

DATA TYPES AND VARIABLES - INTRODUCTION

- ❑ Values (objects) can be grouped into types according to their properties, e.g.
 - Integers
 - Real numbers
 - Characters

- ❑ A **type** denotes how a value is represented and what operations can be performed on it, e.g.
 - Integers have type **int**.
 - Operations including +, -, *, / may be performed on values of type **int**.

BASIC DATA TYPES

- ❑ Basic (built-in) data types include:
 - **char** 'a', 'c', '#', '\')
 - **int** 45, 3, 897878, -36
 - **float** and **double** 63.98, 0.56, -98.7, 99.0

- ❑ Basic data types have different storage sizes which is machine dependent, e.g.
 - **char** usually stored in 8 bits
 - **int** usually stored in machine word (e.g. 32 bits)
 - **void** has no storage

LITTLE BIT OF MATHEMATIC

□ Data are represented in computer using two numbers:

- 0
- 1

Basic information unit in computer is **bit**. It can has value either 0 or 1.

Group of 8th **bits** is called **Byte**.

Example:

Number 12 can be stored inside computer as sequence

00001100

QUALIFIERS

□ Qualifiers modify integers and (double)

- **short**, **long** - applied to integers
- can declare objects as **short int** or just **short**
- These are machine dependent but guarantees:

long >= int >= short >= char
(e.g. 32 bits) (e.g. 32 bits) (e.g. 16 bits) (e.g. 8 bits)

□ Also:

- **long double** for extended floating point precision

long double >= double >= float

- **signed**, **unsigned** are applied to chars and ints

unsigned goes from **zero** to **positive number**

- **const** objects cannot be changed (read only)

QUALIFIERS & TYPES (EXAMPLE)

□ Basic types:

- `int`
- `char`
- `float`
- `void`

□ Quantified types:

- `unsigned int`
- `long double`
- `short unsigned int`
- `constant char`
- `constant void`

CONSTANTS (LITERAL VALUES)

□ Integer constant

- decimal 24, -679, 78665
- octal 014, 052, 04
- hexadecimal 0x0c, 0xff, 0x2a

□ Character constants

- code that represents the characters in the machine's character set
 - `'a' 'y' '?'` printable
 - `'\n' '\t' '\a'` non printable (\ = escape char)
 - `'\0nn'` octal, where n is 0-7
 - `'\xhh'` hexadecimal, h is 0-9, a-f
 - `"hello world"` character string constant
 - `'\0'` null character (zero value)
- Note that `'x'` is not the same as `"x"`! `'x'` is character while `"x"` is character string with only one symbol.

VARIABLES

- ❑ A variable is a **named container** which can hold **objects** representing **values** (skipping “object” point of view, check out analogy with mathematical variables)



The box represents a container named X which is holding an object representing the value one

- ❑ To clarify
 - Values are **abstract** entities, e.g. natural numbers
 - Objects are C++ **representations** of values
 - Variables **hold** objects
- ❑ Note that as a variable is a container, its value can be changed, hence the name.

VARIABLE NAMES AND DEFINITIONS

- ❑ Variable names are case sensitive and are alphanumeric
 - x x34 weight height are **valid**
 - 34x % 34- number of cars are **invalid**
 - number_of_cars NumberOfCars are **valid**
 - Note that C++ variables are case sensitive therefore numberOfCars and NumberOfCars represent different variables
 - A legal identifier is any sequence of letters and digits, there are however a few things that are not allowed
 - An identifier should not
 - begin with a number
 - begin with `_`
 - be a C++ keyword
 - contain spaces

VARIABLE NAMES AND DEFINITIONS

- C++ variables are declared as:

<type> **<var_name>**

- **<type>** is any of the fundamental data types described previously and **<var_name>** is the name that will be associated with this new object

- Example

```
// declares upper and lower as integers
int    upper, lower;
// gender holds a single character
char   gender;
```

- A recently declared object that has not been initialized is a dangerous thing, there is no way of knowing the value of an uninitialized variable!

VARIABLE NAMES AND DEFINITIONS

- The variable may also be initialized on declaration

```
int    age1 = 25, age2 = 30;
float  height = 180.34;
int    age_difference = age2 - age1;
```

- Note that a variable is usable from the point of declaration

- So it can be used even before initialization

- Example:

```
int i,j,k;
k = i+j;
```

- Result is non deterministic

READABILITY

- ❑ It is extremely helpful to both yourself and other programmers to use logical variable names. `int x` may seem like a solid variable name but it is extremely ambiguous to anyone who uses or maintains your code in the future.

- Acceptable variable names

`radius` `NumberOfObjects` `Student_Address`

- Unacceptable (but valid) variable names

`r` `NOO` `S_A`

Use common sense when choosing variable name. Even descriptive names may become "too" annoying to read.

`NumberOfObjectsPresentInListWhenInstanceOfListIsCreated`

`NumberOfObjectsPresentInListWhenInstanceOfListIsDeleted`

STATE AND VARIABLES

- ❑ The state of a C++ program is represented by
 - the state of the virtual machine defining how C++ programs are executed
 - a collection of variables manipulated by the program

Variables are named and used within C++ program text to store and manipulate values

VARIABLES AND TYPES

- All variables are typed and can **only** hold values of that type
 - Think of the variable container being of a certain shape and size that can only hold objects that fit
 - Only operations supported by the type of the variable can be performed on the contents of the variable
 - C++ supports strong typing which enforces these rules

CONSTANTS

- The keyword `const` can be added to a variable declaration to create constants - a variable whose value cannot be changed after initialization

```
const int radius = 12
```

```
const float pi = 3.145
```

- constant must be initialized when declared
- The value of constant **cannot** be changed by assignment
- The compiler will not compile a program with for example,

```
radius = 13
```

ASSIGNMENT OPERATOR

- The contents of a variable can be changed using the assignment operator (=) (It is assignment operator, equality operator is ==)

```
int age;           // declare the variable age
age = 35;          // store the representation of 35 in the
                  // variable named age
age = age + 1;     // add 1 to the contents of age and
                  // store the result back in age
                  // Note that this is NOT a valid
                  // mathematical expression
```

ASSIGNMENT OPERATOR

- Given

```
- int first;
- int second;
- int third;
```

We can write

```
third = second = first = 42;
```

This is equivalent to

```
first = 42;
second = 42;
third = 42;
```

That is

```
third = (second = (first = 42));
```

- Important: = is not a test for equality.

ENUMERATED TYPES

- ❑ An enumerated type is a type whose values are taken from a list of possible constants. The construction for declaring an enumerated type is as follows,
 - Usage: `enum unique_name { constant-list }`
 - The `unique_name` becomes a distinct type after declaration and any variable of that type can only take a value that is contained in the list of constants
- ❑ They make code easier to read and understand, e.g.
 - A program that frequently uses the colors as attributes,

```
enum color {Blue, Red, White, Black};
color car_color;

The field car_color can only take values from Blue, Red, White or Black
```
- ❑ Variables of enumerated type can, once initialized, have their values updated. However you should note that enumerated types cannot be iterated over

ENUMERATED TYPES (EXAMPLE)

```
enum color {Blue, Red, White, Black};
color car_color1, car_color2;
car_color1 = Blue;
car_color2 = White;
car_color1 = Black;

car_color1 = car_color1 + car_color2; invalid!
```

REFERENCES

- A reference is a type which gives an **alternative** name for an object, such as a variable

```
int i;           // normal variable
int &r = i;      // reference variable
```

- r is now an alternative name (an alias) for the variable i
- Both i and r refer to the same container

So if we have `r = 2;`
this means that i is also equal to 2

- Any type name followed by an '&' denotes a reference type

```
int i;
double x, y;
char c;
```

REFERENCES

```
int& r = i;
double &a = x, &b = y;
char &c1 = c;
```

- **Note** that a space can appear before and after the '&'
- **Note** that when a reference is defined, it **MUST** be initialized to be a reference to something of the correct type

```
int x;
int &r = 1;           // invalid
int &s;               // not initialized
```

- s is useless as it does not refer to anything
- **Note** that there is no other way to refer to an object than by initialization

REFERENCES

- The reference is simply another name for an existing object
- The reference can be used in normal expressions, in the same way as a variable name

```
int x;  
int &r = x;  
r = 2 + 4;  
r = r * 2;
```

- **Note** that the operators (=, *, +, etc) are applied to the object that r refers to, i.e. the integer x
- **Note** that there are no operators that operate on the reference itself
- **Note** that a reference can only refer to a single variable