C Program Structure and Keywords

Md. Aminul Islam Shazid

Outline

1. Hello World
2. Program Structure
3. Keywords in C

Hello World

Hello World in C

```c
#include <stdio.h>

int main(){
    printf("Hellow, world!");
    return 0;
}
```

Greetings in C

```c
#include <stdio.h>

int main(){
    char name[20];
    printf("Enter your name: ");
    scanf("%s", &name);
    printf("Good day to you, %s!", name);
}
```

Program Structure

Preprocessor Macros

- `#include`: include header files
- `#define`: define constants/macros
- Conditional macros: `#if`, `#ifdef`, `#ifndef`, `#else`, `#elif`, `#endif`: compile conditionally
- `#undef`: remove macro definitions

The `main` Function

- Entry point of C programs
- Return type (`int`) indicates exit status
- Function name: `main`
- Parentheses for parameters (empty for now)
- Curly braces {} define the body

Defining Variables

- Syntax: `data_type variable_name;`
- Must end with semicolon `;`
- Variables must be declared before use
- Can assign values at declaration

Semicolons and Statements

- Semicolon ends a statement. Two statements in one line:
  - `int x = 10; y = x + 5;`
  - Discouraged in practice
- Multiple statements form the body of functions
- Common source of beginner errors

Calling Functions

- Syntax: `functionName(arguments);`
- Parentheses hold arguments (can be empty)
- Must match function definition/prototype

Return Values

- Functions can return a value to the caller
- Syntax: `return expression;`
- In `main()`, `return 0;` indicates successful execution
- Non-zero return values often indicate an error

Brackets in C

- Parentheses `()`: grouping expressions and function calls
- Curly braces `{}`: define a block of code
- Square brackets `[]`: array indexing

Comments

- Single-line: `// comment`
- Multi-line (often discouraged):

```
/* this is
a multiline comment */
```

Escape Sequences

- `\n`: newline
- `\t`: tab
- `\\`: backslash
- `\"`: double quote
- `\'`: single quote

Whitespace and Indentation

- Whitespace is ignored (except in strings)
- Indentation improves readability
- Use a single tab or four spaces for one level of indentation
- Example:

```
if(condition){
    statement;
}
```

Coding Conventions

- Meaningful variable names
- Consistent indentation
- Opening brace { on the same line as keyword
- Use comments for clarity

Keywords in C

Data Types and Values

- `int`: integer type
- `float`: single precision floating-point
- `double`: double precision floating-point
- `char`: single character
- `void`: no return value / no data
- `signed`, `unsigned`: signed/unsigned integers
- `short`, `long`: specify integer size

Control Flow

- `if`, `else`: conditional branching
- `switch`, `case`, `default`: multi-way branching
- `for`, `while`, `do`: loops
- `break`: exit loop or switch
- `continue`: skip current iteration
- `goto`: jump to label (use sparingly)
- `return`: exit function, optionally returning value

Structuring

- `struct`: group related variables
- `union`: store different types in same memory
- `enum`: named integer constants
- `typedef`: define a type alias

Pointer and Address-of Operators

- *datatype *varname*: declares a pointer
- **varname*: dereferences a pointer to access the value
- *&varname*: gives the memory address of a variable
- Example:

```c
#include <stdio.h>

int main(){
    int x = 10;
    int *p = &x;    // p points to x

    printf("%x", p);    // prints the hex address of x
    printf("\n%d", *p);    // prints 10
}
```

Storage Classes

- `auto`: default local variable storage
- `register`: hint to store variable in CPU register
- `static`: preserve value between function calls
- `extern`: variable defined elsewhere

Memory and Miscellaneous

- `const`: read-only variable
- `volatile`: variable may change unexpectedly
- `restrict`: pointer optimization hint
- `inline`: suggest inline function expansion
- `_Atomic`: atomic variable access
- `_Thread_local`: thread-local storage
- `_Noreturn`: function does not return
- `sizeof`: size of object or type
- `_Alignas`, `_Alignof`: memory alignment
- `_Generic`: type-generic selection (C11)

Preprocessor Keywords

- `#define`: define macro or constant
- `#include`: include header file
- `#if`: conditional compilation
- `#ifdef`: compile if macro defined
- `#ifndef`: compile if macro not defined
- `#else`, `#elif`: alternative conditions
- `#endif`: end conditional
- `#undef`: undefine macro
- `#line`: set line number for compiler messages
- `#error`: generate compilation error
- `#pragma`: compiler-specific instruction

Questions?