Data Types, Variables, and Operators in C

Md. Aminul Islam Shazid

Outline

1. Data Types and Variables
2. Operators
3. Input, Output (IO)
4. Exercise

Data Types and Variables

Basic Data Types in C

- **int**: whole numbers (typically 4 bytes)
  Usage: integer data, counters, loop indices
- **float**: single-precision decimals (~6 digits)
  Usage: decimal data
- **double**: double-precision decimals (~15 digits)
  Usage: precise calculations, finance
- **char**: single character (1 byte, ASCII)
  Usage: characters, text handling
- **void**: represents no value
  Usage: function return type, pointers
- **short, long, unsigned**: integer variants
  Usage: memory optimization, large values

Variable Sizes and Precision

- **Sizes vary by system/compiler**, but common values:
  - `char`: 1 byte
  - `short`: 2 bytes
  - `int`: 4 bytes
  - `long`: 4 or 8 bytes
  - `float`: 4 bytes (about 6 decimal digits)
  - `double`: 8 bytes (about 15 decimal digits)
- Use `sizeof()` operator to check actual size
- Precision: `float` (single) vs. `double` (double precision)

Variable Definition and Declaration

- Syntax: `data_type variable_name;`
- Initialization: `int x = 10;`
- Can also do: `int x; x = 10;`
- Scope:
  - Local: inside a function
  - Global: outside all functions
- Constants:
  - `const int MAX = 100;`
  - `#define PI 3.14`

Type Casting in C

- **Type casting** converts a variable from one data type to another
- **Implicit casting (type promotion):**
  - Done automatically by the compiler
  - Example: `int x = 5; double y = x;` *(x promoted to double)*
- **Explicit casting:**
  - Done by the programmer using cast operator
  - Syntax: `(type) expression`
  - Example: `double a = 5.7; int b = (int)a;` *(b = 5)*
- Use casting carefully: may cause data loss (e.g., truncation)

Variable Naming Rules in C

- Must begin with a letter or underscore (_)
- Can contain letters, digits, and underscores
- Case-sensitive: `value` and `Value` are different
- Cannot be a reserved keyword (`int`, `return`, etc.)
- Should be meaningful for readability (e.g., `total`, not `x1`)

Operators

Operators in C

- **Arithmetic:** `+`, `-`, `*`, `/`, `%`
  Perform basic mathematical operations
- **Relational:** `<`, `<=`, `>`, `>=`, `==`, `!=`
  Compare two values, result is either true (1) or false (0)
- **Logical:** `&&`, `||`, `!`
  Combine conditions: `&&` (AND), `||` (OR), `!` (NOT)
- **Assignment:** `=`, `+=`, `-=`, `*=`, `/=`
  Store values in variables or update them with shorthand forms

Prefix vs Postfix Operators

- **Increment / Decrement operators:** `++`, `--`
- **Prefix form** (`++x`, `--x`)
  - Variable is updated first, then used in the expression
  - Example:
    - `int x = 5;`
    - `int y = ++x;` *(x=6, y=6)*
- **Postfix form** (`x++`, `x--`)
  - Variable is used first, then updated
  - Example:
    - `int x = 5;`
    - `int y = x++;` *(x=6, y=5)*
- Rule of thumb: prefix: “increment before use”, postfix: “increment after use”.

Truth Tables for Logical Operators

AND (&&)

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

OR (||)

A	B	A || B
0	0	0
0	1	1
1	0	1
1	1	1

NOT (!)

A	!A
0	1
1	0

Order of Evaluation and Precedence

Operators in C follow a precedence hierarchy.

Examples (highest to lowest):

- `()`: Parentheses
- `*`, `/`, `%`: Multiplication, Division, Modulo
- `+`, `-`: Addition, Subtraction
- `<`, `>`, `<=`, `>=`: Relational
- `==`, `!=`: Equality
- `&&`: Logical AND
- `||`: Logical OR
- `=`: Assignment (lowest)

Use parentheses `()` to make evaluation explicit.

Example: `int x = 2 + 3 * 4;` → result is 14, not 20.

Input, Output (IO)

Formatted Output: printf()

- Used to display output to the screen
- General form: `printf("format string", values);`
- Format specifiers:
  - `%d` → integer
  - `%f` → float/double
  - `%c` → char
  - `%s` → string
- Example: `printf("Sum = %d", x);`

Formatted Input: scanf()

- Used to take input from the user
- General form: `scanf("format string", &variables);`
- Format specifiers are the same as for `printf`
- Example: `scanf("%d", &x);`

Why use the ampersand sign (&) in scanf()?

- `scanf()` needs the **address of a variable** to store the input value
- The operator `&` (“address-of”) provides that memory location
- Example:
  - `int x;`
  - `scanf("%d", &x);`
  - Without `&`, the program will not know where to put the value
- **Exception:** For strings (`%s`), the variable itself is already an address, so no `&` is needed

Exercise

Exercise

- Write a C program that demonstrates the basic arithmetic operations
- Write a C program that divides an 5 (integer) by 2 (integer), 5.0 (float) by 2 (integer), and 5 (integer) by 2.0 (float)
- Write a C program that checks whether a user-given number is odd or even, you can use the modulo (%) operator
- Guess is the outputs:

```
int x = 5; printf("%d", x++);
int y = 5; printf("%d", ++y);
```

Questions?