C Programming Basics Notes

1. Features of C Language

- Simple and structured language.
- Portable: C programs can run on different machines with little or no modification.
- Fast and efficient due to low-level memory access.
- Extensible: New features and functions can be added easily.
- Provides modularity using functions.

2. Structure of a C Program

A typical C program follows this structure:

```
#include <stdio.h>   // Preprocessor directive

int main() {         // Main function
    // Variable declarations
    // Statements
    return 0;
}
```

Explanation of structure

1. **Documentation Section** – Comments about program.
2. **Preprocessor Section** – `#include` directives.
3. **Global Declaration Section** – Variables/functions declared globally.
4. **main() Function Section** – Starting point of execution.
5. **Subprogram Section** – User-defined functions.

3. Compilation Process

Steps involved in converting source code into an executable:

1. Preprocessing

- Handles directives starting with `#` (e.g., `#include`, `#define`).
- Includes header files, expands macros, and removes comments.
- Outputs an intermediate `.i` file.

2. Compilation

- Converts the preprocessed code (`.i`) into assembly code (`.s`).
- Checks for syntax errors and semantic correctness.
- Produces an assembly file containing low-level code.

3. Assembling

- Translates assembly code (`.s`) into machine code, resulting in an object file (`.o`).
- The object file is in binary format understood by the processor.
- May use tools like `as` (assembler) to perform this step.

4. Linking

- Combines one or more object files (`.o`) into a single executable.
- Incorporates code from libraries (standard or user-defined).
- Resolves references to external functions and variables.
- Produces the final executable (e.g., `a.out`).

5. Loading & Execution

- The operating system loads the executable into memory.
- Allocates memory for code, data, heap, and stack segments.
- Begins execution at the program's entry point (`main()`).

5. Variables in C

- A variable is a name given to a memory location.
- Must be declared before use.

Syntax:

```
data_type     variable_name;
```

- `int            age;`
- `float        salary;`
- `char          grade;`

6. Activity 1 – Hello World Program

```
#include <stdio.h>

int main() {

    printf("Hello, World!\n");

    return 0;

}
```

7. Activity 2 – Program with Different Data Types

```
#include <stdio.h>


int main() {
    int age = 20;
    float height = 5.9;
    double pi = 3.1415926535;
    char grade = 'A';


    printf("Age: %d\n", age);
    printf("Height: %.2f\n", height);
    printf("Value of Pi: %.10lf\n", pi);
    printf("Grade: %c\n", grade);


    return 0;
}
```

Fundamental Data Types in C

Data Type	Storage Size (Typical 32-bit System)	Storage Size (Typical 64-bit System)	Range (Approx.)
`char`	1 byte (8 bits)	1 byte (8 bits)	-128 to 127 (signed) / 0 to 255 (unsigned)
`short int` / `short`	2 bytes (16 bits)	2 bytes (16 bits)	-32,768 to 32,767
`int`	2 bytes (16 bits)	4 bytes (32 bits)	-32,768 to 32,767 (16-bit) / -2,147,483,648 to 2,147,483,647 (32-bit)
`long int` / `long`	4 bytes (32 bits)	8 bytes (64 bits)	-2,147,483,648 to 2,147,483,647 (32-bit) / very large on 64-bit
`float`	4 bytes	4 bytes	~1.2E-38 to 3.4E+38 (6 decimal places)
`double`	8 bytes	8 bytes	~2.3E-308 to 1.7E+308 (15 decimal places)
`long double`	10–12 bytes (compiler dependent)	16 bytes (on many modern systems)	~3.4E-4932 to 1.1E+4932 (19–20 decimal places)

Keywords in C and Their Purpose

C has **32 keywords** that are reserved by the language. They cannot be used as identifiers (variable names, function names, etc.).

Keyword	Purpose
`auto`	Declares automatic (local) variables (default in C).
`break`	Exits from a loop or switch statement.
`case`	Defines a branch in a `switch` statement.
`char`	Declares a character variable (1 byte).
`const`	Declares constants; value cannot be modified after initialization.
`continue`	Skips the rest of the loop iteration and goes to the next iteration.
`default`	Defines the default branch in a `switch` statement.
`do`	Used in `do...while` loops, executes block at least once.
`double`	Declares a double-precision floating-point variable.
`else`	Used with `if` to specify alternative execution path.
`enum`	Defines a set of named integer constants (enumeration).
`extern`	Declares a global variable or function defined in another file.
`float`	Declares a floating-point variable (single precision).
`for`	A looping construct for iteration.
`goto`	Transfers control to a labeled statement (not recommended).

`if`	Conditional branching statement.
`int`	Declares an integer variable.
`long`	Declares a long integer variable.
`register`	Suggests storing variable in CPU register for faster access.
`return`	Exits from a function and optionally returns a value.
`short`	Declares a short integer variable.
`signed`	Declares signed data types (default for int/char).
`sizeof`	Returns the size (in bytes) of a variable or data type.
`static`	Preserves variable value between function calls / internal linkage.
`struct`	Defines a structure (collection of variables).
`switch`	Multi-way branch statement based on a variable's value.
`typedef`	Defines a new name (alias) for an existing data type.
`union`	Defines a union (shared memory for different data types).
`unsigned`	Declares unsigned data types (only positive values).
`void`	Declares functions that return no value, or empty pointers.
`volatile`	Tells compiler variable may change unexpectedly (e.g., hardware).
`while`	Looping construct that repeats while a condition is true.

Additional Notes: Program vs Process

1. What is a Program?

- A **program** is a set of instructions written to perform a task, stored in secondary memory.
- Passive entity: does not execute itself.
- When executed, the operating system creates a process.
- One program can have multiple processes.

Features of a Program

- Multiple programs can run by a single user.
- Stored in secondary memory until executed.
- Inert file containing instructions (no control block).
- Example: A browser program may spawn multiple processes for tabs.

---

2. What is a Process?

- A **process** is an instance of a program in execution.
- Active entity, created when program is loaded into main memory.
- Uses system resources like CPU, memory, and I/O.
- Ends when execution is complete.

Features of a Process

- Each process has a Process Control Block (PCB).
- Temporary: exists only during execution.
- Needs system resources (CPU time, memory, I/O).
- Active entity performing operations.

---

3. Difference Between Program and Process

Program	Process
Set of instructions written to complete a task.	Instance of a program in execution.
Passive entity (resides in secondary memory).	Active entity (loaded into main memory).
Static – does not change itself.	Dynamic – changes during execution.
Exists until deleted.	Exists temporarily until task finishes.
Needs only storage space.	Needs CPU, memory, I/O resources.
No control block.	Has a Process Control Block (PCB).
Components: Code + Data.	Code + Data + Execution info.
Does not execute itself.	Executes as a sequence of instructions.

Table of all numeric types in C with their printf specifiers

Data Type	Size (Typical)	Format Specifier	Description
`int`	4 bytes	`%d`	Signed integer
`unsigned int`	4 bytes	`%u`	Unsigned integer
`short`	2 bytes	`%hd`	Signed short integer

`unsigned short`	2 bytes	`%hu`	Unsigned short integer
`long`	8 bytes (on 64-bit)	`%ld`	Signed long integeression
`unsigned long`	8 bytes	`%lu`	Unsigned long integer
`long long`	8 bytes	`%lld`	Signed long long integer
`unsigned long long`	8 bytes	`%llu`	Unsigned long long integer
`float`	4 bytes	`%f`	Single-precision floating point
`double`	8 bytes	`%lf`	Double-precision floating point
`long double`	16 bytes (varies)	`%Lf`	Extended-precision floating point