# Computer Science I
## CS 135

## 1. Introduction to Programming

   a.   **How to Develop a Program**

   b.   **Writing Pseudocode**

   c.   **First Elements of C++**

- ✓ The basics of a C++ program
- ✓ Data types
- ✓ Arithmetic operators
- ✓ Expressions
- ✓ Variables
- ✓ Type casting
- ✓ ASCII characters
- ✓ Input and output

   d.   **Looking Under the Hood**

# 1.c  First Elements of C++

## The basics of a C++ program

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

int main()
{
        /* read data */
        cout << "Enter 3 numbers ";
        cin >> x >> y >> z;

        sum = x + y + z;      // calculate
        avg = sum/3.0;        // average

        cout << endl;
        cout << "The average is ";
        cout << avg << endl;

        return 0;
}
```

# 1.c  First Elements of C++
## The basics of a C++ program

➢ **Structure of a C++ program**

  ✓ a C++ program is a collection of one or more subprograms, called **functions**

  ✓ a subprogram or a function is a collection of statements that accomplishes something when executed

  ✓ every C++ program has at least one function called `main`

  ✓ the smallest individual unit of a program is called a "token"

➢ **Tokens of a C++ program**

  ✓ special symbols

  ✓ word symbols

  ✓ identifiers

# 1.c  First Elements of C++
## The basics of a C++ program

➢ **Special symbols**

   ✓ mathematical symbols:    **+**     **–**     **\***     **/**    . . .

   ✓ punctuation marks:     **.**     **;**     **?**     **,**    . . .

   ✓ two-character symbols:   **<=**    **!=**    **==**    **>=**   . . .

   ✓ etc.

➢ **Word symbols**

   ✓ **int**, **float**, **char**, **void**, **return**, etc.

   ✓ also called reserved words or **keywords**, as they belong to the language

      ■ cannot be redefined and reused

      ■ are always lowercase

# 1.c First Elements of C++
## The basics of a C++ program

➢ **Identifiers**

  ✓ **main, x, y, sum, student_name,** etc.

  ✓ identifiers are user-created <u>names</u> of things that appear in programs (variables, constants, functions, etc.):

  - consist of letters, digits, and the underscore character **_**

  - must begin with a letter or underscore

  - are case sensitive

  ✓ should be meaningful: **student_name** better than **sdtnm**

  ✓ there exists predefined identifiers, such as **cout** and **cin**

  - unlike reserved words, predefined identifiers may be redefined, but not a good idea as it can be confusing

# 1.c  First Elements of C++
## The basics of a C++ program

➢ **Examples of legal and illegal identifiers**

| Identifier | Legal | Illegal | |
|---|---|---|---|
| `first name` | | ✗ | (space break) |
| `first_name` | ✓ | | |
| `firstName` | ✓ | | |
| `1st_name` | | ✗ | (first char is digit) |
| `first-name` | | ✗ | (hyphen is symbol) |
| `_name1` | ✓ | | |

# 1.c  First Elements of C++
## The basics of a C++ program

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

int main()
{
        /* read data */
        cout << "Enter 3 numbers ";
        cin >> x >> y >> z;

        sum = x + y + z;     // calculate
        avg = sum/3.0;       // average

        cout << endl;
        cout << "The average is ";
        cout << avg << endl;

        return 0;
}
```

char symbols
keywords
identifiers
literals
strings
comments

# 1.c  First Elements of C++
## Data types

➢ **Different data types for different programs**

   ✓ for example, some programs work with numbers (scientific calculation), some other programs manipulate names (alphabetizing lists) and some use both (grading)

   ✓ numbers and words are distinct **data types**

➢ **Categories of data types in C++**

   ✓ simple data type

      ▪ <u>integral</u> data type = integer numbers (without a decimal)

      ▪ <u>floating-point</u> data type = decimal numbers

      ▪ <u>enumeration</u> data type = programmer-created type

   ✓ structured data type & pointers

# 1.c  First Elements of C++
## Data types

➢ **Integral data types**

     ✓ **int**  represents <u>integer numbers</u>, for example:

         ▪ **0**, **37**, **-45**, **12500**  (no comma inside number)

         ▪ minimum: **-2147483648**

         ▪ maximum: **2147483647**

     ✓ **char**  represents <u>any single character</u>, for example:

         ▪ **'a', '2', 'B', '$', ' '**  (in single quotes)

         ▪ also represents small integers between **-128** and **127**

     ✓ **bool**  represents <u>logical (Boolean) values</u> and can be only

         ▪ **true** or **false**  (these are keywords)

# 1.c  First Elements of C++
## Data types

➢ **Floating-point data types**

    ✓ **`float`** represents <u>real numbers</u>, for example:

-     **`75.924`**, **`-1.482`**, **`0.0018`**, **`180.00`**

-     **`7.5924e1`**, **`-1.482e0`**, **`1.8e-3`**, **`1.8e2`**
  (in scientific notation)

-     maximum of 6 or 7 significant digits

-     bounds: **`-3.4e38`** and **`3.4e38`**

    ✓ **`double`** also represent real numbers but with <u>double precision</u> (more significant digits and larger interval)

-     maximum of 15 significant digits

-     bounds: **`-1.7e308`** and **`1.7e308`**

# 1.c  First Elements of C++

## Arithmetic operators

➤ **The C++ arithmetic operators are**

 ✓  addition          **+**

 ✓  subtraction       **−**

 ✓  multiplication    **\***

 ✓  division          **/**

 ✓  remainder         **%**          (mod operator: 11 % 3 = 2)

 ✓  **+**, **−**, **\*** and **/** can be used with both integral and floating-point data types

➤ **Two types of operators**

 ✓  <u>unary</u> operators have only one operand:  **−** ☐  or  **+** ☐

 ✓  <u>binary</u> operators have two operands:  ☐ **\*** ☐  or  ☐ **+** ☐

# 1.c  First Elements of C++
## Arithmetic operators

➢ **Order of precedence**

- ✓ unary operators are evaluated first

- ✓ then all operations inside parentheses **(** **)** are evaluated next

- ✓ then **\*** , **/** and **%** are at the same level of precedence and are evaluated next

- ✓ finally **+** and **-** are at the same level of precedence and are evaluated last

- ✓ when operators are on the same level, evaluation is performed from left to right

- ✓ example: **3 + 7 \* 6** evaluates to **45**

- ✓ example: **(3 + 7) \* 6** evaluates to **60**

# 1.c  First Elements of C++

Expressions

➢ **Integer expressions**

  ✓ all operands are integer

  ✓ the result is an integer

  ✓ example: `10 / 4` evaluates to `2`

  ✓ example: `72 - 7 * (-60 % 8)` evaluates to `100`

➢ **Floating-point expressions**

  ✓ all operands are floating-point

  ✓ the result is floating-point

  ✓ example: `10.0 / 4.0` evaluates to `2.25`

  ✓ example: `72.36 - 0.09 * 4.0` evaluates to `72.0`

# 1.c  First Elements of C++
## Expressions

➢ **Mixed expressions**

  ✓ operands are of different data types, integer and floating-point

  ✓ example: `5.4 * 2 - 13.6 + 21 / 6`

➢ **Evaluation rules for binary operators**

  ✓ if both operands are integer, then result is integer

  ✓ if both operands are floating-point, then result is floating-point

  ✓ if one operand is integer and the other floating-point, then the integer is changed to floating-point and the result is floating-point

  ✓ example: `5.4 * 2 - 13.6 + 21 / 6` yields `0.2`

  `10.8`                              `3`

# 1.c  First Elements of C++
## Variables

➢ **Variables allow for formulas and generic processing**

    ✓ variables are like "unknowns" in math, for example:

- ▪ `surface = length * width;`

- ▪ `total = price + (price * 0.08);`

- ▪ `cout << "The name is " << name;`

➢ **Variable assignment**

    ✓ form of assignment statement: `variable =` *expression*`;`

    ✓ *expression* is evaluated and its value is assigned to the variable on the left side (**no** expression on the left side!)

    ✓ `=` is called the assignment operator

    ✓ for example: `i = i + 1` increases variable `i` by 1

# 1.c  First Elements of C++
## Variables

➢ **Compound operators allow more concise assignments**

  ✓ most compound operators are two-character symbols: an arithmetic operator followed by the equal sign (no space)

  ▪ compound addition  **+=**  and subtraction  **-=**

  ▪ compound multiplication  **\*=**  and division  **/=**

  ▪ compound remainder  **%=**

  ✓ they are shortcuts used in <u>some</u> assignments where the same variable is on both sides of the equal sign, for example:

  ▪ `i += 1`  is the same as  `i = i + 1`

  ▪ `x *= y - 3`  is the same as  `x = x * (y - 3)`

  ▪ . . . but is *not* the same as  `x = x * y - 3`  (this last assignment doesn't have an easy compound equivalent)

# 1.c  First Elements of C++
## Variables

➢ **Variables must be declared before they can be used**

    ✓ form of the declaration statement: *data-type  variable_name;*

-        `int age;`

-        `float surface;`

-        `char name_initial;`

     ✓ once declared, variables can be assigned values using an assignment statement

     ✓ variable declarations can be placed anywhere in a function, but generally at the beginning

     ✓ variables of the same data type can be grouped together in a single declaration

# 1.c  First Elements of C++
## Variables

➢ **Variables can be initialized at declaration or later**

all declarations
at the beginning

```
...

int num1, num2;   // only declared
int num3 = 100;   // declared
                  // and initialized

float average, percentage;  // declared

num1 = 7;         // initialized
num2 = 20;        // initialized

average = (num1 + num2)/2;  // assigned
percentage = average/100;   // assigned

...
```

✓ result: **average** is **13.0**, **percentage** is **0.13**

# 1.c  First Elements of C++
## Variables

```
...

int num1, num2;   // only declared
int num3 = 100;   // declared
                  // and initialized

float average, percentage;  // declared

num1 = 7;          // initialized
num2 = 20;         // initialized

average = (num1 + num2)/2.0;// assigned
percentage = average/100;   // assigned

...
```

✓ result: **average** is **13.5**, **percentage** is **0.135**

# 1.c  First Elements of C++
## Variables

```
...

int num1, num2;   // only declared
int num3 = 100;   // declared
                  // and initialized


int average, percentage;   // declared


num1 = 7;            // initialized
num2 = 20;           // initialized


average = (num1 + num2)/2;   // assigned
percentage = average/100;    // assigned


...
```

✓ result: **average** is **13** , **percentage** is **0**

# 1.c  First Elements of C++

Type casting

➢ **You can also explicitly convert one type into another**
- ✓ this is called "type casting" or "type conversion" and can be written in two ways:
  - ▪ `age = int(24.7);`   yields a value of 24
  - ▪ `age = (int)24.7;`
- ✓ ex: from floating-point to integer → drop the decimal part
  - ▪ `tax_payment = int(dollar_cents);`
- ✓ ex: from integer to floating-point → preserve precision
  - ▪ `mean_age = float(total_age)/num;`
- ✓ ex: from character to integer → alphabetize
  - ▪ `rank = int('a');`   yields a value of 97

# 1.c  First Elements of C++

<span style="background-color:#fdfdc0">ASCII characters</span>

➢ **American Standard Code for Information Interchange**

  ✓ the ASCII standard is character set and a character encoding based on the Roman alphabet, as used in modern English

  ✓ contains printable characters and control characters

  ✓ **printable** characters are

- alphabetical characters: lowercase and uppercase letters
- numerical characters: digits from **0** to **9**
- all symbol characters: **!** , **@** , **#** , **$** , **%** , **^** , **&** , **\*** , **(** , **)** , etc.

  ✓ **control** (nonprintable characters) include

- line feed, carriage return, end of file, escape, delete, etc.

# 1.c  First Elements of C++
## ASCII characters

➢ **American Standard Code for Information Interchange**

 ✓  the code of a character is an integer between 0 and 127

 ✓  example: `int('A')` gives `65`, and `char(65)` gives `'A'`

# 1.c  First Elements of C++

➢ **Reading variables from the keyboard**

   ✓ `cin` takes input from the keyboard and stores it in a designated variable

   ✓ `cin` is used together with `>>` (the "extraction" operator) and the variable name to gather the input value:

      ▪ `cin >> num1;`

   ✓ using more than one variable in a `cin` statement allows more than one value to be read at a time:

      ▪ `cin >> num1 >> num2;`

      ▪ this reads two integers from the keyboard and inputs them into variables `num1` and `num2` respectively

# 1.c  First Elements of C++
## Input and output

➢ **Displaying messages and values to the screen**

 ✓ **cout** takes strings and values (expressions, variables or literals) and outputs them to the screen

 ✓ **cout** is used together with **<<** (the "insertion" operator) and the string or value:

 ▪ **cout << average;**

 ✓ outputs can be combined into one **cout** statement:

 ▪ **cout << "The result is " << num;**

 ✓ **endl** causes a new line and flushes the output buffer:

 ▪ **cout << (2 + 2) << endl;**

# 1.c  First Elements of C++
## Input and output

prompt message
& read values

output result

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

int main()
{
        /* read data, calc and print */
        cout << "Enter 3 numbers ";
        cin >> x >> y >> z;

        sum = x + y + z;        // calculate
        avg = sum/3.0;          // average

        cout << endl;
        cout << "The average is ";
        cout << avg << endl;

        return 0;
}
```

# Computer Science I
## CS 135

## 1. Introduction to Programming

### a. How to Develop a Program

### b. Writing Pseudocode

### c. First Elements of C++

- ✓ The basics of a C++ program
- ✓ Data types
- ✓ Arithmetic operators
- ✓ Expressions
- ✓ Variables
- ✓ Type casting
- ✓ ASCII characters
- ✓ Input and output

### d. Looking Under the Hood