Preview

- Relational operator
- If, if-else-if, nested if statement
- Logical operators
- Validating Inputs
- Compare two c-string
- Switch Statement
- Increment decrement operator
- While, Do-While, For Loop
- Break, Continue statement
- Function and Function prototype
- Local and global variable
- Parameter passing methods: Call by value, Call by reference

Review for COSC 120

(File Processing: Reading Data From a File)

Read data From a File

1. include the fstream header file;
2. declare a variable of type ifstream – create object to handle a file
3. open the file – by using the function open in ifstream object
4. check for an open file error
5. read from the file
6. after each read, check for end-of-file using the eof() member function
7. close the file when access is no longer needed

Review for COSC 120

(File Processing: Reading Data From a File)

```
// readfile.cpp reads values from the file 'outfile.txt'
// and echoes them to the display until a negative value
// is read.
#include <iostream>
#include <fstream>
#include <cstdlib> // for exit function

using namespace std;

int main()
{
        ifstream indata; // create ifstream object to read a data
        int num;
        indata.open("infile.txt", ios::in); // opens the file
        if(!indata)  // checking whether 'outfile.txt' can open or not
        {
                cerr << "Error: file could not be opened" << endl;
                exit(1);
        }
        indata >> num;
        while (!indata.eof()) // keep reading until end-of-file
        {
                cout << "The next number is " << num << endl;
                indata >> num; // sets EOF flag if no value found
        }
        indata.close(); //close the file 'outfile.txt'
        cout << "End-of-file reached.." << endl;
        return 0;
}
```


Review for COSC 120

(File Processing: Reading Data From a File more than one time)

- Read data more than one time from the beginning of a File
  - clear()
  - seekg(0)
- Sequence of two function calls let file pointer move to the beginning of a file!


```
// readagain.cpp reading a sequential access file example
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>
using namespace std;
int main()
{
    ifstream sequential;
    sequential.open("input.txt",ios::in);
    if (!sequential)
    {
            cerr<<" File cannot be opened this time"<<endl;
            exit(1);
    }
    int age;
    char name[20];
    cout <<"     List of Man    "<<endl;
    cout <<"********************"<<endl;

    cout <<"Name"<<setw(20)<<"Age \n";
    cout <<"----"<<setw(20)<<"--- \n";
    while (sequential >> name >>age)
    {
            if (age > 18)
            cout <<name<<setw(20)<< left<<age <<setw(20)<<endl;
    }
    sequential.clear();
    sequential.seekg(0);
    cout <<endl<<endl;
    cout <<"  List of Children    "<<endl;
    cout <<"********************"<<endl;

    cout <<"Name"<<setw(20)<<right<<"Age \n";
    cout <<"----"<<setw(20)<<"--- \n";
    while (sequential >> name >>age)
    {
            if (age <= 18)
            cout <<left <<name<<setw(20)<< age<<setw(20)<<endl;
    }
    return 0;
```


Review for COSC 120

(File Processing: Writing Data To a File)

Writing Data To a File

1. include the **fstream** header file;
2. declare a variable of type **ofstream** – create **ofstream** object
3. open the file
4. check for an open file error
5. use the file
6. close the file when access is no longer needed

Review for COSC 120
(File Processing: Writing Data To a File)

```
// writefile output values from an array
// to a file named outfile.txt
#include <iostream>
#include <fstream>
#include <cstdlib> // for checking error for opening a file
using namespace std;
int main()
{
        ofstream outdata;
        int i;
        int num[5] = {4, 3, 6, 7, 12};

        outdata.open("outfile.txt",ios::out);// create a file
        if( !outdata )
        {
                cerr << "Error: file could not be opened" << endl;
                exit(1);
        }
        for (i=0; i<5; ++i)
                outdata << num[i] << endl; // write data to the file
        outdata.close(); // close file
        return 0;
}
```


Review for COSC 120
(Relational Operators)

- Relational operators are used to compare numerical values.
- A comparison result is true or false.

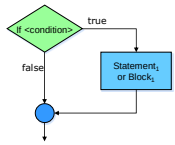
Relational Operators	
Operator Name	**Syntax**
Less Than	`a < b`
Less Than or Equal To	`a <= b`
Greater Than	`a > b`
Greater Than or Equal To	`a >= b`
Not Equal To	`a != b`
Equal To	`a == b`

Review for COSC 120
(The if Statement)

- The if statement change the execution flow of a program based on the conditional expression.

```
if (<conditional expression>)
    statement;
```

```
if (<conditional expression>)
{
    statement1;
    statement2;
    ....
    statementn;
}
```



Review for COSC 120
(The if Statement)

- Careful with semicolon in a if statement

Ex)

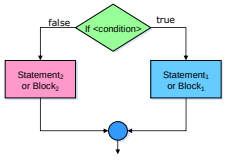
```
if (<condition>);
statementk;
```

- If <condition> is true, execute empty statement since there is semicolon.
- $Statement_k$ always executed no matter what <condition> is

Review for COSC 120
(The if-else Statement)

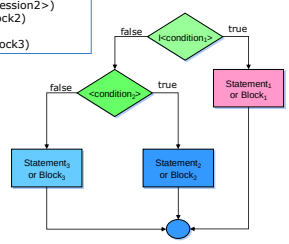
```
if (<conditional expression>)
    Statement1;
else
    Statement2;
```

```
if (<conditional expression>)
{
    Statementa1;
    Statementa2;
     ....
    Statementam;
}
else
{
    Statementb1;
    Statementb2;
     ....
    Statementbm;
}
```



Review for COSC 120
(The if-else-if Statement)

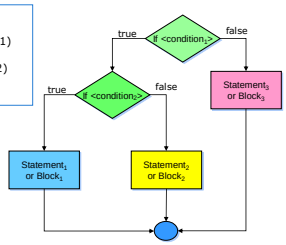
```
if (<conditional expression1>)
    Statement1; (or block1)
else if (<conditional expression2>)
    Statement2; (or block2)
else
    Statement3; (or block3)
```



Review for COSC 120
(The Nested if Statement)

An if statement is appears inside another if statement, it is considered nested if statement.

```
if <condition1>
    if <condition2>
        Statement1; (or block1)
    else
        Statement2;(or block2)
else
    Statement3; (or block3)
```



Review for COSC 120
(Logical Operators)

- Logical operator connects two or more relational expression into one or reverses the logic of an expression.

Logical Operators	
Operator Name	**Syntax**
Logical Negation	`!E`
Logical AND	E_1 `&&` E_2
Logical OR	E_1 `||` E_2

Review for COSC 120
(Validating Inputs)

- Each input should be validated by the program.
- When a program gets an input, the program must validate it.
- If input is illegal, the program need to display message such as " Illegal input! Please provide legal input", and ask a input again.

Review for COSC 120
(Compare Two Strings)

- Cannot compare two C strings (array of character) with the relational operator ==.
  - Define a function to compare two strings
  - Use strcmp function in <cstring> library

Review for COSC 120
(Conditional Operator ? :)

- C++ & C provides the conditional operator **? :**
- It took three operands. The operands, together with the conditional operator, form a conditional expression

```
/* if <condition> is true execute statement1 take
statement2 otherwise.*/

<condition>? <statement1> : <statement2>
```


Review for COSC 120
(Switch Statement:)

```
switch (expr)
{
      case c1: statement1; // do statement1 if expr == c1
            break;
      case c2: statement2; // do statement2 if expr == c2
            break;
      case c3: statement3; // do statement3 if expr == c3
            break;
      …
      case cn: statementn; // do statementn if expr == cn
            break;
      default: statement; // do statement otherwise
}
```

Review for COSC 120
(Increment and Decrement Operators)

- ++ and – are operators that add and subtract 1 from their operands.
- There are post and pre increment and decrement operators
  - Pre increment: ++a
  - Pre decrement: -- a
  - Post increment a++
  - Post decrement a –

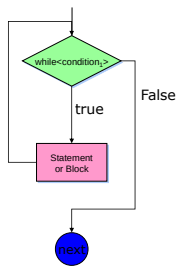
Review for COSC 120
(Increment and Decrement Operators)

```
// Pre increment and decrement example
#include <iostream>
using namespace std;

void main()
{
      int score1 = 5;
      int score2 = 5 ;
      cout << ++score1 <<endl; //pre increment
      cout <<--score2 <<endl ; //pre decrement

}
```


Review for COSC 120
(The while Loop: pre-test)

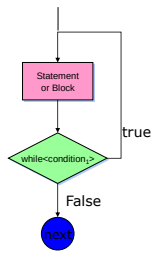
```
while (<conditional expression>)
        Statement;
```

```
while (<conditional expression>)
{
        Statement1;
        Statement2;
        ….
        Statementn;
}
```



Review for COSC 120
(The do-while Loop: post-test)

```
do{
        statement1;
        Statement2;
        …;
        Statementn;
} while (<conditional expression>);
```



Review for COSC 120
(The for Loop: free test)

```
for (startExpression; test_expression; updateExpression)
        statement;
```

```
for (startExpression; testExpression; updateExpression)
{
        statement1;
        statement2;
        ….;
        statementn;
}
```


Review for COSC 120
(The for Loop)

- The **startExpression** may be omitted from inside the for loop's parentheses if it has already been performed or no initialization in needed.

```
// for loop example
#include <iostream>
using namespace std;

void main()
{
        int count =1;
        for (; count <= 10; count++)
                cout << "Round: " << count << endl;
}
```

Review for COSC 120
(The for Loop)

- The **updateExpression** may be omitted from inside the for loop's parentheses if it is being performed elsewhere in the loop or if none is needed.

```
// for loop example
#include <iostream>
using namespace std;

void main()
{
        int count =1;
        for (; count <= 10; )
        {
                ++count ;
                cout << "Round: " << count << endl;
        }
}
```


Review for COSC 120
(The break Statement)

```
//break statement inside a while loop
#include <iostream>
using namespace std;

void main()
{
        int score = 0;

        while (score <= 60)
        {
            cout << " What is your score?" << endl;
            cin >> score;
            if (score > 60)
                break;
            cout << " Sorry, You better retake this course !" << endl;
        }
        cout <<" Congratulation ! You finally passed"<<endl;
}
```


Review for COSC 120
(The continue Statement)

```
// continue statement
#include <iostream>
using namespace std;

void main()

{
        for (int num =0; num <= 10; num++)
        {
                if ((num % 2)==0)
                        continue;
                cout << num << " is odd number"<<endl;
        }
}
```


Review for COSC 120
(User Defined Function)

The general form of a function definition in C++ is as follows:

```
Return_type Function_Name (parameter_List )
{
       Local variables declarations;

       Function_Implementation;
}
```


Review for COSC 120
(The Function Prototype)

- One of the most important features of C++ is the function prototypes
- Function prototypes tells the compiler
  - Name of function
  - Return type
  - Number of parameter list
  - Type of each parameter
- The compiler use function prototypes to validate function calls.
- Function prototypes are usually saved in a header file which need to be included in a program.

Review for COSC 120
(Local and Global Variables)

- We call a variable as a local variable, if it is defined (declared) locally in a function.
- Since local variables are defined locally in a function, it can only be visible inside the function.
- We call a variable as global variable, if it is defined outside all the function in a program.
- It is visible in any location in the program.

Review for COSC 120
(Parameter Passing in C ++: Call by Value and Call by Reference)

```
// call-by-value and call-by-reference in c++
# include <iostream>
using namespace std;
int callbyvalue(int);
void callbyreference(int &);
void main()
{
     int a = 10;
     int b = 10;
     int  c;
     cout << "before call callbyvalue, a = " << a <<endl;
     c = callbyvalue(a);
     cout <<"after call callbyvalue, a = " << a << endl;
     cout << "before call callbyreference b = " << b <<endl;
     callbyreference(b);
     cout <<"after call callbyreference b =" << b << endl;
}
int callbyvalue(int x)
{
     x = x + x;
     return x;
}
void callbyreference(int &x)
{
      x += x;
}
```


Review for COSC 120
(The return Statement)

- The **return** statement stops execution and returns to the calling function.
- When a return statement is executed, the function is terminated immediately at that point, regardless of whether it's in the middle of a loop, etc.

Review for COSC 120
(The Static Variables)

Static Variable

- Defined locally
- One time initialization
- It only can be modified inside function, when the function is called
- It remains in memory until the end of the program

Review for COSC 120
(The Static Variables)

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

int function(int) //function prototype
int main()
{
    int z =0;
    for (int y=0; y<=5; y++)
    {
        z= function(10);
        cout <<"function returns "<< z <<endl;
    }
}

int function(int y)
{
    int x = 5;
    x = x + y;
    return x;
}
```


Review for COSC 120
(The Static Variables)

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

int function(int) //function prototype
int main()
{
    int z =0;
    for (int y=0; y<=5; y++)
    {
        z= function(10);
        cout <<"function returns "<< z <<endl;
    }
}

int function(int y)
{
    static int x = 5;
    x = x + y;
    return x;
}
```


Review for COSC 120
(Default Arguments)

- It is possible to assign **default arguments** to function parameters.
- A **default argument** is passed to the parameter when the actual argument is left out of the function call.
- The **default arguments** are usually listed in the function prototype.

Review for COSC 120
(Default Arguments)

```
// This program shows a example of default arguments
#include <iostream>
#include <iomanip>
using namespace std;

// function prototype with default arguments
void LunchSpecial (char []= "Sandwitch", char [] ="Toco");

void main()
{
        LunchSpecial();
        LunchSpecial("Hamsandwich");
        LunchSpecial("Hamburger", "chiken");
}

void LunchSpecial (char x[], char y[])
{
        cout << "Today's Lunch Special: " << x  <<" and " << y <<endl;
}
```


Review for COSC 120
(Overloading Functions)

- In function overloading, the function is said to be overloaded when same name is given to different functions.
- However, the functions will differ at least in any one of the these.
  - The number of parameters,
  - The data type of parameters,
  - The order of appearance
- These three together are referred to as the **function signature**.

Review for COSC 120
(Overloading Functions)

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

void print(int i)
{
  cout << " Here is int " << i << endl;
}
void print(double  f)
{
  cout << " Here is float " << f << endl;
}

void print(char* c)
{
  cout << " Here is char* " << c << endl;
}

void main()
{
  print(10);
  print(10.10);
  print("ten");
}
```


Review for COSC 120
(Overloading Functions)