

C++ BASICS



C++ Overview:

Common set of basic features shared by a wide range of programming languages

- Built-in types (integers, characters, floating point numbers, etc.)
- Variables (names or identifiers for the entities)
- Expressions and statements to manipulate values of variables
- Control-flow constructs (if, for, while, etc.)
- Functions
- Programmer-defined types (struct, class, etc.)
- Library functions

C++ EXAMPLE

Sample C++ Program :

```
//header comment
#include <iostream> //standard library for i/o
#include <string> // always include this when you use the string class
using namespace std;

int main()
{
    int numPies, numSlices, total;
    cout << "Enter the number of pies: ";
    cin >> numPies;
    cout << "Enter the number of slices per pie: ";
    cin >> numSlices;
    total = numPies * numSlices;
    cout << "If you have "<< numPies;
    cout << " pies with "<< numSlices << " slices each\n";
    cout << "you will have "<< total<< " total slices!\n";
    cout << endl;
    return 0;
}
```

MEMORY

memory cell

an individual storage location in memory

address of a memory cell

the relative position of a memory cell in the computer's main memory

contents of a memory cell

the information stored in a memory cell,
either a *program instruction* or *data*

stored program concept

a computer's ability to store program
instructions in main memory for execution

The diagram illustrates memory cells and their addresses. It features a table with two columns: 'Value' and 'Variable name'. The table contains four rows with data and three empty rows at the bottom. To the left of the table, four orange arrows point to the first four rows. Each arrow is accompanied by a red address label and a green comment. The first row has the value '25' and variable name 'number', with an arrow pointing to it from the address '&number' (comment: '// "address of" number'). The second row has the value '15' and variable name 'number2', with an arrow pointing to it from the address '&number2' (comment: '// "address of" number2'). The third row has the value ''X'' and variable name 'letter', with an arrow pointing to it from the address '&letter' (comment: '// "address of" letter'). The fourth row has the value '55.5' and variable name 'amount', with an arrow pointing to it from the address '&amount' (comment: '// "address of" amount').

Value	Variable name
25	number
15	number2
'X'	letter
55.5	amount

Comments in C++

```
/***SAMPLE PROGRAM HEADER*****
```

```
STUDENT NAME
```

```
Due Date:
```

```
Course:
```

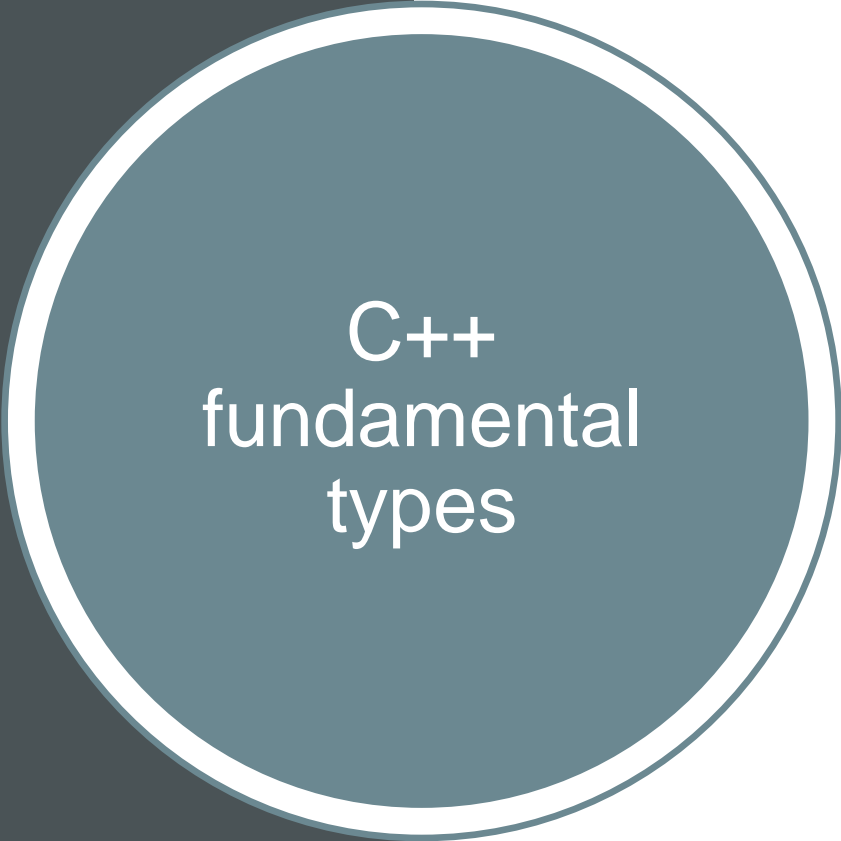
```
Assignment:
```

```
Professor: Sorgente
```

```
Description: (Your program description goes here  
//what it does--In the program we processed....*/
```

```
#include <iostream> //standard library for i/o  
using namespace std;
```

```
int main()  
{  
    return 0;  
}
```



C++ fundamental types

C++ defines a set of primitive types

- Void type
- Boolean type
- Integer types
- Character types
- Floating point types

All other types are composed of these fundamental types in some way

- Variables (identifier name) are used to name and store data
- Variables must be declared before they can be used
- Value is the data in the variable

Data type examples:

```
int number;
```

```
double average;
```

```
char letter;
```

```
string name; //#include <string>
```

```
bool isWorking = true;
```

NAMING VARIABLES (IDENTIFIERS)

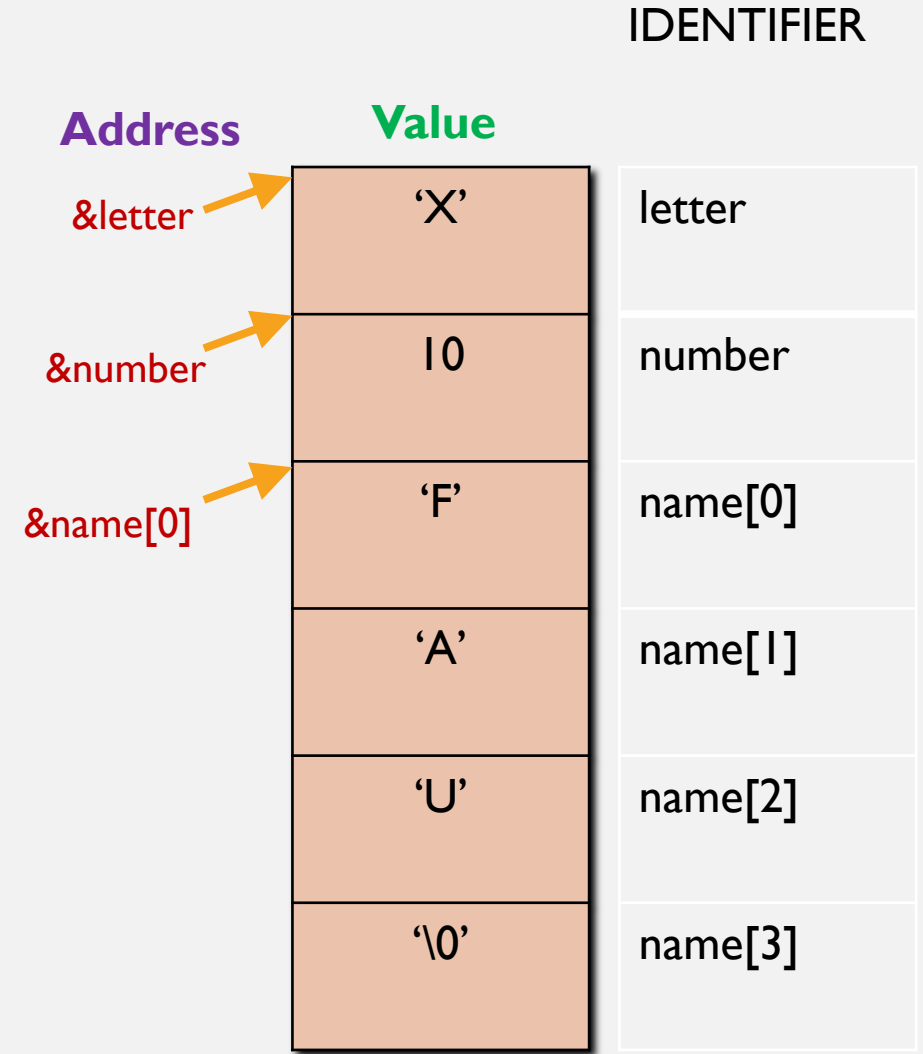
- keywords cannot be used to name a variable, (predefined meaning in C++)
- use meaningful variable names
- variable names must begin with a letter or an underscore _
- All the rest of a variable name must be a letter, number, or underscore _
- Camelback or camelCase naming example: numTrolls
- Variable names in a simple declaration may optionally be followed by an initializer

NAMING VARIABLES (IDENTIFIERS)

- The computer implements variables as memory locations
 - the value is stored at that location
 - assigned to the identifier.
- Assignment statements set the value of the variable

```
int number;  
char letter;  
string name; //#include <string>
```

```
number = 10;  
letter = 'X';  
name = "FAU";
```



SCOPE

- Names in a C++ program are valid only within their scope (block – curly braces)
- The scope of a name begins at its point of declaration

```
int main( )  
{  
    int x = 21, b = 10;  
    {  
        int y = 5; //both x and y are accessible  
    }  
  
    int z = x + y; //y is NOT accessible outside of the scope  
  
    return 0;  
}
```

void type

The void type has no values

- Identified by the C++ keyword void
- No objects of type void are allowed
- Mainly used as a return type for functions that do not return any value

bool type

The bool type can hold two values

- Identified by the C++ keyword bool
- Represents the truth values true and false

Integer (int) type

The integer types represent integral values

- Identified by the C++ keyword `int`

Signed modifiers

- signed integers will have signed representation (i.e. they can represent negative numbers)
- unsigned integers will have unsigned representation (i.e. they can only represent non-negative numbers)

Size modifiers

- short integers will be optimized for space (at least 16 bits wide)
- long integers will be at least 32 bits wide
- long long integers will be at least 64 bits wide

character (char) type

Character types represent character codes and (to some extent) integral values

Any character from the source character set except single quote, backslash and newline

Example: 'a', 'b', '\$'

Escape sequences, e.g. '\', '\\', '\n'

Floating point types

Floating point types of varying precision

- **float** usually represents IEEE-754 32 bit floating point numbers
- **double** usually represents IEEE-754 64 bit floating point numbers
- long double is a floating point type with extended precision (varying width depending on platform and OS, usually between 64 bit and 128 bit)

Floating point types may support special values

- Infinity
- Negative zero
- Not-a-number

DATA TYPES

- Variables (identifier name) are used to name and store data
- Value is the data in the variable

Data type examples:

```
int number;
```

```
double average;
```

```
char letter;
```

```
string name; // #include <string>
```

```
bool isWorking = true;
```


DATA TYPES AND TYPE CASTING

- It is poor style to try to declare a variable of one data type and assign it to another data type.

- Use type casting

```
int total = 100 , numWinners = 10;  
double yourWinnings;
```

```
//get values for the variables  
//one operand should be a double to perform double math  
yourWinnings = static_cast<double>(total) / numWinners;
```

```
yourWinnings = (double)total / numWinners;
```

CONST

- Const objects cannot be modified

```
//header comment here
#include <iostream> //standard library for i/o
using namespace std;

const int MAX_TIRES = 3;
const double PI = 3.14159;

int main()
{
    return 0;
}
```

NAMING CONSTANTS

```
/****SAMPLE PROGRAM HEADER*****
```

```
STUDENT NAME
```

```
Due Date:
```

```
Course:
```

```
Assignment:
```

```
Professor: Sorgente
```

```
Description: (Your program description goes here -- what it does--In the program we processed....
```

```
*****/
```

```
//header comment here
```

```
#include <iostream> //standard library for i/o
```

```
using namespace std;
```

```
const int MAX_TIRES = 3;
```

```
const double PI = 3.14159;
```

```
int main()
```

```
{
```

```
    return 0;
```

```
}
```

COUT AND CIN

- **cin** is an input stream connected to the keyboard
- **cout** is an output stream connected to the screen

```
cout << "\n"; // in quotes
cout << endl; // no quotes
```

```
//set the number of decimal places for doubles
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2); //use any number here for the number of decimal places
```

```
int numPies, numSlices;
```

```
//insertion operator
```

```
cout << "Enter the number of pies and slices: ";
```

```
//extraction operator (extract data from input)
```

```
cin >> numPies >> numSlices; //get 2 inputs in one line is not recommended
```

ENUMERATION TYPE

Enumeration type – values are defined by a list of constants of type int

List of declared constants

- *Two or more named constants can have the same numerical value*
- *If you do not specify any numerical values, it will start with 0*

```
//declaring constant
const int SUMMER_MONTHS = 3;

// Defining enum month length
enum MonthLength {
    Jan_length = 31, Feb_length = 28, Mar_length = 31, Apr_length = 30,
    May_length = 31, Jun_length = 30, Jul_length = 31, Aug_length = 31,
    Sep_length = 30, Oct_length = 31, Nov_length = 30, Dec_length = 31 };

// Defining enum month placement
enum Month {
    noMonth, Jan, Feb, Mar, Apr,
    May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };

int main()
{
    cout << "There are " << SUMMER_MONTHS << " months in the summer";
    cout << "\n\nJune is month " << Jun << ", and has " << Jun_length << " days\n";
    cout << "July is month " << Jul << ", and has " << Jul_length << " days\n";
    cout << "August is month " << Aug << ", and has " << Aug_length << " days\n";
    return 0;
}
```

C++ provides a rich set of operators

- Operators and operands can be composed into expressions
- Most operators can be overloaded for custom types

Fundamental expressions

- Variable names
- Literals

Operators act on a number of operands

- Unary operators: negation (-), address-of (&), dereference (*)
- Binary operators: equality (==), multiplication (*)
- Ternary operator: $x ? y : z$

ARITHMETIC OPERATORS

$+x$	Unary plus	
$-x$	Unary minus	
$x + y$	Addition	
$x - y$	Subtraction	
$x * y$	Multiplication	
x / y	Division	
$x \% y$	Modulo	

Incorrectly using the arithmetic operators can lead to undefined behavior

LOGICAL AND RELATIONAL OPERATORS

!x	Logical Not	
x && y	Logical AND	
x y	Logical OR	
x == y	Equal to	
x != y	Not equal to	
x < y	Less than	
x > y	Greater than	
x <= y	Less than or equal to	
x >= y	Greater than or equal to	

ARITHMETIC EXPRESSION EXAMPLES

Mathematical Formula

C++ Expression

$$b^2 - 4ac$$

b*b - 4*a*c

$$x(y + z)$$

x*(y + z)

$$\frac{1}{x^2 + x + 3}$$

1/(x*x + x + 3)

$$\frac{a + b}{c - d}$$

(a + b)/(c - d)

ASSIGNMENT OPERATORS

The assignment operators return a reference to the left- hand side

```
int a, b, c = 10; // a = ?, b = ?, c = 10
```

```
a = b = c = 10; //a, b, and c all have value 10
```

```
a = b + 100; //a has value 110, b has value 10
```

```
a = 20;
```

```
b = 20;
```

```
c = 20;
```

SHORTCUT NOTATION EXAMPLES

+=

count = count + 2;

count += 2;

***=**

bonus = bonus * 2;

bonus *= 2;

/=

time = time / factor;

time /= factor;

%=

rem = rem % (cnt1+ cnt2);

rem %= (cnt1 + cnt2);

INCREMENT AND DECREMENT OPERATORS

++count // Prefix increase by one

count++ // Postfix increase by one

count = count + 1;

count += 1;

--count // Prefix decrease by one

count-- // Postfix decrease by one

count = count - 1;

count -= 1;

INCREMENT AND DECREMENT OPERATORS

How many times will this loop iterate?

```
const int max1 = 5;

int main()
{
    int count = 0;

    cout << "\ncount is " << count << " before the loop";
    while (count++ < max1) //test then add
        //++count //add then test
    {
        cout << "\ncount is " << count;
    }

    return 0;
}
```

TERNARY CONDITIONAL OPERATORS

Ternary operator: $x ? y : z$

- Semantics
- x is evaluated and converted to `bool`
- If the result was `true`, y is evaluated
- Otherwise z is evaluated

The type and value category of the resulting expression depend on the operands

```
int n = (1 > 2) ? 21 : 42; // 1 > 2 is false (n value will be 42)
```

```
int m = 42;  
((n == m) ? m : n) = 21; // true (m will be assigned 21)
```

- **Branching (example: if/ else)**

//assume variables have values and this is inside a function body

```
if (hours > 40)
{
    //overtime calculation
    overTimeHours = (hours - 40);
    grossPay = (rate * 40) + (1.5 * rate) * (hours - 40);
}
else
    //no overtime
    grossPay = rate * 40;
```

- **Looping (example: while)**

```
//assume this is inside a function body
```

```
int countdown;  
cout << "How many greetings do you want? ";  
cin >> countdown; //initialization
```

```
//while loop example  
while (countdown > 0) /////boolean expression (test)  
{  
    cout << "Hello! ";  
    countdown--; //update  
}
```


- **Looping (example: do/while)**

```
//assume this is inside a function body
```

```
char ans;
```

```
//do/ while loop example
```

```
do {  
    cout << "Welcome!\n";  
    cout << "Do you want another greeting?\n"  
        << "Press y for yes, n for no, \n"  
        << "and then press return: ";  
    cin >> ans; //update  
  
} while (ans == 'y' || ans == 'Y'); //test
```

PROGRAMMING STYLE

- Use meaningful variable names (identifiers)

Indenting

- A program should be laid out so that elements that are naturally considered a group are made to look like a group
- Use a style that shows the structure of the program, is consistent, and easy to read

Comments

- Include explanatory notes at key places in the program
- Whenever something is important and not obvious, it merits a comment
- It will take experience to get the feel for when it is best to include comments
- All programs should include a descriptive header comment

Undefined behavior

In some situations, the behavior of a program is not well-defined

Undefined behavior falls outside the specification of the C++ standard

- The compiler is allowed to do anything when it encounters undefined behavior
- Fall back to some sensible default behavior
- Do nothing
- Do anything else you can think of

A C++ program must never contain undefined behavior!