Basics Of C Programming

Data Types

In order to use a variable, the data type of the variable must be specified along with it's name. To pass the variable in a printf statement, a format specifier must be used. There are 4 basic data types along with it's format specifier

1. char - %c or %s
2. int - %d
3. float - %f
4. double - %lf

NOTE: there is no string data type in C

Arithmetic Operators

The arithmetic operators in C, listed by their precedence from highest to lowest, are:

`* → / → % → + → - → ++ → --`

Logical Operators

These operators are used to specify the logic between variables and operators, with multiple conditions

- `&&` - AND: Returns true only when both values are true
- `||` - OR: Returns true if any one value is true
- `!` - NOT: Returns true if false and false if true

Bitwise Operators

- '&' - AND: Returns true only when both bits are true
- '|' - OR: Returns true if any one bit is true
- '^' - XOR: Returns 1 if both bits are different
- '~' - NOT: Returns true if false and false if true
- '<<' - LEFT SHIFT: left shifts bits of 1st operand and 2nd operand decides number of places to shift
- '>>' - RIGHT SHIFT: right shifts bits of 1st operand and 2nd operand decides number of places to shift

NOTE:

- left shift formula: a x (2^n)
- right shift formula: a / (2^n) this is only for checking and calculating purpose

---

Printing Statements

A printf statement is used to return a formatted output to the screen

```
#include <stdio.h>

int main() {
  printf("This is a C Programming");
  return 0;
}
```

To pass a variable through the printf statement, we must use the specified variable's format specifier and the variable name.

```
#include <stdio.h>

int main() {
  int x = 81;
  printf( "%d", x );
  return 0;
}
```

There are special characters, each serving a different purpose. Some basic special characters are \n, \t and more . . .

Conditional Statament

Conditional statements in C are used to control the flow of execution based on specified conditions. The conditional statements in C are if statements and switch-case statements

If Statements

An if statement is a conditional statement used in programming to execute a block of code only if a specified condition is true.

Syntax:

```
if ( condition ) {
  // code block
} elseif ( condition ) {
  // code block
} else {
  // code block
}
```

We can also use elseif statements to add more conditions. Only when the first if condition fails, the elseif statement run.

For Ex:

```
#include <stdio.h>

int main() {
  // Declaring the variable with its value
  int x = 10;
  // to check if the number is even or odd
  if ( x % 2 == 0 ) {
    printf( "%d is even!\n ", x );
  } else {
    printf( "%d is odd!\n ", x );
  }
  return 0;
}
```

Output: 10 is even!

Switch-Case Statements

Switch-case statements are also a type of conditional statement in C. They are used to execute different blocks of code based on the value of a single variable or expression. This statement works best when checking a variable against multiple constant values.

Syntax:

```
switch (expression) {
    case constant1:
        // code block
        break;
    case constant2:
        // code block
        break;
    ...
    default:
        // default code block
}
```

Without the use of break, execution will continue into the next case, even if a match is found. This is called fall-through and is usually not efficient or desired. Therefore, we use break to exit the switch block once a matching case has been executed.

For Ex:

```c
#include <stdio.h>

int main() {
    int day = 3;

    switch (day) {
        case 1:
            printf("Monday\n");
            break;
        case 2:
            printf("Tuesday\n");
            break;
        case 3:
            printf("Wednesday\n");
            break;
        case 4:
            printf("Thursday\n");
            break;
        case 5:
            printf("Friday\n");
            break;
        case 6:
            printf("Saturday\n");
            break;
        case 7:
            printf("Sunday\n");
            break;
        default:
            printf("Invalid day\n");
    }
    return 0;
}
```

`Output:` Wednesday

Loop Statements

For Loop

A for loop is used when a block of statements needs to be iterated a known and fixed number of times.

`Syntax:`

```c
for ( initialization; condition; update ){
  // code block
}
```

For Ex:

```
// This program prints the first 10 natural numbers using for loop
#include <stdio.h>

int main() {
  for ( int i = 1; i <= 10; i++ ){
    printf("%d\t", i);
  }
  return 0;
}
```

Output: 1 2 3 4 5 6 7 8 9 10

While Loop

A while loop is used when a block of statements needs to be iterated an unknown number of times, depending on a condition.

Syntax:

```
while ( condition ) {
  // code block
}
```

For Ex:

```
// This program prints the first 10 natural numbers using while loop
#include <stdio.h>

int main() {
  int i = 1
  while ( i < 11 ) {
    printf("%d\t", i);
    i++;
  }
  return 0;
}
```

Output: 1 2 3 4 5 6 7 8 9 10

Do-while Loop

A do-while loop is similar to a while loop, but it checks the condition after each iteration of the loop body.

Syntax:

```
do {
    // Code block
} while (condition);
```

Use a do-while loop when you want the code inside the loop to execute at least once, regardless of whether the condition is true or false initially.

`For Ex:`

```
#include <stdio.h>

int main() {
    int i = 10;
    do {
        printf("%d is not less than 5!", i);
    } while (i < 5);  // Condition is false
    return 0;
}
```

`Output:` 10 is not less than 5!

Jump Statements

Jump statements are used to change the flow of control by transferring execution to a different part of the program. In C programming, `break`, `continue`, `return`, and `goto` are classified as jump statements.

Break Statement

The `break` statement is often used inside a loop. It immediately terminates the current loop and transfers control to the first statement after the loop.

`For Ex:`

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3)
            break;  // exits the loop when i is 3
        printf("%d\t", i);
    }
    return 0;
}
```

`Output:` 1 2

Continue Statement

The `continue` statement is also often used inside a loop. It skips the remaining statements in the current iteration and proceeds directly to the next iteration of the loop.

`For Ex:`

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3)
            continue;  // skips printing 3
        printf("%d\t", i);
    }
    return 0;
}
```

`Output:` 1 2 4 5

Return Statement

The `return` statement is used to exit from a function. It immediately terminates the function's execution and optionally sends a value back to the calling function.

`For Ex:`

```
#include <stdio.h>

int square(int n) {
    return n * n;  // exits the function and returns the result
}

int main() {
    int result = square(4);
    printf("Square = %d!\n", result);
    return 0;
}
```

`Output:` Square = 16!

Goto Statement

The `goto` statement is used to transfer control unconditionally to a labeled statement within the same function. It allows the program to jump to a specific part of the code, bypassing normal control flow.

`For Ex:`

```
#include <stdio.h>

int main() {
    int x = 1;
    if (x == 1)
        goto skip;
    printf("This line is skipped.\n");
skip:
    printf("Jumped to the label.\n");
    return 0;
}
```

`Output:` Jumped to the label.

Arrays in C

Arrays store multiple values in a single variable. To create one, specify the data type, array name, and use square brackets `[]`.

`Syntax:`

```
(datatype) variable_name [ size ];
```

`For Ex:`

```
//defining a character array having a size of 10
char name[10];
```

We can also define a 2D by specifying the size in two different square brackets. `For Ex:`

```
//Here we are defining an 2D integer array of dimensions 3x3.
int matrix[3][3]
```

Types of Functions

In C, functions are categorized into two main types:

- Library (built-in) functions – Predefined functions provided by the C standard library, such as printf(), scanf(), sqrt(), etc.

- User-defined functions

User - defined functions

Functions written by the programmer to perform specific tasks and improve code modularity and reusability.

For Ex:

```
#include <stdio.h>

void even_or_odd( int x ) {
  if ( x % 2 == 0 ) {
    printf("%d is even!\n", x );
  } else {
    printf("%d is odd!\n", x );
  }
}

int main ( void ) {
  //Let the input be 14
  int number;
  printf("Enter your Number: ");
  scanf("%d", &number );
  even_or_odd( number );
  return 0;
}
```

Output: 14 is even!

Scope Of Variables

In C, a variable can only be accessed in the region it is created. There are 2 types of scopes:

Local Scope

The variables defined in the local scope can only be used within the local scope.

Global Global

The variables defined in the global scope can be used anywhere within the program.

For Eg:

```
#include <stdio.h>
void variable() {
  int x = 10;
  printf("Local Scope: %d\n", x );
}
```

```
int main() {
  int x = 5;
  variable();
  printf(" Global Scope: %d\n", x );
  return 0;
}
```

Output:

Local Scope: 10

Gobal Scope: 5

Structures in C

A structure in C is a user-defined data type that allows grouping multiple variables of different data types under a single name. It is defined using the keyword `struct`. The individual variables inside a structure are called members or fields, and each member can be of any valid data type.

For Eg:

```
#include <stdio.h>
struct person {
    char name[50];
    int age;
    float height;
};

int main() {
    struct Person p1 = {"Alice", 25, 5.4};
    printf("Name: %s\n", p1.name);
    printf("Age: %d\n", p1.age);
    printf("Height: %.1f\n", p1.height);
    return 0;
}
```

The typedef keyword in C is used to define a new name (alias) for an existing data type. This helps improve code readability and makes complex type declarations easier to manage.

For Eg:

```
#include <stdio.h>
typedef struct person {
    char name[50];
    int age;
    float height;
} Person;
```

```
int main() {
    Person p1 = {"Alice", 25, 5.4};
    printf("Name: %s\n", p1.name);
    printf("Age: %d\n", p1.age);
    printf("Height: %.1f\n", p1.height);
    return 0;
}
```

Union in C

A union in C is a user-defined data type like a struct, but all members share the same memory space, and only one member can hold a value at a time.

For Eg:

```
#include <stdio.h>
union Data {
    int i;
    float f;
    char c;
};

int main() {
    union Data d;
    d.i = 10;
    d.f = 3.14;  // overwrites i
    d.c = 'A';   // overwrites f
    printf("Current value: %c\n", d.c);  // Only c is valid
}
```

Pointers in C

A pointer is a variable that stores the address of another variable.

Parts of a Pointer

- A type (like int*, char*)
- An address (where it points)
- Can also give the value at that address using *

Rules for Pointers:

Use * to create a pointer

- int* p; means p is a pointer to an int

Use & to get the address of a variable

- p = &x; stores address of x in p

Use * to get the value at that address

- *p gives the value of x

For Eg:

```
int x = 10;
int* p = &x;
printf("%d", *p); // prints 10
```