C++

Control Structures

CO650 Advanced Programming

Topics

- Blocks
- Selection
- Iteration
- Logical Operators
- Arrays
- Random Numbers
- Enumerate Types
- Static Casting
- Pre processor Directives

Blocks

- Compound-statements
- The block of statements are enclose in curly brackets { }

```
{
        statment1;
        statment2;
        statment3;
}
```

if statement

- The if keyword is followed by a condition in brackets ()
- The condition is an expression that evaluates to true or false (a Boolean value)
- If true the statement following the condition or enclosed within the if block is/are execute.
- If the condition is false these statments are ignored

```
if (condition)  statement;
```

```
if (condition)
        statement;
```

```
if (condition)  {
        statement1;
        statement2;
}
```

if else statement

- Optionally we can specify statments to be executed if the condition evaluates to false.

```
if (condition)
    statement1;
else
    statment2;
```

```
if (condition)  {
        statement1;
        statement2;
}
else {
        statement3;
        statement4;
}
```

If Statement Examples

```
if (age >= 18)
		cout << "You can vote" << endl;
```

```
if (age >= 18)
		cout << "You can vote" << endl;
else
		cout << "You are too young to vote" << endl;
```

```
if (age >= 18){
		cout << "You can vote" << endl;
		cout << "Register here" << endl;
}
else {
		cout << "You are too young to vote" << endl;
		cout << "Try again in " << (18-age) << " Years" << endl;
}
```

Arithmetic if Operator (?:)

- Also known as the conditional operator.
- Only C++ ternary operator (takes 3 arguments).
- Evaluates an expression and returns one value if the expression is true and another if it is false.

```
returnValue = (condition) ? result1: result2;
```

- If condition is true result1 is returned else result2 is returned.

```
int x = 1,y = 2, lower;

lower = (x <= y) ? x : y;

cout << lower << endl;    //  Displays 1
```

Switch Statement

- An alternative to multiple if statements
- The break keyword avoids fall through to the next statement.
- The default case is optional

```
switch(expression) {
   case constant1 : statement/s;
                    break;
   case constant2 : statement/s;
                    break;
   default: statement/s
}
```

Switch Statement Example

```
int option;
cout << "Enter your preferred means of transport (1..3) ";
cin >> option;

switch(option) {
        case 1 : cout << "Go by bus" << endl;
                 break;
        case 2 : cout << "Go by train" << endl;
                 break;
        case 3 : cout << "Go by plane" << endl;
                 break;
        default : cout << "Invalid option selected" << endl;
}
```

While Loops

- While Loop executes the statements in the body {} while the expression evaluates to true

```
while(expression) {
      statement/s;
}
```

- do while loops are similar to the while loop except the statements are garanteed to execute at least once (usefull for validation).

```
do {
    statement/s;
}
while(expression);
```

For Loops

The for loop is known as a counting loop

```
for(initialisation;condition; increment) {
      statement/s;
}
```

1. The initialisation expression is executed once only, during the first iteration.
2. The condition expression is then evaluated.
3. If the condition is true the statements within the body are executed.
4. The increment expression is then executed.
5. Step 3 & 4 are repeated.

```
for(int i=0; i< 5; i++){
            cout << i << endl;
}
```

Logical Operators

- ! (Logical NOT) Flips the Boolean value

```
cout << (3 > 1) << endl;   // Displays 1 (true)
cout << !(3 > 1) << endl;  // Displays 0 (false)
```

- && (Logical AND) Used to evaluate two expressions and return a single Boolean value. Where true is returned if both expressions are true

```
cout << ((3 > 1)&&( 3 < 10)) << endl;  // Displays 1 (true)
cout << ((3 > 1)&&( 3 < 1)) << endl;  // Displays 0 (false)
```

- || (Logical OR) Used to evaluate two expressions and return a single Boolean value. Where true is returned if one or more expressions are true

```
cout << ((3 > 1)||( 3 < 10)) << endl;  // Displays 1 (true)
cout << ((3 > 1)||( 3 < 1)) << endl;  // Displays 1 (true)
```

Static Arrays

- The array is declared with the number of elements placed in the [] brackets

```
Type arrayName[ size];
```

- Individual elements can be accessed through an index value.
- The first element having an index on 0 and the last element an index of size-1

```
arrayName[ index]
```

```
int marks[5];
marks[0] = 40;
marks[1] = 65;
marks[2] = 54;
marks[3] = 77;
marks[4] = 53;
for (int n=0; n < 5;n++){
        cout << "Mark " << n << " is " << marks[n] << endl;
}
```

Array Pointer

- The array name contains a constant pointer to the first element of the array.

```
Type arrayName[ size];
```

- The array is a contiguous block of memory.
- The array name can be dereferenced to obtain the values within the elements.

```
*arrayName // alternative to arrayName[0]
```

```
*(arrayName + 1)  // alternative to arrayName[1]
```

```
for (int n=0; n < 5;n++){
        cout << "Mark " << n << " is " << *(marks+n) << endl;
}
```

Initialising Arrays

- Global arrays are automatically initialised with the default values for the array type.
- When declaring an array within local scope the elements will not be initialise (undetermined). So it may be necessary to do so within your code.
- Both global and local arrays can be intialised when declared.

```
Type arrayName[optional size] = { value1,value2};
```

- If an optional size is included the values must not exceed this

```
int scores[] = {20,13,45};
for (int n=0; n < 3;n++){
        cout << "Score " << n << " is " << scores[n] << endl;
}
```

Multidimensional Arrays

- Multi dimensional array can be declared by appending additional []

```
type arrayName[size][size];
```

```
int board[8][8];
for(int r=0;r<8;r++)
    for(int c=0;c<8;c++)
        board[r][c] = 0;
```

- Only the size of the first dimension may be omitted if the array is initialised

```
int  testArray[4][2] = { {1,1},{2,2},{3,3},{4,4}};
```

or

```
int  testArray[ ][2] = { {1,1},{2,2},{3,3},{4,4}};
```

Random Numbers

- The rand() function returns a number in the range 0 to RAND_MAX
- RAND_MAX being defined within <cstdlib> usual value is 32767
- The algorithm used generates a sequence of random numbers
- The algorithm uses a **seed** to generate the sequence
- Invoke srand (once only) passing it a distinctive value before generating the random number/s to ensure the seed is not repeated.
- To generate a random number with a specific range use the modulo operator to return the remainder of integer division.

Random Numbers

- It is common practice to use the time to seed the random sequence.
- time(NULL) returns the number of seconds since 1/1/1970
- The example below generates a random number in the range 0..9

```
#include <time.h>

int main(){
    srand(time(NULL));
    int number  = rand() % 10;
}
```

Enumerate Types

- A user defined type
- Consisting of a set of named/symbolic constants (enumerators)

```
enum EnumTypeName { enum1, enum2, enum3 };
```

- The enumeration defines a range of values
- A variable of this type can be declared

```
EnumTypeName VariableName;
```

Enumerate Types

- The variable matchDay can only be assigned an enumerator of the DaysOfWeek type.
- Trying to do otherwise will generate a compiler error.

```
enum DaysOfWeek { Mon,Tue,Wed,Thur,Fri,Sat,Sun};

int main(){
        DaysOfWeek matchDay;
        matchDay = Sat;
        cout << "Match Day is " << matchDay << endl;
        return 0;
}
```

- Each enumerator is assigned a number that reflects its place in the list of enumerators. Starting at 0.
- The output from the above example is 5.

Enumerate Types

- Enumerate types improve the readability of the code.

```
enum DaysOfWeek { Mon,Tue,Wed,Thur,Fri,Sat,Sun};

int main(){
        DaysOfWeek matchDay;
        matchDay = Sat;
        if (matchDay == Sat)
                cout << "I will go " << endl;
        else
            cout << "I‘m not able to go" << endl;
        return 0;
}
```

Enumerate Switch Example

```
enum DaysOfWeek { Mon, Tue, Wed, Thu, Fri, Sat, Sun};

int main(){
        for(int n = Mon; n <= Sun;n++){
                switch(n){
                        case Sat:
                        case Sun: cout << n <<  " is a weekend day" << endl;
                                break;
                        case Mon:
                        case  Tue:
                        case Wed:
                        case  Thu:
                        case    Fri: cout << n << " is a week day" << endl;
                }
        }
        return 0;

}
```

Note: Enumerate types can’t be incremented using ++, hence n is of type int

Cast Enumerate Types

- The default integer value can be overridden

```
enum DaysOfWeek { Mon=1,Tue=2,Wed=3,Thur=4,Fri=5,Sat=6,Sun=7};
```

- An integer value can by typecast to an enumerate type

```
(EnumTypeName)IntVariable
```

```
enum DaysOfWeek { Mon=1,Tue=2,Wed=3,Thur=4,Fri=5,Sat=6,Sun=7};
DaysOfWeek day;
srand(time(NULL));
day = (DaysOfWeek)((rand() % 7)+1);
cout << "Day : " << day << endl;   // Displays the integer value
```

- When casting a random number place brackets around the calculation before casting. (DaysOfWeek)rand() % 7 will generate a compiler error.

static_casting

- Converts primitive types or pointers.
- No run-time check is made so may be unsafe

```
static_cast<type-id>(expression)
```

- Alternative to C type casting

```
(type-id)expression
```

```
day = static_cast<DaysOfWeek>((rand() % 7)+1);
```

- Programmer's responsibility to ensure conversion is safe. If not results are undefined.

Pre-processor Directives

- Instructions for the pre-processor
- Executed before the code is compiled
- Some common directives include

  #include, #define, #undef, #ifde, #endif, #else
- All directives start with a #
- Extend over a single line
- Do not end in a semi colon

Macro Definitions

To define a macro use the #define directive followed by its name and value.

```
#define name value
```

```
#define SIZE 100
```

The pre-processor will replace all instances of the macro within the code with the macro's value.

Before

```
if (count > SIZE) {

}
```

After

```
if (count > 100) {

}
```

Macro definition of constants is not encouraged as it is not type safe.

Conditional Compilation

#ifdef returns true if a macro has been defined
(it does not have to be assigned a value)

```
#define XBOX

#ifdef XBOX
        state = controller.getState();
#endif
```

The C++ statement will only be compiled if the XBOX macro is defined. As it is in this case.
Statements to be conditionally compiled are placed between the #ifdef and #endif directives

Conditional Compilation

#undef undefines a previously defined variable

```
#define XBOX
#undef XBOX

#ifdef XBOX
        state = controller.getState();
#endif
```

The C++ statement above will not be compiled. Alternatively we could comment out the define to stop the compilation.

```
//#define XBOX

#ifdef XBOX
        state = controller.getState();
#endif
```

Conditional Compilation

#else provides an alternative compilation path

```
#define XBOX
#undef XBOX

#ifdef XBOX
        state = controller.getState();
#else
        state = keyboard.getState();
#endif
```

In this example XBOX is undefined so the

state = keyboard.getState();

Statement will be compiled.

Summary

- In common with most languages C++ has selection and iteration constructs.
- Arrays are one of a number of data structures we will be using.
- Enumerate types provide a type safe readable solution.
- Pre processor directives can determine which code is included within the executable.