Data Types, Variables, and Operators in C

Md. Aminul Islam Shazid

Outline

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
  Usage: decimal data
- **double**: double-precision decimals
  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)

Signed vs Unsigned Integers

- **Signed integers** can store both positive and negative values
- **Unsigned integers** can store only non-negative values
- **Range difference:** For the same number of bytes, unsigned types can store roughly twice the positive range. *Example (32-bit):*
  `int: -2,147,483,648 to 2,147,483,647`
  `unsigned int: 0 to 4,294,967,295`
- **When to use unsigned:** When negative values are impossible or meaningless, e.g., array indices, sizes, counters, or bitwise operations

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
- **Increment or decrement:** `++`, `--`**:** `++` increseases the value of a variable by 1, `--` decreases the value of a variable by 1

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;` *(Now: x=6, y=6)*
- **Postfix form** (`x++`, `x--`)
  - Variable is used first, then updated
  - Example:
    - `int x = 5;`
    - `int y = x++;` *(Now: 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
- Can display literal text as well as values of variables
- General form: `printf("text to display", values);`
- To display values of variables, format specifiers are needed:
  - `%d` → integer
  - `%f` → float/double
  - `%c` → char
  - `%s` → string
- Example: `printf("Sum = %d", x);`
- In the above, the value of `x` is printed in place of `%d`
- A single `printf()` call can have multiple format specifiers to print the values of multiple variables: `printf("%d %d", x, y);`

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);`
- In the above, the computer expects an integer value to be given as input by the user, then stores it in the variable `x`
- Inputting multiple values: `scanf("%d %d", &x, &y);`

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 already holds an address, so no `&` is needed

Examples

Sum of Two Integers

```c
#include <stdio.h>

int main(){
    int a, b, c;
    a = 1;
    b = 2;
    c = a + b;
    printf("%d", c);

    return 0;
}
```

Sum of Two User-Given Integers

```c
#include <stdio.h>

int main(){
    int a, b;
    printf("Enter first integer:\n");  // \n: newline
    scanf("%d", &a);

    printf("Enter second integer:\n");
    scanf("%d", &b);

    int c = a + b;
    printf("The sum is: %d", c);
    return 0;
}
```

Implicit Typecasting

```c
#include <stdio.h>
int main() {
    int i = 10, j;
    float d, e = 5.25;

    // implicit typecasting (int -> float):
    d = i;
    printf("Value of i (int): %d\n", i);
    printf("Value of d (double): %f\n", d);

    // implicit typecasting (float -> int):
    j = e;
    printf("Value of j (int): %d", j);
    return 0;
}
```

Explicit Typecasting

```c
#include <stdio.h>

int main() {
    double d = 9.78;
    int i;

    // explicit typecasting (double -> int)
    i = (int)d;

    printf("Value of d (double): %lf\n", d);
    printf("Value of i (int after explicit cast):
    ↪  %d\n", i);

    return 0;
}
```

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)
- Guess the outputs:

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

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

Questions?