Semester 2 2024

Astroinformatics II

Tutorial 4: Introduction to C/C++ (II)

In this tutorial session, we will practice the usage of the GCC and coding in C++. It is also recommended that you use VSCode and the debugger available within VSCode.

1 Data Types

C++ uses a range of built-in (fundamental) data types.
The `sizeof()` function gives us the size (in bytes) of a variable in these types.

Task: Write a in C++ program to find the size of fundamental data types.
The output should look like the following:

```
Find Size of fundamental data types :
------------------------------------------
The sizeof(char) is : 1 bytes
The sizeof(short) is : 2 bytes
The sizeof(int) is : 4 bytes
The sizeof(long) is : 8 bytes
The sizeof(long long) is : 8 bytes
The sizeof(float) is : 4 bytes
The sizeof(double) is : 8 bytes
The sizeof(long double) is : 16 bytes
The sizeof(bool) is : 1 bytes
```

2 Arrays

Generally, arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
To declare an array in C++, we define the type, specify the name of the array followed by square brackets and specify the number of elements it should store.

example:
`string types[3];`

We insert values into it like in the following example:
`string types[3] = "elliptical", "spiral", "irregular";`

The following code finds both maximum and minimum numbers in a numerical array.
For example, for the input
`arr[] = 22, 12, 45, 48, 22, 18`
the output is:

```
cout << "Minimum element of array: 48
cout << "Maximum element of array: 12
```

```cpp
// C++ program to find minimum (or maximum) element
// in an array.

using namespace std;

int getMin(int arr[], int n)
{
    return *min_element(arr, arr + n);
}

int getMax(int arr[], int n)
```

```
{
    return *max_element(arr, arr + n);
}

int main()
{
    int arr[] = { 12, 1234, 45, 67, 1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Minimum element of array: " << getMin(arr, n) << " ";
    cout << "Maximum element of array: " << getMax(arr, n);
    return 0;
}

```

Task:
a) Try to understand what the above program does, line by line. You can also use the debugger in VSCode for this, to step through the code line by line.
b) Write a similar program to calculate the average of all the elements in an array.
c) Modify your program so that it writes the result into a file.

3 Control Structures

C/C++ is using similar control structures to other programming languages like Python. As an introduction, we are covering only three here: `if-else` statements, `while` loops and `for` loops.

3.1 `if-else` statements

The `if-else` statement gives programs the capability to make decisions by evaluating statements. The `else` clause is optional.

syntax:

```
if (condition) {
  // block of code if condition is true
}
else {
  // block of code if condition is false
}
```

An **example:**

```
// Program to check whether an integer is positive or negative
// This program considers 0 as a positive number

#include <iostream>
using namespace std;

int main() {

  int number;

  cout << "Enter an integer: ";
  cin >> number;

  if (number >= 0) {
    cout << "You entered a positive integer: " << number << endl;
  }
  else {
    cout << "You entered a negative integer: " << number << endl;
  }

  cout << "This line is always printed.";

  return 0;
}

```

The if...else statement is used to execute a block of code among two alternatives. However, if we need to make a choice between more than two alternatives, we extend it to a `if...else if...else` statement.
syntax:

```
if (condition1) {
  // code block 1
}
else if (condition2){
  // code block 2
}
else {
  // code block 3
}
```

Task:
Write a small program with an if...else statement, and use the debugger in VSCode.

3.2 `while` loops

The `while` loop is the fundamental approach to repetitive sequences. It loops through a block of code as long as a specified condition is true.

syntax:

```
while (condition) {
  // code block to be executed
}
```

In the **example** below, the code in the loop will run, over and over again, as long as a variable `i` is less than 5:

```
int i = 0;
while (i < 5) {
  cout << i << "\n";
  i++;
}
```

3.3 `for` loops

Although `while` loops are the basic loop structure, `for` loops are the enhanced version that allows for more flexibility.
When you know exactly how many times you want to loop through a block of code, use the `for` loop.

syntax:

```
for (statement 1; statement 2; statement 3) {
// code block to be executed
}
```

It does the following:
Statement 1 is executed (one time) before the execution of the code block.
Statement 2 defines the condition for executing the code block.
Statement 3 is executed (every time) after the code block has been executed.
The `example` below will print the numbers 0 to 6:

```
for (int i = 0; i < 6; i++) {
  cout << i << "\n";
}
```