# Programming Fundamentals I

## Chapter 7: User-Defined Simple Data Types, Namespaces, and the `string` Type

Dr. Adriana Badulescu

# Objectives

- Learn how to create and manipulate your own simple data type called the enumeration type
- Become familiar with the `typedef` statement
- Learn about the `namespace` mechanism
- Explore the `string` data type and learn how to use the various string functions to manipulate strings

# Enumeration Type

- **Data type**: a set of values together with a set of operations on those values
  - int data type
    - Values: the integers between -2,147,483,648 and 2,147,483,647
    - Operations: +, -, *, /, %, ++, --, =, +=, -=, *=, /=
- To define a new simple data type, we need:
  - A name for the data type
  - A set of values for the data type
  - A set of operations on the values

# Enumeration Type

- A new simple data type can be defined by specifying its name and the values, but not the operations
  - The values must be identifiers
- Syntax:

```
enum EnumTypeName {value1, value2, ...} ;
```

  - `value1`, `value2`, … are identifiers called **enumerators**
  - The set of values is ordered

```
value1 < value2 < value3 <...
```

# Enumeration Type

- The values have to be identifiers

```
enum grades {'A', 'B', 'C', 'D', 'F'}; //illegal enumeration type
enum places {1ST, 2ND, 3RD, 4TH};  //illegal enumeration type
```

```
enum grades {A, B, C, D, F};
enum places {FIRST, SECOND, THIRD, FOURTH};
```

- If a value has been used in one enumeration type, it can't be used by another enumeration type in same block
- The same rules apply to enumeration types declared outside of any blocks

```
enum mathStudent {JOHN, BILL, CINDY, LISA, RON};
enum compStudent {SUSAN, CATHY, JOHN, WILLIAM}; //illegal
```

# Enumeration Type

Default values

0 1 2 3 4

```
enum colors {BROWN, BLUE, RED, GREEN, YELLOW};
```

defines a new data type, called `colors`, and the values belonging to this data type are BROWN, BLUE, RED, GREEN, and YELLOW.

0 1 2 3

```
enum standing {FRESHMAN, SOPHOMORE, JUNIOR, SENIOR};
```

defines `standing` to be an enumeration type. The values belonging to `standing` are FRESHMAN, SOPHOMORE, JUNIOR, and SENIOR.

# Declaring Variables

- Syntax:

```
EnumTypeName  identifier, identifier, … ;
```

- For example, given the following definition:

```
enum sports {BASKETBALL, FOOTBALL, HOCKEY, BASEBALL, SOCCER,
             VOLLEYBALL};
```

we can declare the following variables:

```
sports popularSport, mySport;
```

# Assignment

- The assignment statement:

```
popularSport = FOOTBALL;
```

stores `FOOTBALL` into `popularSport`

- The statement:

```
mySport = popularSport;
```

copies the value of the `popularSport` into `mySport`

# Operations on Enumeration Types

- No arithmetic operations are allowed on enumeration types

```
mySport = popularSport + 2;              //illegal
popularSport = FOOTBALL + SOCCER;        //illegal
popularSport = popularSport * 2;         //illegal
```

- ++ and -- are illegal too:

```
popularSport++; //illegal
popularSport--; //illegal
```

- Solution: use a static cast:

```
popularSport = static_cast<sports>(popularSport + 1);
```

```
enum sports {BASKETBALL, FOOTBALL, HOCKEY, BASEBALL, SOCCER, VOLLEYBALL};
```

# Relational Operators

- An enumeration type is an ordered set of values:

```
FOOTBALL <= SOCCER is true
HOCKEY > BASKETBALL is true
BASEBALL < FOOTBALL is false
```

- Enumeration type is an integer data type and can be used in loops:

```
for (mySport = BASKETBALL;
     mySport <= SOCCER;
     mySport = static_cast<sports>(mySport + 1))
```

```
enum sports {BASKETBALL, FOOTBALL, HOCKEY, BASEBALL, SOCCER, VOLLEYBALL};
```

# Input /Output of Enumeration Types

- I/O are defined only for built-in data types
- Enumeration type cannot be input/output directly

Input

```
enum objectType{ROCK, PAPER, SCISSORS};
```

Output

```
objectType object;
switch (selection)
{
case 'R':
case 'r':
    object = ROCK;
    break;
case 'P':
case 'p':
    object = PAPER;
    break;
case 'S':
case 's':
    object = SCISSORS;
}
return object;
```

```
switch (object)
{
case ROCK:
    cout << "Rock";
    break;
case PAPER:
    cout << "Paper";
    break;
case SCISSORS:
    cout << "Scissors";
}
```

# Functions and Enumeration Types

- Enumeration types can be passed as parameters to functions either bv value or bv reference

```
void convertEnum(objectType object)
{
    switch (object)
    {
    case ROCK:
        cout << "Rock";
        break;
    case PAPER:
        cout << "Paper";
        break;
    case SCISSORS:
        cout << "Scissors";
    }
}
```

# Functions and Enumeration Types

- A function can return a value of the enumeration type

```
objectType retrievePlay(char selection)
{
    objectType object;
    switch (selection)
    {
    case 'R':
    case 'r':
        object = ROCK;
        break;
    case 'P':
    case 'p':
        object = PAPER;
        break;
    case 'S':
    case 's':
        object = SCISSORS;
    }
    return object;
}
```

# Declaring Variables When Defining the Enumeration Type

- You can declare variables of an enumeration type when you define an enumeration type:

```
enum grades {A, B, C, D, F} courseGrade;
```

# Anonymous Data Types

- **Anonymous type** : values are directly specified in the declaration, with no type name

```
enum {BASKETBALL, FOOTBALL, BASEBALL, HOCKEY} mySport;
```

- Drawbacks:
  - Cannot pass/return an anonymous type to/from a function
  - If you want to use it again, you cannot declare it (with same identifiers) in the same block

# Exercise: Rock-Paper-Scissor Game

- Implement the Rock-Paper-Scissor Game using enumeration

```
//declare the enumeration for the game
enum RockPaperScissorsGame { ROCK, PAPER, SCISSORS };

void PrintGameChoice(RockPaperScissorsGame Player)
{
        cout << "\nYour choice: ";
        switch (Player)
        {
        case ROCK:
                cout << "ROCK";
                break;
        case PAPER:
                cout << "PAPER";
                break;
        case SCISSORS:
                cout << "SCISSORS";
                break;
        }
}
```

# Exercise: Rock-Paper-Scissor Game

```
//function to enter the game choice for a player
void EnterGameChoice(RockPaperScissorsGame &Player)
{
	//declare the character choice
	char Choice;
	do
	{
		//print a message with the menu option
		cout << "\n\nPlease enter your play choice (R=ROCK, P=PAPER, S=SCISSORS): ";
		//input the character choice
		cin >> Choice;
		//compute the game choice
		if (toupper(Choice) == 'R')
			Player = ROCK;
		else
			if (toupper(Choice) == 'P')
				Player = PAPER;
			else
				if (toupper(Choice) == 'S')
					Player = SCISSORS;
				else
					cout << "Wrong game choice! Try again!";
	} while ((toupper(Choice) != 'R') && (toupper(Choice) != 'P') && (toupper(Choice) != 'S'));
}
```

# Exercise: Rock-Paper-Scissor Game

```
int main()
{

        //declare Rock Paper Scissors Game

        //declare variable for player 1
        RockPaperScissorsGame Player1;

        //declare variable for player 2
        RockPaperScissorsGame Player2;

        //declare and initialize variables for the score
        int ScorePlayer1 = 0;
        int ScorePlayer2 = 0;

        //repeat as long as they wanna play the game
        char PlayAgain = 'Y';
        do
        {

                //get the Player 1 choice
                EnterGameChoice(Player1);

                //output the choice for player 1
                PrintGameChoice(Player1);

                //get the game choice for player 2
                EnterGameChoice(Player2);

                //output the choice for player 2
                PrintGameChoice(Player2);
```

# Exercise: Rock-Paper-Scissor Game

```
//decide who won the round
cout << "\n\nROUND WINNER: ";
//if ( (Player1== ROCK) && (Player2 == ROCK) )
if (Player1 == Player2)
        cout << "DRAW";
else
        if ((Player1 == ROCK) && (Player2 == PAPER))
        {
                cout << "PLAYER2 with PAPER";
                ScorePlayer2++;
        }
        else
                if ((Player2 == ROCK) && (Player1 == PAPER))
                {
                        cout << "PLAYER1 with PAPER";
                        ScorePlayer1++;
                }
                else
                        if ((Player1 == PAPER) && (Player2 == SCISSORS))
                        {
                                cout << "PLAYER2 with SCISSORS";
                                ScorePlayer2++;
                        }
                        else
```

# Exercise: Rock-Paper-Scissor Game

```
if ((Player2 == PAPER) && (Player1 == SCISSORS))
{
        cout << "PLAYER1 with SCISSORS";
        ScorePlayer1++;
}
else
        if ((Player1 == SCISSORS) && (Player2 == ROCK))
        {
                cout << "PLAYER2 with ROCK";
                ScorePlayer2++;
        }
        else
                if ((Player2 == SCISSORS) && (Player1 == ROCK))
                {
                        cout << "PLAYER1 with ROCK";
                        ScorePlayer1++;
                }
                else
                        cout << "ERROR";
```

# Exercise: Rock-Paper-Scissor Game

```
        //output the win/score
        cout << "\n\nCURRENT SCORE: PLAYER1=" << ScorePlayer1 << " PLAYER2=" << ScorePlayer2 <<"\n";

        //ask if they wanna play again
        cout << "\n\nDo you want to play again (enter Y for Yes and N for No)? ";
        cin >> PlayAgain;

    }
    while (toupper(PlayAgain) == 'Y');

    //print the game winner
    if (ScorePlayer1 > ScorePlayer2)
        cout << "\n\nPLAYER1 WON THE GAME!";
    else
        if (ScorePlayer1 < ScorePlayer2)
            cout << "\n\nPLAYER2 WON THE GAME!";
        else
            cout << "\n\nIT IS A TIE!";
```

# Exercise: Rock-Paper-Scissor Game

```
Please enter your play choice (R=ROCK, P=PAPER, S=SCISSORS): R
Your choice: ROCK
Please enter your play choice (R=ROCK, P=PAPER, S=SCISSORS): P
Your choice: PAPER
ROUND WINNER: PLAYER2 with PAPER
CURRENT SCORE: PLAYER1=0 PLAYER2=1

Do you want to play again (enter Y for Yes and N for No)? Y

Please enter your play choice (R=ROCK, P=PAPER, S=SCISSORS): P
Your choice: PAPER
Please enter your play choice (R=ROCK, P=PAPER, S=SCISSORS): R
Your choice: ROCK
ROUND WINNER: PLAYER1 with PAPER
CURRENT SCORE: PLAYER1=1 PLAYER2=1

Do you want to play again (enter Y for Yes and N for No)? N

IT IS A TIE!
```

# typedef Statement

- You can create synonyms or aliases to a data type
- Syntax: `typedef ExistingTypeName  NewTypeName;`
- `typedef` does not create any new data types

```
typedef bool Boolean;
const Boolean TRUE = 1;
const Boolean FALSE = 0;
Boolean flag;
flag = TRUE;
flag=false;
```

# Namespaces

- **ANSI/ISO standard C++**
  - Was officially approved in July 1998
  - Most of the recent compilers are also compatible with it
  - For the most part, standard C++ and ANSI/ISO standard C++ are the same, however, ANSI/ISO Standard C++ has some features not available in Standard C++
- Global identifiers in a header file used in a program become global in the program
  - Syntax error occurs if an identifier in a program has the same name as a global identifier in the header file
- ANSI/ISO Standard C++ attempts to solve this problem with the **namespace mechanism**

# Namespaces

- Syntax:

```
namespace NameSpaceName
{
        members
}
```

- Member is usually a variable declaration, a named constant, a function, or another namespace

```
namespace globalType
{
    const int N = 10;
    const double RATE = 7.50;
    int count = 0;
    void printResult();
}
```

# Namespaces

- The scope of a `namespace` members is local to the namespace
- The `namespace` members can be accessed outside the `namespace`:

```
NamespaceName::identifier
```

```
using namespace NamespaceName;
```

```
using NamespaceName::identifier
```

- Examples:

| | |
|---|---|
| `globalType::RATE`<br>`globalType::printResult();` | `using namespace globalType;`<br>`printResult();` |

# Namespaces

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

int main()
{
    char x, y;
    string z;
    cin >> x;
    cin.get(y);
    getline(cin,z);
    cout << x << y << z;
}
```

```
#include <iostream>
#include <string>

int main()
{
    char x, y;
    std::string z;
    std::cin >> x;
    std::cin.get(y);
    getline(std::cin,z);
    std::cout << x << y << z;
}
```

# `string` Type

- To use the data type **`string`**, the program must include the header file `string`
- The statement:

```
string name = "William Jacob";
```

  declares name to be a string variable and also initializes name to `"William Jacob"`
  - `name` is capable of storing any size string
  - The first character, `'W'`, is in position `0`

| W | i | l | l | i | a | m | | J | a | c | o | b |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

# `string` Type

- Binary operator `+` and the array subscript operator `[]`, have been defined for the data type `string`
  - **+** performs the **string concatenation operation**
- Example:
  - `str1 = "Sunny";`

| S | u | n | n | y |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |

| | D | a | y |
|---|---|---|---|
| 0 | 1 | 2 | 3 |

  - `str2 =  str1 + " Day";`

| S | u | n | n | y | | D | a | y |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

# `string` Constants

- **string::npos**

A static member constant value with the greatest possible value for an element of type size_t. The maximum size for a string is 4294967295 on many machines.

# `string` Functions

| Method | Effect |
|---|---|
| **size** | Return length of string |
| **length** | Return length of string |
| **max_size** | Return maximum size of string |
| **resize** | Resize string |
| **clear** | Clear string |
| **empty** | Test if string is empty |
| **operator[]** | Get character in string |
| **at** | Get character in string |

# `string` Functions

| Method | Effect |
|---|---|
| operator= | String assignment |
| operator+= | Append to string |
| append | Append to string |
| push_back | Append character to string |
| assign | Assign content to string |
| insert | Insert into string |
| erase | Erase characters from string |
| replace | Replace part of string |
| swap | Swap contents with another string |

# `string` Functions

| Method | Effect |
|---|---|
| **copy** | Copy sequence of characters from string |
| **find** | Find content in string |
| **rfind** | Find last occurrence of content in string |
| **find_first_of** | Find character in string |
| **find_last_of** | Find character in string from the end |
| **find_first_not_of** | Find absence of character in string |
| **find_last_not_of** | Find absence of character in string from the end |
| **substr** | Generate substring |
| **compare** | Compare strings |

# string Type

```
// declare empty string
string s1;
s1 = "01234567890123456789 0";
cout << "S1 is " << s1 << endl;

// declare and intialize string
string s2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
cout << "S2 is " << s2 << endl;

// create string from other string
// copy constructor
string s3 (s1);
cout << "S3 is " << s3 << endl;

// create string from other string
// starting from position, a number of character
string s4 (s3,11,5); // copy "12345" from s2
cout << "S4 is " << s4 << endl;

// create string and fill it with a character
string s5 (10,'*');
cout << "S5 is " << s5 << endl;
```

```
S1 is 0123456789012345678 90
S2 is ABCDEFGHIJKLMNOPQRSTUVWXYZ
S3 is 0123456789012345678 90
S4 is 12345
S5 is **********
```

# string Type

## Input and Output

```
cout << endl<< endl<< "INPUT:"<< endl<< endl;

//read until whitespace
cin >> s3;
cout << endl;
cout << "S3 is " << s3 << endl;

//read until delimiter '\n' (new line)
getline(cin, s4, '\n');
cout << "S4 is " << s4 << endl;

cout << endl<< endl<< "OUTPUT:"<< endl<< endl;

cout << endl;
cout << "S3 is " << s3 << endl;
cout << "S4 is \"" << s4 << "\" "<<endl;
```

```
INPUT:

111     222222

S3 is 111
S4 is   222222


OUTPUT:


S3 is 111
S4 is " 222222"
```

# string Type - Operations

```
cout << endl<< endl<< "OPERATORS:"<< endl<< endl;

//assignment
s3 = s1;
cout << "S3 is " << s3 << endl;

//concatenation
s4 = s1 + s2;
cout << "S4 is " << s4 << endl;

//assignment + concatenation
s5 += "000";
cout << "S5 is " << s5 << endl;

string s6;
s6+='a';
cout << "S6 is " << s6 << endl;
```

```
S1 is 01234567890123456789 0
S2 is ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
S3 is 01234567890123456789 0
S4 is 01234567890123456789 0ABCDEFGHIJKLMNOPQRSTUVWXYZ
S5 is **********000
S6 is a
```

# string Type - Functions

```
S3 is 01234567890123456789 0
S4 is 01234567890123456789 0ABCDEFGHIJKLMNOPQRSTUVWXYZ
S5 is **********000
S6 is a
```

## append

```
s3.append(s2);
cout << "S3 is " << s3 << endl;

s6.append(s1,0,10);
cout << "S6 is " << s6 << endl;
```

```
S3 is 01234567890123456789 0ABCDEFGHIJKLMNOPQRSTUVWXYZ
S6 is a0123456789
```

## assign

```
s4.assign(s2);
cout << "S4 is " << s2 << endl;

s5.assign("");
cout << "S5 is \"" << s5 << "\"" << endl;
```

```
S4 is ABCDEFGHIJKLMNOPQRSTUVWXYZ
S5 is ""
```

# string Type - Functions

```
S1 is 01234567890123456789 0
S2 is ABCDEFGHIJKLMNOPQRSTUVWXYZ
S4 is ABCDEFGHIJKLMNOPQRSTUVWXYZ
S5 is ""
```

## at and operator []

```
size_t position;
cout << "Spelled S1 is ";
for ( position = 0; position < s1.length(); ++position )
    cout << s1.at(position) << " ";
cout << endl;
cout << "Spelled S2 is ";
for ( position = 0; position < s2.length(); ++position )
    cout << s2[position] << " ";
cout << endl;
```

```
Spelled S1 is 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
Spelled S2 is A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

## empty

```
cout << "S4 empty is " << s4.empty() << endl;
cout << "S5 empty is " << s5.empty() << endl;
```

```
S4 empty is 0
S5 empty is 1
```

# string Type- Functions

## find

```
S1 is 01234567890123456789 0
S2 is ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
//int pos;
string::size_type pos ;
pos=s1.find("345");
cout << "Position 345 in S1 is " << pos<< endl;

pos=s1.find("ABC");
cout << "Position ABC in S1 is " << pos<< endl;

pos=s2.find('B');
cout << "Position B in S2 is " << pos<< endl;

pos=s1.find('0',2);
cout << "Position 0 in S1 after 2 is " << pos<< endl;
```

```
Position 345 in S1 is 3
Position ABC in S1 is 4294967295
Position B in S2 is 1
Position 0 in S1 after 2 is 10
```

## find_first_not_of

```
pos=s2.find_first_not_of("aeiouAEIOU");
cout << "First consonant in S2 is " << pos;
cout << " - '"<<s2[pos]<<"'" <<endl;;
```

```
First consonant in S2 is 1 - 'B'
```

# string Type - Functions

## insert

```
S3 is 0123456789012345678 90ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
cout << endl<< "Insert: "<<endl;
s3.insert(10, s2);
cout << "S3 is " << s3 << endl;
```

```
Insert:
S3 is 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ01234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

## erase

```
cout << endl<< "Erase: "<<endl;
s3.erase(10,36);
cout << "S3 is " << s3 << endl;
```

```
Erase:
S3 is 01234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

# string Type - Functions

## length

```
S3 is 01234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
cout << endl<< "Length: "<<endl;
pos=s3.length();
cout << "S3 has " << pos << " characters" <<endl;
```

```
Length:
S3 has 37 characters
```

## replace

```
cout << endl<< "Replace: "<<endl;
s3.replace(3, // start position in str
        5,  // how characters to replace
        s2,     // source for replacement
        0,     // start positio from source
        3);    // how chracter start from y1
cout << "S3 after replacing 34567 with ABC is " << s3 << endl;
```

```
Replace:
S3 after replacing 34567 with ABC is 012ABC890ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

# string Type - Functions

substr

```
S1 is 01234567890123456789 0
S2 is ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
cout << endl<< "Substr: "<<endl;
s3=s2.substr(10);
cout << "S3 is " << s3 << endl;

s4=s2.substr(10,5);
cout << "S4 is " << s4 << endl;

int n = s1.find("345");

s5 = s1.substr(n,7);
cout << "S5 is: " << s5 << endl;
```

```
S3 is KLMNOPQRSTUVWXYZ
S4 is KLMNO
S5 is: 3456789
```

# string Type - Functions

swap

```
S3 is KLMNOPQRSTUVWXYZ
S5 is: 3456789
```

```
cout << "S3 is " << s3 << endl;
cout << "S5 is: " << s5 << endl;
s3.swap(s5);
cout << "S3 is " << s3 << endl;
cout << "S5 is: " << s5 << endl;
```

```
S3 is 3456789
S5 is: KLMNOPQRSTUVWXYZ
```

# string Type - Functions

## compare

```
cout << "S1 compare S2 is "<<s1.compare(s2)<<endl;
cout << "S3 compare S1 is "<<s3.compare(s1)<<endl;
cout << "S1 compare S3 is "<<s1.compare(s3)<<endl;
cout << "S1 compare S1 is "<<s1.compare(s1)<<endl;
```

```
S1 is 01234567890123456789 0
S2 is ABCDEFGHIJKLMNOPQRSTUVWXYZ
S3 is 0123456789
```

```
S1 compare S2 is -1
S3 compare S1 is -1
S1 compare S3 is 1
S1 compare S1 is 0
```

## comparison operators

```
s1 = "Hello";
s2 = "Hi";
if (s1 < s2)
    cout << "S1<S2 is true" << endl;
else
    cout << "S1<S2 is false" << endl;
if (s1 > "Hen")
    cout << "S1>\"Hen\" is true" << endl;
else
    cout << "S1>\"Hen\" is false" << endl;
if (s1 == "hello")
    cout << "S1==\"hello\" is true" << endl;
else
    cout << "S1==\"hello\" is false" << endl;
if (s2 >= "Hill")
    cout << "S1>=\"Hill\" is true" << endl;
else
    cout << "S1>=\"Hill\" is false" << endl;
```

```
S1 is Hello
S2 is Hi
```

```
S1<S2 is true
S1>"Hen" is false
S1=="hello" is false
S1>="Hill" is false
```

# Example: Strings and Searches

- Create a string with all the digits between 0 and 9, without hardcoding it

```
char c;
string S;
for (c = '0'; c <= '9'; c++)
        S += c;
cout << "\nThe string S is " << S;
```

- Append 10 more of 0123456789 to the string s

```
int k;
for (k = 1; k <= 10; k++)
        S.append("0123456789");
cout << "\nThe string S is " << S;
```

```
The string S is 0123456789
The string S is 01234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
```

# Example: Strings and Searches

- Search for “123" in S

```
//search for 123 in S
Search = "123";
Pos = S.find(Search);
//check if we found it
if (Pos != string::npos)
        cout << "\n\n" << Search << " is at position " << Pos << " in " << S;
else
        cout << "\n\n" << Search << " is not in " << S;
```

```
123 is at position 1 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
```

- search for the second “123" in S

```
Search = "123";
Pos = S.find(Search, Pos+1);
//check if we found it
if (Pos != string::npos)
        cout << "\n\n" << Search << " is at position " << Pos << " in " << S;
else
```

```
123 is at position 1 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789

123 is at position 11 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
```

# Example: Strings and Searches

- Find how many times/all occurences of "123" in S

```
int Number = 0;
Search = "123";
Pos = -1;
do
{
    Pos = S.find(Search, Pos + 1);
    //check if we found it
    if (Pos != string::npos)
    {
        cout << "\n\n" << Search << " is at position " << Pos << " in " << S;
        Number++;
    }
} while (Pos != string::npos);

if (Number==0)
    cout << "\n\n" << Search << " is not in " << S;
else
    cout << "\n\n" << Search << " is in " << S << "\n" << Number << " times";
```

# Example: Strings and Searches

```
123 is at position 1 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
123 is at position 11 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
123 is at position 21 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
123 is at position 31 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
123 is at position 41 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
123 is at position 51 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
123 is at position 61 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
123 is at position 71 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
123 is at position 81 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
123 is at position 91 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
123 is at position 101 in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
123 is in 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678911 times
```

# Example: Strings and Searches

```
string A;
char c;
for (c = 'A'; c <= 'Z'; c++)
        A+=c;
cout << "\nThe string A is " << A;
```

```
The string A is ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
//find first consonant in A
Pos = A.find_first_not_of("AEIOU");
cout << "\n\nThe first consonant ("<<A[Pos]<<") is at position " << Pos;
//find last vowel in A
Pos = A.find_last_of("AEIOU");
cout << "\n\nThe last vowel (" << A[Pos] << ") is at position " << Pos;
```

```
The first consonant (B) is at position 1

The last vowel (U) is at position 20
```

# Summary

- Enumeration types
- Anonymous types
- Typedef statements
- Namespaces
- Strings