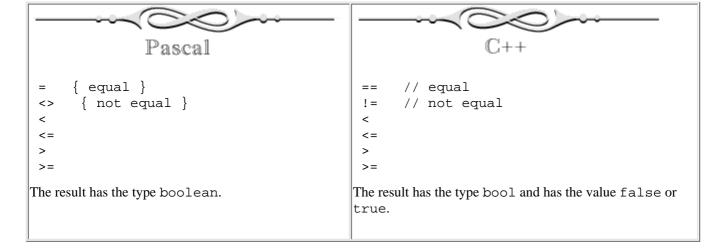


Conditions and if-else Statements

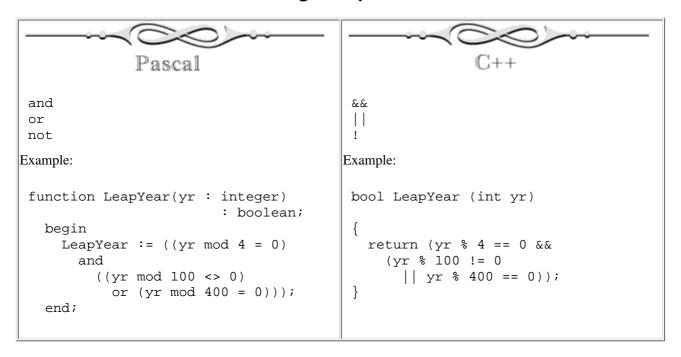
Boolean Variables and Values



Relational Operators



Logical Operators



if-else Statements

```
Pascal
if <condition> then
                                          if ( <condition> )
  <statement>;
                                            <statement>;
if <condition> then
                                          if ( <condition> )
 <statement1> {semicolon not allowed!}
                                            <statement1>; // semicolon required!
else
                                          else
 <statement2>;
                                            <statement2>;
if <condition> then begin
                                          if ( <condition> ) {
 <statement11>;
                                            <statement11>;
 <statement12>;
                                            <statement12>;
                                          }
end
else begin
                                          else {
 <statement21>;
                                            <statement21>;
  <statement22>;
                                            <statement22>;
end;
```

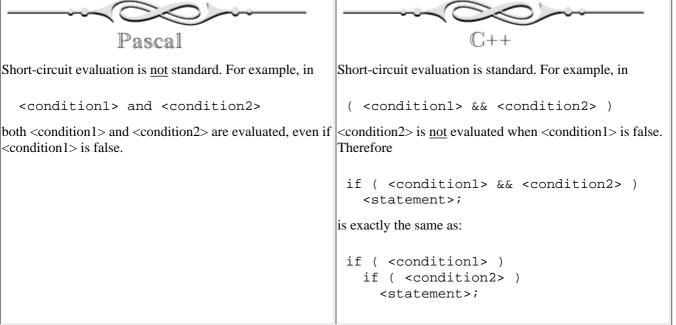
Short-Circuit Evaluation



Short-circuit evaluation is <u>not</u> standard. For example, in

<condition1> and <condition2>

<condition1> is false.



Iterations

while, for, repeat / do

```
Pascal
   { while }
while <condition> do begin
   <statement>;
end;
 { for }
for i := n1 to n2 do begin
           { the step is 1 }
end;
for i := n2 downto n1 do begin
            {the step is -1 }
for ch := ltr1 to ltr2 do
   { repeat - until }
repeat
  <statement>;
until <condition>;
repeat-until keeps iterating as long as <condition>
```

```
C++
// while
while ( <condition> ) {
 <statement>;
// for
for ( <initialization>;
           <condition>;
               <increment> ) {
}
// <initialization>,
    <condition> and
//
//
     <increment> are
//
    arbitrary statements.
//
     For example, a common idiom,
//
     similar to Pascal's
       for i := 0 to n-1 do
//
for (i = 0; i < n;
                     i++)
 . . .
// do - while
```

remains false; quits when condition becomes true.

```
do {
     <statement>;
     ...
} while ( <condition>);
```

do-while keeps iterating as long as <condition> remains true; quits when condition becomes false.

break and continue



No such thing.



break is used within a loop to quit early. continue quits the current iteration and sends control to the new iteration.

Example:

```
for (i = 0; i < n; i++) {
  if (a[i] < 0) break;
    // Quit the for loop
  if (a[i] == 0) continue;
    // Continue with the next i

    // If we get here, a[i] is >0
    product *= a[i];
    ...
}
```

case / switch

```
switch ( <expression> ) {
  case <constl>:
        <statementl>; break;
        ...
  case <constN>:
        <statementN>; break;

  default: // optional
        <dfltstatement>; break;
}
Examples:

switch (ch) {
  case 'A':
        Add();
        break;
```

```
case 'D':
case d of
                                             Delete();
  1,2: ...;
                                             break;
  99: ...;
100: ...;
                                            case 'M':
                                             Modify();
end;
                                             break;
                                            case 'Q':
case age >= 65 of
                                             break; // Do nothing
                                          }
 true: ...;
 false: ...;
                                          switch (d) {
end;
                                            case 1:
                                            case 2:
                                              . . . ;
                                              break;
                                            case 99:
                                              . . . ;
                                              break;
                                            case 100:
                                              break;
                                          }
                                          switch (age >= 65) {
                                            case true: ...;
                                            case false: ...;
```

Input and Output

Standard Input / Output

```
Pascal

write (x, y, ...);
writeln (x, y);
writeln;

read (x, y, ...);
readln (x, y);
readln;

write (x : width : decimals);

writeln ('$', amt : 6 : 2); { e.g. $ 19.00 }

To read a line of text:

var
   str : packed array [1..80] of char;
   i : integer;
```

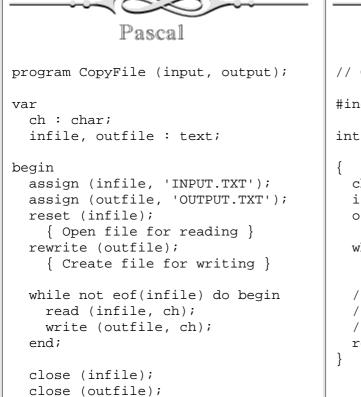
```
{ Read to the end of the line,
   but at most 80 chars: }
i := 1;
while (not eoln) and (i <= 80) do begin
   read(str[i]);
   i := i + 1;
end;

{ Throw away the rest of the line, if any: }
readln;
...</pre>
```

```
#include <iostream.h>
 cout << x << ' ' << y << ...;
 cout << x << ' ' << y << endl;
 cout << endl;
 cin >> x >> y >> ...;
 cin >> x >> y; cin.ignore(80, '\n');
 cin.ignore(80, '\n');
 #include <iostream.h>
 #include <iomanip.h>
  // defines the so-called manipulators:
   //
        setw(...), setprecision(...), etc.
 cout << setprecision(decimals) << setw(width) << x;</pre>
cout.setf(ios::fixed | ios::showpoint);
cout << setprecision(2);</pre>
cout << '$' << setw(6) << amt << endl;  // e.g. $ 19.00</pre>
setf and setprecision stay in effect until changed; setw only for one output item.
To read a line of text:
  char str[81];
   cin.getline(str, 81);
   // Reads to the end of the line or
   // until you get 80 chars, whichever
   // happens first.
   // Appends a null char at the end.
   cin.get(str, 81).ignore(1000, '\n');
   // Reads to the end of the line,
   // but at most 80 chars. Appends null.
   // Throws away remaining chars on the line,
   // if any.
```

end.

Files



```
#include <fstream.h>
int main()

{
   char ch;
   ifstream infile("INPUT.TXT");
   ofstream outfile("OUTPUT.TXT");

   while (infile.get(ch))
     outfile.put(ch);

// Files are closed automatically
   // when infile, outfile variables
   // go out of scope.
   return 0;
}
```

Strings



No such thing in standard Pascal, but many environments provide a String type and operators that handle strings.



Many standard library functions are provided for handling *null-terminated* strings. Null-terminated strings use a null ('\0', zero value) character to mark the end of the string. Literal strings (e.g. "Hello") are nullterminated, so the actual number of bytes required for storage is the number of characters plus one:

```
char hello[6] = "Hello";
```

A String class is provided in many environments, but it is not completely standardized, yet.

Sets



Pasca1

```
if ch in ['0'..'9'] then
   writeln (ch, ' is a digit.');
 if ch in ['A'..'Z', 'a'..'z']
  then
    writeln (ch, ' is a letter.');
In general:
 type
   <settypename> = set of <sometype>;
   setX : <settypename>;
   x : <sometype>;
   setX := [<value1>,
     <value2>...<value3>, ...];
   if (x in setX)
```

Compilers may limit the size of sets to the range of char values, usually 256.



No such thing in standard C++. A Set class is provided in many environments.

Pointers, References, Dynamic Memory Allocation

Pointers, new and dispose / delete

```
Pasca1
type
  RealArray = array [1..100] of real;
var
  i : integer;
  pi1, pi2 : ^integer; { pointers to integer }
  pa : ^RealArray; { pointer to an array of reals }
begin
 i := 99;
  new (pi1); { allocates one integer }
  if pil = nil then
   writeln ('Memory allocation error');
  pi1^ := i;
  pi2 := pi1;
  writeln (pi2^);
                    { output: 99 }
  dispose (pil);
  new (pa); { allocate an array of RealArray type }
```

```
pa^[1] := 1.23;
pa^[2] := pa^[1];
writeln (pa^[2]); { output: 1.23 }
dispose (pa);
end.
```

```
int main()
  int i, *pi1, *pi2; // pi1, pi2 are pointers to int.
  double *pa;
                     // pointer to a double
  i = 99;
  pil = new int;
  if (!pi1)
                 // or: if (pi == 0)
   cout << "Memory allocation error" << endl;</pre>
  *pi1 = i;
  pi2 = pi1;
  cout << *pi2 << endl; // output: 99</pre>
  delete pil;
  pa = new double[100]; // allocate an array of 100 doubles
  pa[0] = 1.23;
  pa[1] = pa[0];
  cout << pa[1] << endl; // output: 1.23</pre>
  delete [] pa;
  return 0;
```

In C++ there is the "address of" operator & and a pointer can be set to the address of a variable of the same type. For example:

```
int a, *pa = &a; // Pointer pa is set to the address of a.
```

Pointers and Arrays



There is no direct connection between pointers and arrays. Pointers are used primarily for linked lists, trees, and other linked structures.



In C++ there is an intimate connection between arrays and pointers. Array name (without [. . .]) has the pointer type, and this pointer points to the first element of the array. So, given

```
int a[100];
a[0] is the same as *a.
```

[] can be viewed as an operator "subscript", which is applied to a pointer and an integer (subscript). C++ supports "pointer arithmetic" which mimics calculation of

```
subscripts in an array. So, a[i] is the same as *(a+i).
```

In addition to its use with linked lists and other linked structures, the new operator supports allocation of arrays of variable length. For example:

```
int n;
cin >> n; // Enter n

// Allocate an array of n integers:
int *a = new int[n];

a[0] = ...;
```

Reference Variables





No such thing.

Reference variables parallel pointers, but use different syntax:

```
int main()
{
  int i = 1, &ri = i;
  // ri becomes an alias for i
  i = 99;
  cout << ri << endl;
  // Output: 99
  ...
}</pre>
```

Reference variables are mostly used to pass arguments to functions by reference and to return reference values from functions and overloaded operators.

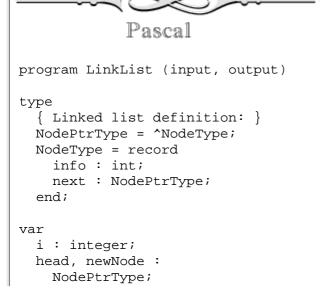
Records / Structures

Declarations and Member Access

```
Pasca1
program (...)
type
 PointType = record
   x, y : real;
  end;
  RectType = record
    upperLeft : PointType;
    lowerRight : PointType;
    color : integer;
  end;
var
 rect : RectType;
begin
  { "Dot" notation: }
  rect.color := 255;
  rect.upperLeft.x := 0.0;
  { "with" statement: }
  with rect do begin
    color := 255;
    { Same as: rect.color := 255; }
   upperLeft.x := 0.0;
    . . .
  end;
end.
```

```
struct Point {
 double x, y;
};
struct Rect {
 Point upperLeft;
 Point lowerRight;
 int color;
};
int main()
 Rect rect;
 rect.color = 255;
 rect.upperLeft.x = 0.0;
 // No direct equivalent of
  // "with"
}
```

Member Access with Pointers



```
// LINKLIST.CPP

// Linked list definition:
struct Node {
  int info;
  Node *next;
};

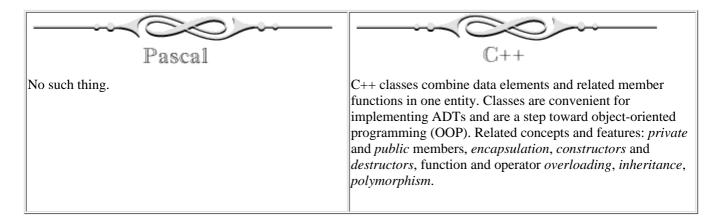
int main()

{
  int i;
  Node *head, *newNode;
  ...
```

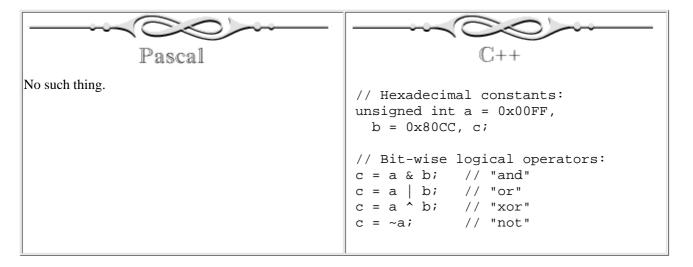
```
begin
...
new (newNode);
newNode^.info := i;
newNode^.next := head;
head = newNode;
...
end.
...
newNode = new Node;
newNode ->info = i;
newNode->next = head;
head = newNode;
...
}

end.
```

Classes



Bit-Wise Logical Operators



Skylight Publishing Home Page

Copyright © 1997 by Maria Litvin and Skylight Publishing. All rights reserved. Skylight Publishing <Revised 1/1/99> support@skylit.com