CS 332/532 Systems Programming

Lecture 3

- C Variables, Operators -

Professor : Mahmut Unan – UAB CS

Agenda

- Variables & Data Types
- const, define
- printf, scanf
- operators
- branching, loops

Announcement

- Problem with Vulcan server?
  - Email CS IT (BJ Wilson)

const

- A variable whose value cannot change during the execution of the program is called *constant*.

```
const int a = 10;
const double PI = 3.14;
int somefunction(const int *data, size_t size);
```

`printf()`

- The `printf()` function is used to display data to stdout (*st*andar*d* *out*put stream).
- The `printf()` function accepts a variable list of parameters.
  - The first argument is a format string, that is, a sequence of characters enclosed in double quotes, which determines the output format.
- The format string may contain escape sequences and conversion specifications.

Escape Sequences

- Escape sequences are used to represent nonprintable characters or characters that have a special meaning to the compiler. An escape sequence consists of a backslash (\) followed by a character.

Conversion Specifications

- A conversion specification begins with the % character, and it is followed by one or more characters with special significance. In its simplest form, the % is followed by one of the conversion specifiers below

Conversion Specifier	Meaning
c	Display the character that corresponds to an unsigned integer value.
d ή i	Display a signed integer.
u	Display an unsigned integer.
f	Display a floating-point number. The default precision is six digits.
s	Display a sequence of characters.
e, E	Display a floating-point number in scientific notation. The exponent is preceded by the chosen specifier e or E.
g, G	%e or %E form is selected if the exponent is less than −4 or greater than or equal to the precision. Otherwise, the %f form is used.
P	Display the value of a pointer variable.
x, X	Display an unsigned integer in hex form; %x displays lowercase letters (a–f), while %X displays uppercase letters (A–F).
O	Display an unsigned integer in octal.
n	Nothing is displayed. The matching argument must be a pointer to integer; the number of characters printed so far will be stored in that integer.
%	Display the character %.

```c
#include <stdio.h>
int main(void)
{
        int len;
        printf("%c\n", 'w');
        printf("%d\n", -100);
        printf("%f\n", 1.56);
        printf("%s\n", "some text");
        printf("%e\n", 100.25);
        printf("%g\n", 0.0000123);
        printf("%X\n", 15);
        printf("%o\n", 14);
        printf("test%n\n", &len);
        printf("%d%%\n", 100);
        return 0;
}
```

The program outputs:

`w` (the character constant must be enclosed in single quotes).
`-100`
`1.560000`
`some text` (the string literal must be enclosed in double quotes).
`1.002500e+002` (equivalent to $1.0025 \times 10^2 = 1.0025 \times 100 = 100.25$).
`1.23e-005` (because the exponent is less than `-4`, the number is displayed in scientific form).
`F` (the number `15` is equivalent to `F` in hex).
`16` (the number `14` is equivalent to `16` in octal).
`test` (since four characters have been printed before `%n` is met, the value `4` is stored into `len`.
`100%` (to display the `%` character, we must write it twice).

Precision

```c
#include <stdio.h>
int main(void)
{
        float a = 1.2365;
        printf("%f\n", a);
        printf("%.2f\n", a);
        printf("%.*f\n", 3, a);
        printf("%.f\n", a);
        return 0;
}
```

```
1.236500
1.24
1.237
1
```

scanf()

- The scanf() function is used to read data from stdin (*st*andar*d* *in*put stream) and store that data in program variables.

- The scanf() function accepts a variable list of parameters. The first is a format string similar to that of printf(), followed by the memory addresses of the variables in which the input data will be stored.

- Typically, the format string contains only conversion specifiers. The conversion characters used in scanf() are the same as those used in printf().

scanf()

```
int i;
float j;
scanf("%d%f", &i, &j);

char str[100];
scanf("%s", str);
```

```c
#include <stdio.h>
int main(void)
{
    char ch;
    int i;
    float f;
    printf("Enter character, int and float: ");
    scanf("%c%d%f", &ch, &i, &f);
    printf("\nC:%c\tI:%d\tF:%f\n", ch, i, f);
    return 0;
}
```

```
Enter character, int and float: s 17 22.6

C:s     I:17    F:22.600000
```

The assignment operator =

```
int a,b,c,d,e;
a=b=c=d=e=25;
```

or even the following is legal

```
a=25;
d=a + ( b = ( (e = a+10) + 40));
```

Arithmetic Operators

- + - / * %
- int/int = cuts off the decimal part

```
int a=7;
int b=5;
a/b will be equal to 1
```

also, be careful with the % operator

```
if ((n%2)==1) is dangerous**
if((n%2)!=0)  is safe
```

** if n is odd and negative

++ and --

- Similar to Java

```
int a=25;
a++; /* is equal to a = a+1; */
a--; /* is equal to a = a-1; */

int c=5,d;
d=c++;
```

or

```
d=++c;
```

Compound Assignment Operators

```c
#include <stdio.h>
int main(void)
{
        int a = 4, b = 2;
        a += 6;
        a *= b+3;
        a -= b+8;
        a /= b;
        a %= b+1;
        printf("Num = %d\n", a);
        return 0;
}
```

Comparisons

- > >= < <= != ==
- if (a == 10)

Logical Operators

- ! not operator, && operator, || operator

```
#include <stdio.h>
int main(void)
{
        int a = 4;
        printf("%d\n", !a);
        return 0;
}
```

The Comma Operator

- The comma (,) operator can be used to merge several expressions to form a single expression

```
#include <stdio.h>
int main(void)
{
        int b;
        b = 20, b = b+30, printf("%d\n", b);
        return 0;
}
```

Operator Precedence

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

https://www.tutorialspoint.com/cprogramming/c_operators_precedence.htm

```c
#include <stdio.h>

int main() {

    int a=10,b=20,c=30,d=40,e;

    e = (a + b) * c / d;       // (10+20)* 30 / 40
    printf("Value of (a + b) * c / d is : %d\n",  e );

    e = ((a + b) * c) / d;     // ((10+20)* 30 ) / 40
    printf("Value of ((a + b) * c) / d is  : %d\n" ,  e );

    e = (a + b) * (c / d);   // (10+20) * (30/40)
    printf("Value of (a + b) * (c / d) is  : %d\n",  e );

    e = a + (b * c) / d;     //  10 + (20*30)/40
    printf("Value of a + (b * c) / d is  : %d\n" ,  e );

    return 0;
}
```


https://www.tutorialspoint.com/cprogramming/c_operators_precedence.htm

```c
#include <stdio.h>

int main() {

    int a=10,b=20,c=30,d=40,e;

    e = (a + b) * c / d;       // (10+20)* 30 / 40
    printf("Value of (a + b) * c / d is : %d\n",  e );

    e = ((a + b) * c) / d;     // ((10+20)* 30 ) / 40
    printf("Value of ((a + b) * c) / d is  : %d\n" ,  e );

    e = (a + b) * (c / d);   // (10+20) * (30/40)
    printf("Value of (a + b) * (c / d) is  : %d\n",  e );

    e = a + (b * c) / d;     //  10 + (20*30)/40
    printf("Value of a + (b * c) / d is  : %d\n" ,  e );

    return 0;
}
```

22

22

0

25

https://www.tutorialspoint.com/cprogramming/c_operators_precedence.htm

if else else if

```
#include <stdio.h>
int main(void)
{
        int a = 10, b = 20, c = 30;
        if(a > 5)
        {
                if(b == 20)
                        printf("1 ");
                if(c == 40)
                        printf("2 ");
                else
                        printf("3 ");
        }
        else
                printf("4\n");
        return 0;
}
```

switch statement

```c
#include <stdio.h>
int main(void)
{
        int a;
        printf("Enter number: ");
        scanf("%d", &a);
        switch(a)
        {
                case 1:
                        printf("One\n");
                break;
                case 2:
                        printf("Two\n");
                break;
                default:
                        printf("Other\n");
                break;
        }
        printf("End\n");
        return 0;
}
```

for loop

```c
#include <stdio.h>
int main(void)
{
    int a;
    for(a = 0; a < 5; a++)
    {
        printf("%d ", a);
    }
    return 0;
}
```

The `break` Statement

```
#include <stdio.h>
int main(void)
{
        int i;
        for(i = 1; i < 10; i++)
        {
                if(i == 5)
                        break;
                printf("%d ", i);
        }
        printf("End = %d\n", i);
        return 0;
}
```

The continue Statement

```
#include <stdio.h>
int main(void)
{
        int i;
        for(i = 1; i < 10; i++)
        {
                if(i < 5)
                        continue;
                printf("%d ", i);
        }
        return 0;
}
```

while loop

```c
#include <stdio.h>
int main(void)
{
        int i = 10;
        while(i != 0)
        {
                printf("%d\n", i);
                i--;
        }
        return 0;
}
```

do-while loop

```
#include <stdio.h>
int main(void)
{
        int i = 1;
        do
        {
                printf("%d\n", i);
                i++;
        } while(i <= 10);
        return 0;
}
```

References

- https://www.tutorialspoint.com/cprogramming/c_constants.htm
- C From Theory to Practice - 2nd edition, Nikolaos D. Tselikas and George S. Tselikis

Data Types in C

Integer Types

The following table provides the details of standard integer types with their storage sizes and value ranges –

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

https://www.tutorialspoint.com/cprogramming/c_data_types.htm

Floating-Point Types

The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision –

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

The header file float.h defines macros that allow you to use these values and other details about the binary representation of real numbers in your programs. The following example prints the storage space taken by a float type and its range values –

https://www.tutorialspoint.com/cprogramming/c_data_types.htm

C - Type Casting

- https://www.tutorialspoint.com/tpcg.php?p=LlUZrX
- https://www.tutorialspoint.com/tpcg.php?p=VkqrXY
- https://www.tutorialspoint.com/tpcg.php?p=MImFCX

https://https://www.tutorialspoint.com/cprogramming/c_type_casting.htm