Module 4

Chapter 4: Making Decisions

SCI 120, BMCC-CUNY

Topics

4.1 Relational Operators

- Are used to compare numeric and char values to determine relative order
- Relational expressions are Boolean (i.e., evaluate to true or false)
- Hierarchy of Relational Operators

Operator	Precedence
> >= < <=	Highest
== !=	Lowest

- The result of a relational expression can be assigned to a variable

```
bool result = (x <= y); / 0 or 1
```

4.2 The if Statement

- Allows statements to be conditionally executed or skipped over

```
if (condition)
{
    statement 1;
    statement 2;
    .
    .
    .
    statement n;
}
```

No semicolon goes here

Semicolons go here

- The block of statements inside the braces is called the body of the if statement. If there is only 1 statement in the body, the { } may be omitted.

`if` Statement Notes

- `if` is a keyword. It must be lowercase
- **(condition)** must be in ( ) and it need not be a comparison
- Do not place **;** after (condition)
- 0 is considered `false` and non-zero is considered `true`.
- Don't confuse **=** (assignment) with **==** (comparison)
- Don't forget the **{ }** around a multi-statement body

Example 4.1

```
// Ex_4_1.cpp
#include <iostream>
#include <string>
using namespace std;

string interweave(long long int n, int f, string s)
{
    if (n == 0) // if statement
        return "0"; // special case
    bool neg = n < 0;
    if (neg)
        n = -n;
    string ans;
    int d = 1;
    while (n > 0)
    {
        ans = static_cast<char>(n % 10 + '0') + ans;
        n /= 10;
        if (d % f == 0 && n != 0) // if statement, relational expression
            ans = s + ans; // prepend new digit
        d += 1;
    }
    if (neg)
        ans = "-" + ans;
    return ans;
}

int main()
{
    cout << "Enter number: ";
    long long n;
    cin >> n;
    cout << interweave(n, 3, ",") << endl;
}
```

Output:

```
Enter number: 5673454564567
5,673,454,564,567
```

```
Enter number: -389345
-389,345
```

4.3 The if/else Statement

- Allows a choice between statements depending on whether (condition) is true or false
- Format:

```
if (condition)
{
   statement set 1;
}
else
{
   statement set 2;
}
```

- If (condition) is true, statement set 1 is executed and statement set 2 is skipped.
- If (condition) is false, statement set 1 is skipped and statement set 2 is executed.

Comparisons with Floating-Point Numbers

- It is difficult to test for equality when working with floating point numbers.
- It is better to use
  - greater-than or less-than tests, or
  - test to see if value is very close to a given value

Example 4.2

```
// Ex_4_2.cpp
#include <iostream>
#include <random>
using namespace std;

random_device rd;
uniform_int_distribution<int> uid(0,1);

string flipCoin()
{
    if (uid(rd))
        return "heads";
    else // not necessary
        return "tails";
}

int main()
{
    cout << "heads or tails? ";
    string choice;
    cin >> choice;

    cout << "\nflip\n";
    auto coinToss = flipCoin();
    cout << "It is " << coinToss << ", ";
    if (choice == coinToss)
        cout << "you win!!\n\n";
    else
        cout << "you lose.\n\n";
}
```

Output:

```
heads or tails? tails

flip
It is tails, you win!!
```

```
heads or tails? tails

flip
It is heads, you lose.
```

if Statement Style Recommendations

- Place each statement; on a separate line after (condition)
- Indent each statement in the body
- When using { and } around the body, put { and } on lines by themselves (MS style)

4.8 Flags

- A variable that signals a condition
- Flags are often booleans

```
bool validMonths = true; // bool flag
        …
if (months < 0)
         validMonths = false;
        …
if (validMonths)
         monthlyPayment = total / months;
```

- Integer variables can be used as flags

```
int errorNo = 0;  // int flag, 0 = no errors
        …
if (denominator == 0)
         errorNo = 1;  // change error flag to 1
        …
return errorNo; // returns flag with error status
```

4.4 The if/else if Statement

- Chain of if/else statements that test several conditions in order until one is found to be true
- Format:

```
if (condition 1){
  statement set 1;
} else if (condition 2){  // else if optional
  statement set 2;
} else if (condition 3){  // else if optional
  statement set 3;
} else {  // trailing else, optional
  statement set 4;
}
```

- The first condition to be true is executed and everything else is skipped.
- There can be as many if else as necessary.
- The trailing else catches everything else.

Example 4.3

```
// Ex_4_3.cpp
#include <iostream>
using namespace std;

int main()
{
    int hour24;
    cout << "Enter hour in 24 format: ";
    cin >> hour24;

    int hour12 = hour24 % 12;
    if (hour12 == 0)
        hour12 = 12;

    string ap;
    if (hour24 > 24)
        ap = "invalid time";
    else if (hour24==24)
        ap = "midnight";
    else if (hour24>12)
        ap = "P.M.";
    else if (hour24==12)
        ap = "noon";
    else if (hour24>0)
        ap = "A.M.";
    else if (hour24==0)
        ap = "midnight";
    else
        ap = "invalid time";

    cout << hour12 << " " << ap << "\n";
}
```

Output:

```
Enter hour in 24 format: 18
6 P.M.
```

```
Enter hour in 24 format: 12
12 noon
```

```
Enter hour in 24 format: 7
7 A.M.
```

```
Enter hour in 24 format: 27
3 invalid time
```

4.5 Menus

- Menu-driven program: program execution controlled by user selecting from a list of choices
- A menu-driven program can be written using if/else if statements

Example 4.4

```
// Ex_4_4.cpp
#include <iostream>
using namespace std;

int main()
{
    cout << "Which one of the following operators means not-equal to?\n";
    cout << "A) 0=\n";
    cout << "B) ~=\n";
    cout << "C) |=\n";
    cout << "D) !=\n";

    cout << "\n? ";
    char c = cin.get();
    cin >> c;
    if (c == 'D')
        cout << "Correct!\n";
    else
        cout << "Not correct\n";
}
```

Output:

```
Which one of the following operators means not-equal to?
A) 0=
B) ~=
C) |=
D) !=

? D
Correct!
```

4.6 Nested if Statements

- An `if` statement that is part of the if or else part of another if statement

```
if (score < 100)
   if (score >= 90)
      grade = 'A';
   else ...  // goes with second if,
             // not first one
```

- Proper indentation aids comprehension
- The `if/else if` is in reality a nested if else statement

```
if (score >= 90)
    cout << "You got an A. Excellent work!\n";
else
    if (score >= 60)
       cout << "You passed!\n";
    else
       cout << "doh!\n";
```

4.7 Logical Operators

- The logical operators !, &&, and ||, take one or two bool operands and output a bool.
- They can be combined with relational operators and boolean variables.
- Example:

```
bool done = false;
if (!done && count < 6)
{
  . . .
}
```

Logic Truth Tables

```
bool a,b;
```

C++ Operators

not

a	!a
0	1
1	0

and

a	b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

or

a	b	a || b
0	0	0
0	1	1
1	0	1
1	1	1

Not in C++

xor

a	b	a Xor b
0	0	0
0	1	1
1	0	1
1	1	0

nand

a	b	a Nand b
0	0	1
0	1	1
1	0	1
1	1	0

Not(And(a,b))

implication

a	b	a Imp b
0	0	1
0	1	1
1	0	0
1	1	1

similar to: if a then b

For binary operators, there are 2^4 output combinations and therefore 16 possible tables.

Logical Precedence

Highest !

&&

Lowest ||

Example:

```
(2 < 3) || (5 > 6) && (7 > 8)
```

is true because && is evaluated before | |

4.10 Checking Numeric Ranges with Logical Operators

- Used to test if a value is within a range

```
if (grade >= 0 && grade <= 100)
   cout << "Valid grade";
```

- Can also test if a value lies outside a range

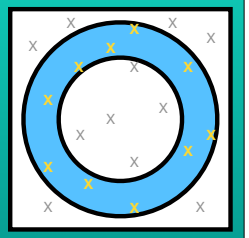
```
if (grade < 0 || grade > 100)
   cout << "Invalid grade";
```

- You **cannot** use mathematical notation

```
if (0 <= grade <= 100) // Doesn't work!
```

Example 4.5

```
#include <iostream>
#include <random>
using namespace std;

bool is_inRing(double x, double y, double r0, double r1)
{
    int a = sqrt(x*x + y*y);
    return a >= r0 && a <= r1; // check range
}

int main()
{
    random_device rd;
    uniform_real_distribution<double> udd(-1.,1.);

    int i = 1;
    while (i <= 10)
    {
        double x = udd(rd);
        double y = udd(rd);
        cout << x << "," << y;

        if (is_inRing(x, y, 0.5, 1.0)) {
            cout << "\tis in ring\n";
        } else {
            cout << "\tis not in ring\n";
        }
        i += 1;
    }
}
```



Output:

```
0.280199,0.303651 is not in ring
0.456645,0.825299 is in ring
0.803828,0.92883 is not in ring
0.839182,0.136814 is in ring
0.425505,0.455482 is in ring
0.281689,0.347704 is not in ring
0.854116,0.1209 is in ring
0.962677,0.712325 is not in ring
0.0391464,0.932826 is in ring
0.00716509,0.42371 is not in ring
```

Short-Circuit Evaluation

- If an expression using the && operator is being evaluated and the subexpression on the left side is false, then there is no reason to evaluate the subexpression on the right side. It is skipped.

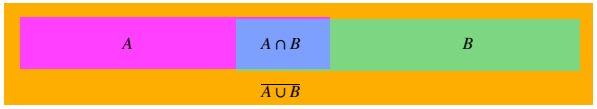
```
if (a >= 0 && pow(a,b) <= 100 )
   cout << a << "\n";
```

- If an expression using the || operator is being evaluated and the subexpression on the left side is true, then there is no reason to evaluate the right side. It is skipped.

```
if (foundHere || checkElswhere())
   cout << "Found\n";
```

Extra: Sets and Logic

Set Theory



Venn diagram

Set operation	Name	Meaning	Logic Equivalent
$A \cap B$	intersection	set of elements in common	a AND b
$A \cup B$	union	set of elements from either	a OR b
$\bar{A}$	compliment	set of elements not in A	NOT a

for $a \in A, b \in B$

Extra: Properties of Logic and Relational Operators

In set theory

- De Morgan's Laws:

$$\overline{A \cup B} = \bar{A} \cap \bar{B} \qquad \overline{A \cap B} = \bar{A} \cup \bar{B}$$

- Involution:

$$\bar{\bar{A}} = A$$

In Boolean Algebra

- a && b = a*b
- a xor b = (a+b)%2
- !a = 1 - a
- a || b = (a+b)%2 + a*b

Identities, given bool a, b;

- false && a = false
- true && a = a
- false || a = a
- true || a = true

All logic expressions can be obtain from a nand !!

- !a = **nand**(a, true)
- a && b = **nand**(**nand**(a, b), true)
- a || b = !(!a && !b) by De Morgan
- xor(a, b) = (a || b) && !(a && b)

Example 4.6

```
// Ex_4_6.cpp
#include <iostream>
#include <functional>
using namespace std;

bool NAND(bool a, bool b)
{
    if (a)
        if (b)
            return 0;
    return 1;
}

bool NOT(bool a)
{
    return NAND(a, true);
}

bool AND(bool a, bool b)
{
    return NAND(NAND(a, b), true);
}

bool OR(bool a, bool b)
{
    return NOT(AND(NOT(a), NOT(b)));
}

bool XOR(bool a, bool b)
{
    return AND(OR(a,b),NOT(AND(a,b)));
}
```

```
void truthTable(function<bool(bool, bool)> re)
{
    cout << "A B Out\n";
    bool A,B;

    A = B = false;
    cout << A << ' ' << B << ' ' << re(A,B) <<'\n';

    A = false; B = true;
    cout << A << ' ' << B << ' ' << re(A,B) <<'\n';

    A = true; B = false;
    cout << A << ' ' << B << ' ' << re(A,B) <<'\n';

    A = B = true;
    cout << A << ' ' << B << ' ' << re(A,B) <<'\n';
}

int main()
{
    cout << "xor:\n";
    truthTable(XOR);
}
```

Output:

```
xor
A B Out
0 0 0
0 1 1
1 0 1
1 1 0
```

4.8 Validating User Input

- Input validation: inspecting input data to determine if it is acceptable
- You want to avoid accepting bad input
- You can perform various tests
  - Range
  - Reasonableness
  - Valid menu choice
  - Zero as a divisor

Example 4.7

```
// Ex_4_7.cpp
#include <iostream>
#include <cmath>
using namespace std;

void sqrt_ratio(float x, float y)
{
    if (y) // Nested if and input validation
    {
        float z = x/y; // z scope in this block only
        if (z >= 0)
            cout << " sqrt(x/y) = " << sqrt(z) << endl;
        else
            cout << " Cannot take the square root of x/y = " << z << endl; ;
    }
    else
        cout << " Cannot divide by " << y << "\n";
}

int main()
{
    cout << "Enter x and y\n";
    float x, y;
    cin >> x >> y;
    sqrt_ratio(x, y);
}
```

Output:

```
Enter x and y
6 7
 sqrt(x/y) = 0.92582
```

```
Enter x and y
7 0
 Cannot divide by 0
```

```
Enter x and y
-8 2
 Cannot take the square root of x/y = -4
```

4.9 More About Blocks and Scope

- The Scope of a variable is the block in which it is defined, from the point of definition to the end of the block
- Variables are usually defined at the beginning of a function
- They may instead be defined close to the place where they are first used

More About Blocks and Scope

- Variables defined inside { } have local or block scope

- When in a block that is nested inside another block, you can define variables with the same name as in the outer block.
  - When the program is executing in the inner block, the outer definition is not available.
  - This generally *not a good idea*. The program may be hard to read or understand.

6.10 Local and Global Variables

- Local variable:
  - A variable defined within a function or a block
  - Function *parameters* are also local variables.
- *Accessibility*:
  - Accessible only within the function or the block.
  - Other functions and blocks can define different variables with the same name without interference
  - When a function is called, local variables in the calling function are not accessible from within the called function

Local Variable Lifetime

- *Lifetime*:
  - A local variable only exists while its defining function is executing
  - Local variables created when a function defines them and are destroyed when the function terminates
  - Data cannot be retained in local variables between calls to the function in which they are defined
- *Initial value*:
  - Local variables must be initialized by the programmer
  - Otherwise, the initial value is undefined

Local and Global Variables

- Global variable: A variable defined outside all functions;
- *Scope*:
  - The scope of a global variable is from its point of definition to the program end
  - It is accessible to all functions within its scope
- *Initialization*: Global variables are set to `0` (or equivalent) if not explicitly initialized
- Can be seen as an easy way to share data between functions but use sparingly! They make the program
  - more error prone
  - harder to understand
  - more difficult to debug

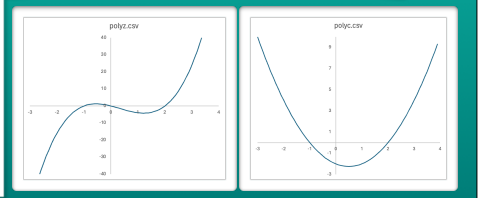
Global Constants

- A global constant is a named constant that can be used by every function in a program
- It is useful if there are unchanging values that are used throughout the program
- They are safer to use than global variables, since the value of a constant cannot be modified during program execution

Example 4.8

```
// Ex_4_8.cpp
#include <iostream>
#include <vector>
#include <fstream>
#include <iomanip>
#include <string>
#include <functional>
using namespace std;

struct Range {
    double x0, x1, dx;
};

double y_scale; // global var.

vector<double> Zeroes; // global var.
double polyZero(double x) {
    double p = 1.;
    int i = 0;
    while(i<Zeroes.size()) { // global var.
       p *= (x-Zeroes[i]); // global var.
       i += 1;
    }
    return p * y_scale; // global var.
}

vector<double> Coeff; // global var.
double polyCoef(double x) {
    double p = 0;
    int i = 0;
    double xi = 1;
    while(i<Coeff.size()) { // global var.
       p += Coeff[i]*xi; // global var.
       xi *= x;  // x^i = x^i*x : next x power
       i += 1;
    }
    return p;
}
```

```
void printFunction(function<double(double)> f,
                   Range r,
                   string fileName) {
    ofstream fout(fileName);
    fout << fixed;
    double x = r.x0;
    while (x <= r.x1) {
        fout << x << "," << f(x) << endl;
        x += r.dx;
    }
}

int main() {
    Range xrange = {-3., 4., 0.1};

    y_scale = 2.0;  // global var.
    Zeroes = {-1., 0, 2.0}; // global var.
    printFunction(polyZero, xrange, "polyz.csv");

    y_scale = 1.0;  // global var.
    Coeff = {-2. , -1., 1.}; // global var.
    printFunction(polyCoef, xrange, "polyc.csv");
}
```



Local and Global Variable Names

- Local variables can have same names as global variables
- When a function contains a local variable that has the same name as a global variable, the global variable is unavailable from within the function. The local definition "hides" or "shadows" the global definition.

4.10 More About Characters and Strings

- You can use relational operators with characters and string objects

  `if (menuChoice == 'A')`, **or** `if (name1 >= name2)`
- Comparing characters is really comparing the ASCII values of characters
- Comparing string objects is comparing the ASCII values of the characters in the strings. Comparison is character-by-character, starting with the first character of each string.
- You cannot compare C-style strings by using relational operators. They are character arrays described later.

Converting Between Strings and Numbers

- The string header provides a few simple functions to converting a numeric string into a number and viceversa.
- To change from a number to a string use **`to_string()`**, e.g.,

```
string text = to_string(4.5); // gives "4.5"
```

- To change from a string to a number use **`sto`***x*`()`, where *x* is either i,f,d,.. for integer, float, double and others (see STL), e.g.,

```
double x = stod(text); // sets x to 4.5
 int n = stoi("365"); // sets n to 365
```

- Alternatively, use the sstring header (covered later)

Testing Characters

- These functions require the cctype header file

FUNCTION	MEANING
`isalpha`	**`true`** if arg. is a letter, **`false`** otherwise
`isalnum`	**`true`** if arg. is a letter or digit, **`false`** otherwise
`isdigit`	**`true`** if arg. is a digit 0-9, **`false`** otherwise
`islower`	**`true`** if arg. is lowercase letter, **`false`** otherwise

Testing Characters

- These functions require the cctype header file

FUNCTION	MEANING
`isprint`	**`true`** if arg. is a printable character, **`false`** otherwise
`ispunct`	**`true`** if arg. is a punctuation character, **`false`** otherwise
`isupper`	**`true`** if arg. is an uppercase letter, **`false`** otherwise
`isspace`	**`true`** if arg. is a whitespace character, **`false`** otherwise

Testing Characters

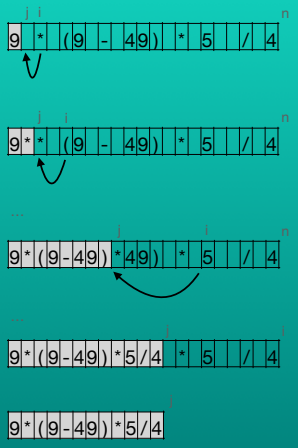
ASCII values	characters	iscntrl	isblank	isspace	isupper	islower	isalpha	isdigit	isxdigit	isalnum	ispunct	isgraph	isprint
0x00 .. 0x08	NUL, (other control codes)	x											
0x09 .	tab ('\t')	x	x	x									
0x0A .. 0x0D	(white-space control codes:	x		x									
0x0E .. 0x1F	(other control codes)	x											
0x20 .	space (' ')		x	x									x
0x21 .. 0x2F	!"#$%&'()*+,-./										x	x	x
0x30 .. 0x39	123456789							x	x	x		x	x
0x3a .. 0x40	:;<=>?@										x	x	x
0x41 .. 0x46	ABCDEF				x		x		x	x		x	x
0x47 .. 0x5A	GHIJKLMNOPQRSTUVWXYZ				x		x			x		x	x
0x5B .. 0x60	[\]^_`										x	x	x
0x61 .. 0x66	abcdef					x	x		x	x		x	x
0x67 .. 0x7A	ghijklmnopqrstuvwxyz					x	x			x		x	x
0x7B .. 0x7E	{|}~										x	x	x
0x7F .	(DEL)	x											

Example 4.9

```
// Ex 4.9
#include <iostream>
#include <string>
#include <cctype>
using namespace std;

string squeeze(string inpstr) {
    auto n = inpstr.size();
    int i = 0, j = 0;
    while (i < n) {
        if (!isblank(inpstr[i]) ) {
            inpstr[j] = inpstr[i];
            j += 1; // update j only if not blank
        }
        i += 1;
    }
    inpstr.resize(j);
    return inpstr;
}

int main() {
    cout << "Write equation\n";
    string equation;
    getline(cin, equation);

    equation = squeeze(equation);

    cout << equation << endl;
}
```

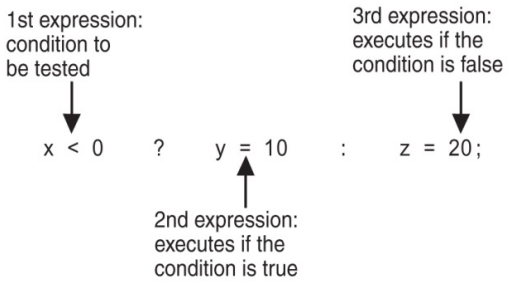
Output:

```
Write equation
9 * (9 - 49) * 5  / 4
9*(9-49)*5/4
```



4.11 The Conditional Operator

- This can be used to create short if/else statements
- Format: **`expr ? expr : expr;`**

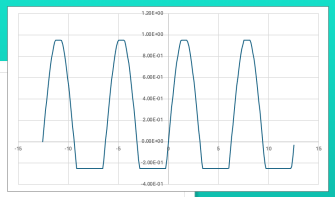


The Value of a Conditional Expression

- A conditional expression is an expression that uses a conditional operator
- The value of a conditional expression is determined by whichever of the subexpressions is executed

```
int num = 13;
string result = (num%2 ==0) ? "even" : "odd";
cout << num << " is " << result;
```

Example 4.10



```
// Ex 4.10
#include <iostream>
#include <fstream>
#include <cmath>
using namespace std;

const double PI = 3.1415927; // = M_PI; if MacOS

double clamp(double y, double min, double max)
{
    return (y>min? (y<max? y: max): min);
}

int main()
{
    ofstream fout("clamped.csv");
    double x = -4*PI;
    while (x <= 4*PI)
    {
        fout << x << "," << clamp(sin(x), -0.25, 0.95) << endl;
        x += 0.1;
    }
}
```

Output:

```
-12.5664,4.89859e-16
-12.4664,0.0998334
-12.3664,0.198669
-12.2664,0.29552
-12.1664,0.389418
-12.0664,0.479426
-11.9664,0.564642
-11.8664,0.644218
-11.7664,0.717356
…
```

4.12 The `switch` Statement

- The `switch` statements is used to select from multiple choices, sometimes used as an alternative to if/else if statements
- Format

```
switch (IntExpression)
{
case exp1:
  statements;
  break; // optional. without break it falls through
...
case expn:
  statements;
  break; // optional.
default: // everything else
  statements;
}
```

- **intExpression** is used to determine the cases to match and it **must be an integer** variable or a char, or an expression that evaluates to an integer value
- ***exp1*** through ***expn*** must be constant integer type expressions and must be unique in the switch statement
- ***default*** is optional but recommended

The `switch` Statement

- The `break` statement is used to stop execution in the current block
- It is used to execute the statements for a single case expression without executing the statements of the cases following it
- Example

```
char choice = readFromStdIn();
switch (choice){
  case 'Y': // falls through to next case
  case 'y':
    cout << "you said Yes!\n";
    break; // exits switch block
  case 'N':
  case 'n':
    cout << "you said No!\n";
    break; // exits switch block
  default: // optional catch all
    cout << "You are undecided\n";
}
```

4.13 Enumerated Data Types

- Enum Is a data type created by the programmer
- Contains a set of **named integer constants**
- Format:

```
enum name {val1, val2, … valn};
```

- Are associated with integers, starting with 0:
- Associated values can be changed

Enumerated Data Types

- Examples:

```
enum Fruit {apple, grape, orange};
cout << orange << endl; // prints 2
```

- All or some of their values can be overridden

```
enum Base {Bin=2, Oct=8, Dec=10,
 Hex=16};
```

Enumerated Data Types

- To define variables, use the enumerated data type name

```
Fruit snack;
Base defaultBase, address;
```

- A variable may contain any valid value for the data type

```
snack = orange;      // no quotes
if (defaultBase == Dec) // none here either
  cout << "Using Decimal\n";

if (defaultBase == 2) // comparing to int
  cout << "Using binary\n";

Hex = 6; // Error: cannot be changed
```

More about Struct

- The member variables can be initialized by default, so when a structure variable is created the members have predetermined values. This is called default member initializers (aka In-place member initializers). Example:

```
struct Rectangle
{
 int length = 1.; // Default member initializer
 int width = 1.;
};

Rectangle door;
cout << "length = " door.length << endl;
```

- This will output 1. to the standard output

Example 4.11

```
// Ex_4_11.cpp
#include <iostream>
#include <vector>
#include <fstream>
using namespace std;

enum Cardinal {East, North, West, South};

struct Coord
{
    int x=0, y=0;
};

Coord step(Coord p, Cardinal dir)
{
    switch (dir) {
        case East:
            p.x += 1;
            break;
        case North:
            p.y += 1;
            break;
        case West:
            p.x -= 1;
            break;
        case South:
            p.y -= 1;
            break;
    }
    return p;
}

vector<Coord> buildPath(vector<Cardinal> directions)
{
    vector<Coord> path(1, {0,0}); // init with 1 Coord 0,0
    int d = 0;
    while (d  < directions.size())
    {
        path.push_back(step(path.back(), directions[d]));
        d += 1;
    }
    return path;
}

int main()
{
    auto path = buildPath({East, South, South, East, East,
North});
    int i = 0;
    while (i < path.size())
    {
        cout << path[i].x << ',' << path[i].y << endl;
        i += 1;
    }
}
```

Output:

```
0,0
1,0
1,-1
1,-2
2,-2
3,-2
3,-1
```