ESc 101: Fundamentals of Computing

Lecture 5

Jan 11, 2010

Outline

Reading Data from Input

- When a program needs to read data from input device (keyboard for example), it asks the OS to read the input and transfer it to appropriate memory location.
- OS provides only one mode of reading the input: symbol-by-symbol.
- Suppose the program wishes to read a number from input.
- Then it must do the following:
  - Read the symbols from input invoking the OS repeatedly until a non-digit symbol is encountered.
  - Convert the sequence of digits read to a number.

Reading Data from Input

- When a program needs to read data from input device (keyboard for example), it asks the OS to read the input and transfer it to appropriate memory location.
- OS provides only one mode of reading the input: symbol-by-symbol.
- Suppose the program wishes to read a number from input.
- Then it must do the following:
  - Read the symbols from input invoking the OS repeatedly until a non-digit symbol is encountered.
  - Convert the sequence of digits read to a number.

Reading Data from Input

- When a program needs to read data from input device (keyboard for example), it asks the OS to read the input and transfer it to appropriate memory location.
- OS provides only one mode of reading the input: symbol-by-symbol.
- Suppose the program wishes to read a number from input.
- Then it must do the following:
  - Read the symbols from input invoking the OS repeatedly until a non-digit symbol is encountered.
  - Convert the sequence of digits read to a number.

Reading Data from Input

- When a program needs to read data from input device (keyboard for example), it asks the OS to read the input and transfer it to appropriate memory location.
- OS provides only one mode of reading the input: symbol-by-symbol.
- Suppose the program wishes to read a number from input.
- Then it must do the following:
  - Read the symbols from input invoking the OS repeatedly until a non-digit symbol is encountered.
  - Convert the sequence of digits read to a number.

Reading Data from Input

- When a program needs to read data from input device (keyboard for example), it asks the OS to read the input and transfer it to appropriate memory location.
- OS provides only one mode of reading the input: symbol-by-symbol.
- Suppose the program wishes to read a number from input.
- Then it must do the following:
  - Read the symbols from input invoking the OS repeatedly until a non-digit symbol is encountered.
  - Convert the sequence of digits read to a number.

The View of OS

- For OS, reading input means reading one symbol.
- Similarly, displaying output means displaying one symbol.
- The rest must be taken care of by the program.

The View of OS

- For OS, reading input means reading one symbol.
- Similarly, displaying output means displaying one symbol.
- The rest must be taken care of by the program.

The View of OS

- For OS, reading input means reading one symbol.
- Similarly, displaying output means displaying one symbol.
- The rest must be taken care of by the program.

The View of OS

- ASCII code provides a mapping of symbols to numbers.
- These numbers, written as binary sequences of eight bits (one byte), are stored in memory.
- It is the job of a program to assign appropriate interpretation to the symbols.

The View of OS

- ASCII code provides a mapping of symbols to numbers.
- These numbers, written as binary sequences of eight bits (one byte), are stored in memory.
- It is the job of a program to assign appropriate interpretation to the symbols.

The View of OS

- ASCII code provides a mapping of symbols to numbers.
- These numbers, written as binary sequences of eight bits (one byte), are stored in memory.
- It is the job of a program to assign appropriate interpretation to the symbols.

Outline

Basic Structure

```
main()
<statement-block>
```

Statement Block

<statement-block> has the form:

```
{
    <variable declarations>
    <statements>
}
```

Statement Block

<statement-block> has the form:

```
{
    <variable declarations>
    <statements>
}
```

<variable declarations> reserve memory locations and give names to them. These are called variables.

Statement Block

<statement-block> has the form:

```
{
    <variable declarations>
    <statements>
}
```

<statements> is a sequence of instructions to be executed.

Statement Block

<statement-block> has the form:

```
{
    <variable declarations>
    <statements>
}
```

Declarations of variables can mix with statements, however, it is advisable to declare all the variables before the statements in a block.

Variable Declarations

<variable declarations> is a sequence of declarations:

```
<declaration-1>
<declaration-2>
      :
      :
<declaration-n>
```

Declaring a Variable

A single declaration has the form:

```
<type>  <name>;
```

<type> denotes the type of data that the memory location stores.
<name> is the variable, equivalently, it is the name assigned to the memory location.
The semi-colon at the end denotes the end of the declarations.

Declaring a Variable

A single declaration has the form:

```
<type>  <name>;
```

<type> denotes the type of data that the memory location stores.

<name> is the variable, equivalently, it is the name assigned to the memory location.

The semi-colon at the end denotes the end of the declarations.

Declaring a Variable

A single declaration has the form:

```
<type>  <name>;
```

<type> denotes the type of data that the memory location stores.
<name> is the variable, equivalently, it is the name assigned to the memory location.

The semi-colon at the end denotes the end of the declarations.

Declaring a Variable

A single declaration has the form:

```
<type>  <name>;
```

<type> denotes the type of data that the memory location stores.
<name> is the variable, equivalently, it is the name assigned to the memory location.
The semi-colon at the end denotes the end of the declarations.

Types

There are several pre-defined types in C:

`int`: represents integers

`char`: represents a symbol, or character

`float`: represents a fractional number

`double`: also represents a fractional number, but with more space for achieving better precision

There are some variations of these types which we will discuss later.

Types

There are several pre-defined types in C:

`int`: represents integers

`char`: represents a symbol, or character

`float`: represents a fractional number

`double`: also represents a fractional number, but with more space for achieving better precision

There are some variations of these types which we will discuss later.

Types

There are several pre-defined types in C:

`int`: represents integers

`char`: represents a symbol, or character

`float`: represents a fractional number

`double`: also represents a fractional number, but with more space for achieving better precision

There are some variations of these types which we will discuss later.

Types

There are several pre-defined types in C:

`int`: represents integers

`char`: represents a symbol, or character

`float`: represents a fractional number

`double`: also represents a fractional number, but with more space for achieving better precision

There are some variations of these types which we will discuss later.

Types

There are several pre-defined types in C:

`int`: represents integers

`char`: represents a symbol, or character

`float`: represents a fractional number

`double`: also represents a fractional number, but with more space for achieving better precision

There are some variations of these types which we will discuss later.

Types

There are several pre-defined types in C:

`int`: represents integers

`char`: represents a symbol, or character

`float`: represents a fractional number

`double`: also represents a fractional number, but with more space for achieving better precision

There are some variations of these types which we will discuss later.

Space Reserved for Different Types

`int`: 4 bytes

`char`: 1 byte

`float`: 4 bytes

`double`: 8 bytes

Space Reserved for Different Types

`int`: 4 bytes

`char`: 1 byte

`float`: 4 bytes

`double`: 8 bytes

Space Reserved for Different Types

`int`: 4 bytes

`char`: 1 byte

`float`: 4 bytes

`double`: 8 bytes

Space Reserved for Different Types

`int`: 4 bytes

`char`: 1 byte

`float`: 4 bytes

`double`: 8 bytes