CSE 1101: Structured Programming Language

Weeks 3: Fundamental Concepts in C

- Others C Tokens like String and Punctuations
- Input and output functions in C
- Format Specifiers
- Operators
- Errors in Program

Lecture 7: C Tokens

2.6 C Tokens

In C programming language, a token is the smallest unit of a program that has a meaning. There are six types of tokens in C:

1. **Keywords**: Keywords are reserved words in C that have a special meaning and cannot be used as identifiers (variable names, function names, etc.). Examples of keywords in C are int, char, while, if, etc.
2. **Identifiers**: Identifiers are names given to variables, functions, arrays, etc. An identifier must start with a letter or an underscore and can be followed by any combination of letters, digits, and underscores.
3. **Constants**: Constants are fixed values in a program that cannot be changed during the execution of the program. There are several types of constants in C, including integer constants, floating-point constants, character constants, and string literals.
4. **Strings**: A string is a sequence of characters. In C, a string is represented by a set of characters enclosed in double quotes.
5. **Operators**: Operators are special symbols that perform specific operations on values and variables. Examples of operators in C are +, -, *, /, etc.
6. **Punctuators**: Punctuators are symbols that are used to separate different parts of a program, such as semicolons, brackets, braces, etc.

These tokens are combined to form expressions and statements in a C program. The C compiler uses these tokens to parse the program and generate machine code that can be executed by the computer.

In previous lectures we have discussed variable, identifiers, keyword and constant. In this lecture we will discuss string punctuation and operators and some other related topics.

Engr. Mohammad Mamun Hossain
BSc.(Eng.) and MSc.(Thesis) in CSE, SUST
PhD (Pursuing) in CSE, RUET.
Assistant Professor, Dept. of CSE, BAUST

Contact: mhossain@baust.edu.bd
linkedIn:mamunsust12
Resources: mamunsust12@GitHub
Emergency: 01717690847

2.7 Strings

A string is a sequence of characters encoded with double code that is terminated by a null character ('\0').In C, strings are represented as arrays of characters. The standard way to work with strings is using the char data type and arrays of char. Additionally, format specifiers are used in functions like printf and scanf to format the input and output. Here's an overview of strings and format specifiers in C:

Here's an example of a string in C:

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

In this example, greeting is an array of characters that stores the string "Hello". The null character at the end of the string is important, as it signals the end of the string to various string functions.

Strings can also be defined using string ***literals***, which are sequences of characters surrounded by double quotes. For example:

```
char greeting[] = "Hello";
```

In this example, the size of the array greeting is automatically determined by the compiler, based on the length of the string literal.

C provides a variety of functions for manipulating strings, such as strlen (to determine the length of a string), strcat (to concatenate two strings), strcpy (to copy one string to another), etc. These functions are declared in the string.h header file.

2.7.1 C input and output

In C, input and output operations are performed using standard I/O functions provided by the standard input/output library (<stdio.h>). Here are some commonly used input and output functions:

Input Functions:

scanf - Scan Formatted Input:

Reads input from the standard input (usually the keyboard) based on a specified format.

Example:

```
int num;
printf("Enter an integer: ");
scanf("%d", &num);
```

getchar - Read a Character:

Reads a single character from the standard input.

Example:

```
char ch;
printf("Enter a character: ");
ch = getchar();
```

gets - Read a String (Deprecated):

Reads a line of text (string) from the standard input. Note that gets is considered unsafe and has been deprecated in recent C standards.

Example:

```
char str[50];
printf("Enter a string: ");
gets(str);
```

fgets - Read a Line of Text:

Reads a line of text (string) from the standard input, ensuring a maximum number of characters are read to prevent buffer overflow.

Example:

```
char str[50];
printf("Enter a string: ");
fgets(str, sizeof(str), stdin);
```

Output Functions:

printf - Print Formatted Output:

Prints formatted output to the standard output (usually the console).

Example:

```
int num = 42;
printf("The value is: %d\n", num);
```

putchar - Write a Character:

Writes a single character to the standard output.

Example:

```
char ch = 'A';
putchar(ch);
```

puts - Write a String:

Writes a string to the standard output followed by a newline character.

Example:

```
char str[] = "Hello, World!";
puts(str);
```

fputs - Write a String to a File:

Writes a string to a specified file stream.

Example:

```
FILE *file = fopen("output.txt", "w");
fputs("Hello, File!", file);
fclose(file);
```

fprintf - Write Formatted Output to a File:

Writes formatted output to a specified file stream.

Example:

```
FILE *file = fopen("output.txt", "w");
int num = 42;
fprintf(file, "The value is: %d\n", num);
fclose(file);
```

These are just a few examples of the input and output functions available in C. It's important to handle errors and edge cases appropriately, especially when dealing with user input and file operations. Additionally, be cautious about the use of deprecated functions like gets and consider safer alternatives like fgets.

2.7.2 Format Specifiers

Format specifiers are used in the printf and scanf functions to specify the type of data being printed or read.

%s: String specifier
%c: Character specifier
%d: Integer specifier
%f: Float specifier
%lf: Double specifier
%p: Pointer specifier
%x, %X: Hexadecimal specifier
%u: Unsigned decimal specifier
%o: Octal specifier

Example:

```
int num = 42;
char ch = 'A';
float floatNum = 3.14;

printf("Integer: %d\n", num);
printf("Character: %c\n", ch);
printf("Float: %f\n", floatNum);
```

These format specifiers help define the type and formatting of the data being printed or read. Keep in mind that using the wrong format specifier can lead to undefined behavior or unexpected results.

2.7 Punctuations / separators

Punctuations in C are characters used to separate and delimit various elements of the C language, such as statements, expressions, and blocks of code. The most commonly used punctuation characters in C are:

1. Semicolon (;): Used to separate statements.
2. Comma (,): Used to separate elements in a list, such as arguments in a function call or items in an array declaration.
3. Parentheses (()): Used to group expressions, such as in function calls or arithmetic expressions.
4. Braces ({}): Used to delimit blocks of code, such as in functions or loops.
5. Square brackets ([]): Used to define arrays and to access elements in arrays.
6. Period (.): Used to access members of structures and unions.
7. Arrow (->): Used to access members of structures through pointers.
8. Hash sign (#): Used in preprocessor directives.
9. Asterisk (*): This symbol is used to represent pointers and also used as an operator for multiplication.
10. Tilde (~): It is used as a destructor to free memory.

Examples of punctuation usage in C can be found in the following code:

```
#include <stdio.h>
int main() {
    int i, j;
    int arr[5] = {1, 2, 3, 4, 5};
    // Semicolon
    for (i = 0; i < 5; i++) {
        // Comma
        printf("%d, ", arr[i]);
    }
    printf("\n");

    // Parentheses
    i = (3 + 4) * 5;
    printf("i = %d\n", i);

    // Braces
    for (i = 0; i < 5; i++) {
        // Square brackets
        printf("arr[%d] = %d\n", i, arr[i]);
    }

    // Period and Arrow
    struct point {
        int x;
        int y;
    };
```

```
    struct point p;
    p.x = 10;
    p.y = 20;
    printf("p.x = %d, p.y = %d\n", p.x, p.y);

    struct point *ptr = &p;
    ptr->x = 30;
    ptr->y = 40;
    printf("ptr->x = %d, ptr->y = %d\n", ptr->x, ptr->y);

    //  Hash sign
    #define PI 3.14159
    printf("PI = %f\n", PI);

    return 0;
}
```

2.8 Operators

Operators in C are symbols that perform specific operations on one or more operands (values or variables). C has several types of operators, including:

1. Arithmetic Operators: Used to perform arithmetic operations such as addition, subtraction, multiplication, division, etc.
2. Relational Operators: Used to compare values and determine relationships between them, such as equal to, less than, greater than, etc.
3. Logical Operators: Used to combine relational expressions and perform logical operations such as AND, OR, NOT, etc.
4. Assignment Operators: Used to assign values to variables, such as =, +=, -=, etc.
5. Bitwise Operators: Used to perform bit-level operations on values, such as AND, OR, NOT, etc.
6. Ternary Operators: Used to evaluate a conditional expression, such as (condition) ? x : y.
7. Sizeof Operators: Used to determine the size of a data type or a variable in bytes.
8. Comma Operators: Used to separate two or more expressions in a single statement.

Examples of each type of operator in C can be found in the following code:

```
#include <stdio.h>
int main() {
    int x = 10, y = 20;
    // Arithmetic Operators
    printf("x + y = %d\n", x + y);
    printf("x - y = %d\n", x - y);
    printf("x * y = %d\n", x * y);
```

```
    printf("x / y = %d\n", x / y);

    // Relational Operators
    printf("x == y is %d\n", x == y);
    printf("x < y is %d\n", x < y);
    printf("x > y is %d\n", x > y);

    // Logical Operators
    printf("x < y && x > 0 is %d\n", x < y && x > 0);
    printf("x < y || x > 0 is %d\n", x < y || x > 0);
    printf("!(x < y) is %d\n", !(x < y));

    // Assignment Operators
    x += y;
    printf("x = %d\n", x);
    x -= y;
    printf("x = %d\n", x);

    // Ternary Operator
    x = (x < y) ? x : y;
    printf("x = %d\n", x);

    // Sizeof Operator
    printf("Size of int is %lu bytes\n", sizeof(int));
    printf("Size of x is %lu bytes\n", sizeof(x));

    // Comma Operator
    x = (y = 10, y + 20);
    printf("x = %d\n", x);

    return 0;
}
```

Lecture 09: Error in C Program

2.10 Error in C Program:

When discussing errors in a C program, it's important to distinguish between different types of errors. Errors can broadly be categorized into three main types:

- **Syntax Errors**
- **Runtime Errors**
- **Logical Errors and**
- **Linker Error**

2.10.1 Syntax Errors:

Syntax errors occur when the code violates the rules of the C language.
These errors are detected by the compiler during the compilation process.
Examples include missing semicolons, mismatched parentheses, or using undeclared variables.

```
int main() {
   printf("Hello, World!"  // Syntax error: Missing semicolon
   return 0;
}
```

To fix syntax errors, carefully review the code for typos, missing punctuation, or other violations of the C language rules.

2.10.2 Runtime Errors:

Runtime errors occur when a program is executed and encounters an unexpected condition that cannot be handled.
Examples include division by zero, accessing an array out of bounds, or dereferencing a null pointer.

```
int main() {
   int a = 10, b = 0;
   int result = a / b;  // Runtime error: Division by zero
   return 0;
}
```

To fix runtime errors, you need to identify and correct the logic or conditions causing the unexpected behavior. Additionally, consider implementing error-checking mechanisms to prevent such issues.

2.10.3 Logical Errors:

Logical errors occur when the program does not produce the expected output due to a flaw in its logic.These errors are challenging to detect because the program compiles and runs without any error messages.
Examples include using the wrong formula, incorrect condition checks, or improper algorithm implementation.

```
int main() {
   int x = 5, y = 10;
```

```
    if (x > y) {
       printf("x is greater than y");  // Logical error: Incorrect condition
    }
    return 0;
}
```

To fix logical errors, carefully review the program's logic and ensure that the algorithms and conditions are correct. Debugging tools and techniques, such as print statements, can be valuable for identifying logical errors.

2.10.4 Linker errors: There is another type of error called Linker Error.
Linker errors occur during the linking phase of the compilation process. The linker is responsible for combining multiple object files and libraries into a single executable or shared library. Linker errors typically indicate that the linker is unable to find or resolve symbols (functions or variables) referenced in the program.
Here are some common causes of linker errors:

- Undefined Reference: This error occurs when the linker cannot find the definition (implementation) of a function or variable that is referenced in your code.
- Multiple Definitions: This error occurs when the linker encounters multiple definitions for the same symbol (function or variable).
- ***Missing Library: If your program uses functions from external libraries, you need to link with the appropriate library during the linking phase.***
- Mismatch in function name: If there is a mismatch in the function signature (return type, parameter types) between the function declaration and definition, it can result in linker errors.

Linker errors can be more challenging to debug than compiler errors, as they often involve multiple files and libraries. Reviewing the linker error messages and carefully checking for the causes mentioned above will help you identify and fix the issues.

When communicating errors in a C program, it's essential to provide context, describe the symptoms, and, if possible, include relevant portions of the code. Additionally, consider using debugging tools and techniques to pinpoint the location and cause of the errors.

Exercise: The following program contains some bugs. Debug it.

Erroneous Program	Error Description

`#include<stdio.h>`
 	
`#define PI 3.1416`
 	
`int main(){`
`  int r;`
 	
`  printf("Enter the radius: ");`
`  scanf("%d", radius);`
 	
`  area= 2*PI*pow(r,2);`
 	
`  printf("\n Area=%d", area);`
 	
`  return 0;`
`}`
	1. Syntax error
	At line number 11;
	undeclared variable area;
	should be declared as float;
	At line number 9;
	Undeclared radius;
	At line number 13;
	format specifier for area
	should be %f;
	
	2. Runtime Error
	At line number 9; &
	missing for radius r;
	
	3. Logical Error
	At line number 11; area of
	circle is PI*r*r;
	
	4. Linker Error
	At line number 11;
	no header file for pow()

```c
#include<stdio.h>
#include<math.h>

#define PI 3.1416

int main(){
    int r;
    float area;

    printf("Enter the radius: ");
    scanf("%lf", &radius);

    area= PI*pow(r,2);

    printf("\n Area=%.2f", area);

    return 0;
}
```

Engr. Mohammad Mamun Hossain
BSc.(Eng.) and MSc.(Thesis) in CSE, SUST
PhD (Pursuing) in CSE, RUET.
Assistant Professor, Dept. of CSE, BAUST

Contact: mhossain@baust.edu.bd
linkedIn:mamunsust12
Resources: mamunsust12@GitHub
Emergency: 01717690847