# Chapter 5

The C Programming Language
Types, Operators and Expressions

Computer Application

Outline

➡ • **Data Types and Sizes**
  • Constants
  • Declaration and Qualifier
  • Arithmetic Operations
  • Relational and Logic Operations
  • Type Conversions
  • Increment and Decrement Operations
  • Bitwise Operations
  • Assignment and Expressions
  • Conditional Expressions
  • Order of Evaluation
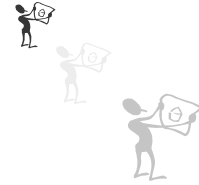
Data Type and Size

Type	signed	unsigned
char	8 2'com	8
short	16 2'com	16
long	32 2'com	32
int	16/32 2'com	16/32
float	32 fp	x
double	64 fp	x
long double	80 fp	x

Constant

Type	signed	unsigned
short	1234	1234U
long	1234L	1234UL
float	123.4f	x
double	123.4	x
long double	123.4L	x

Type	hexa	octal
integer	0x30FF	0377

Amazing Example

```
#include <ansi_c.h>
void main()
    {
    char x;
    unsigned char y;
    float z;
    int A;
    double B, C, D;

    x = 127;
    y = 1;
    z = -1.75;

    x = x+2;
    y = y-2;
    memcpy (&A, &z, 4);

    B = 2147483647 + 2;
    C = 2147483647 + 2U;
    D = 2147483647;
    D = D + 2;
    printf("%d   %d   %08x\n", x, y, A);
    printf("%12.0f   %12.0f\n", B, C);
    printf("%12.0f\n", D);
    getchar();
    }
```
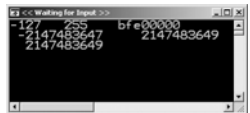
```
-127    255    bf e00000
-2147483647            2147483649
2147483649
```

Amazing Example FP

```
##include <ansi_c.h>
void main()
    {
    int k;
    float X, Y, Z;
    double A, B, C;

    X = 1.0;
    for (k=0; k<100000; k++)
            X+=1e-8;
    Y = 0.0;
    for (k=0; k<100000; k++)
            Y+=1e-8;
    Z = 1.0;
    Z += 100000*1e-8;
    printf("%12.8f   %12.8f   %12.8f\n", X, Y, Z);

    A = 1.0;
    for (k=0; k<100000; k++)
            A+=1e-8;
    B = 0.0;
    for (k=0; k<100000; k++)
            B+=1e-8;
    C = 1.0;
    C += 100000*1e-8;
    printf("%12.8f   %12.8f   %12.8f\n", A, B, C);
    }
```
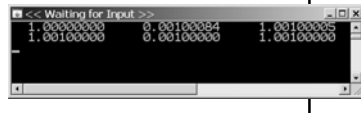
```
1.00000000    0.00100084    1.00100005
1.00100000    0.00100000    1.00100000
```

Constant

Type	symbol	ascii
Character	'A'	'\xhh'

Type	symbol
String (Fin by Null)	"Hello World"

String constant stored in memory continuously

'X' is not "X"

Character & String Example

```
#include <ansi_c.h>
void main()
{
  char c, *s="Hello World";

  c = 65;
  printf("%c  %d\n", c, c);

  c = c+1;
  printf("%c  %d\n", c, c);

  c = 'A';
  printf("%c  %d\n", c, c);

  printf("%c  %c%c%d\n", c, 10, 13, c);

  c = '!' * 2 + 4;
  printf("%c  %d\n", c, c);

  printf("%s\n", s);
  printf("%p\n", s);
  printf("%s\n", (char*)0x4280A8);
  printf("%d\n", s[11]);
  printf("%p  %p\n", "Hello World", "Hello World");
  getchar();
}
```
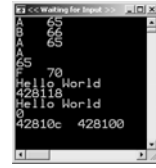
---

enumeration Constant

enum name {RED, GREEN, BLUE}

    ↑   ↑   ↑

    0   **1**   **2**

enum name {RED=7, GREEN, BLUE}

    ↑   ↑   ↑

    7   **8**   **9**

Enum & Base Example

```
#include <ansi_c.h>
void main()
{
  enum Color {RED, GREEN, BLACK=10, YELLOW, BLUE=48};
  enum Color MyColor1, MyColor2;

  char a,b,c,d,e;

  a = '0';
  b = 48;
  c = 0x30;
  d = 060;
  e = BLUE;

  MyColor1 = GREEN;

  printf("%d  %d  %d  %d\n", RED, GREEN, BLACK, YELLOW);
  printf("%d\n", MyColor1);
  printf("%d  %d  %d  %d  %d\n", a,b,c,d,e);

  MyColor1 = 10;
  MyColor2 = 2;

  getchar();
}
```

Declaration

StorageClass **Qualifier** DataType VarName = *InItVal*;

auto static extern register	const volatile	char short long int float double signed unsigned

---

Declaration Storage-Class and Qualifier

- **auto** – Identified - in the block. Life - in the block.
- **static** – Identified - in the block. Life - always.
- **extern** – Identified - anywhere. Life - always.
- **register** - auto variable recommended to CPU register
- By default auto variable init to *garbage*.
- By default extern and static variable init to zero.
- auto variable init multi-time to any value.
- extern and static variable init one's to constant value.

- **const** is a variable that not change and init one's.
- **volatile** is a variable that change by hardware.

Storage-Class and Qualifier Example

```
#include <ansi_c.h>
int Ex;
static int SEx;
const int CEx = 100;
int Tx = 10;

void f1(void)
  {
  char x1, x2 = 1;
  char static x3, x4 = 1;
  int N = 1000;
  x1++; x2++; x3++; x4++;
  {
  int N = x2 + x4;
  printf("%d  %d  %d  %d\n", x1, x2, x3, x4, N);
  }
  }
void main()
  {
  int i;
  int Tx = 20;
  for (i=0; i<5; i++)
         f1();
  printf("\n\n%d  %d  %d  %d\n", Ex, SEx, CEx, Tx);
  getchar();
  }
```

```
21  2  1  2  4
21  2  2  3  5
21  2  3  4  6
21  2  4  5  7
21  2  5  6  8

0   0  100  20
```

Arithmetic Operator

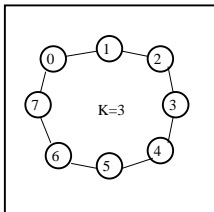Binary Operators

$$+ \quad - \quad * \quad / \quad \%$$

Unary Operators

$$+ \quad -$$

Arithmetic Operator

- The operators are associative to the operands.
- The minus is in 2-compliment representation.
- The integer arithmetic is modulo 2^k.

K=3 (circle diagram with nodes 0,1,2,3,4,5,6,7)

$$1 + 3 = 4 = 100 = -4$$
$$6 + 3 = 1 = 001 = +1$$
$$1 - 2 = 7 = 111 = -1$$

Arithmetic Example

```
#include <ansi_c.h>
void main()
  {
  int A, B, C;
  double X, Y;

  A = 3 / 2;
  X = 3 / 2;
  printf("%d  %f\n", A, X);

  A = 3.0 / 2;
  X = 3.0 / 2;
  printf("%d  %f\n", A, X);

  A = 3 / 5 * 10;
  B = 3 * 10 / 5;
  printf("%d  %d\n", A, B);

  A = 3 / 5 * 10.0;
  B = 3 / 5.0 * 10;
  printf("%d  %d\n", A, B);

  A = 256 * 256 * 256 / 256 / 256 / 256;
  B = 256 / 256 * 256 / 256 / 256 * 256;
  printf("%d  %d\n", A, B);

  A =  7 % 4;
  B = -7 % 4;
  printf("%d  %d\n", A, B);

  getchar();
  }
```

```
1  1.000000
1  1.500000
0  6
0  6
0  256
3  -3
```

Relational and Logical Operator

$$> \quad >= \quad < \quad <= \quad == \quad !=$$

$$\&\& \quad || \quad !$$

- **The type of Relational and Boolean expression is integer.**
- **The evaluation of Relational and Boolean expression is 0 or 1**
- **The value of operand for logical expression is:**
  0 – False   Non 0 – True

Relational and Logical Operator Example

```
#include <ansi_c.h>
void main()
  {
  int A, B, C;

  A = 3 > 2;
  B = 1 > 2;
  printf("%d  %d\n", A, B);

  A = 3 && 7;
  B = 2 || 0;
  printf("%d  %d\n", A, B);

  A = (3 > 0) * 5 + 2;
  B = ((7 == 1) + 3) * 2;
  printf("%d  %d\n", A, B);

  A = 5;
  B = !!A;
  printf("%d  %d\n", A, B);

  getchar();
  }
```

```
1  0
1  1
7  6
5  1
```

Assignment and Expressions

The assignment has a value of the expression.

X = Expressions

Shortened assignment

X *OP*= Expressions

X = X *OP* Expressions

Assignment Example

```
#include <ansi_c.h>
void main()
{
int A, B, C;

A = 1;  B = 2;
printf("%d  %d\n", A, B);
printf("%d  %d\n", A=5, B=9);

A = 3 * (B=2);
printf("%d  %d\n", A, B);

A = 3 * (B==3);
printf("%d  %d\n", A, B);

A = 4;   B = 8;
A += 2;
B /= 4;
printf("%d  %d\n", A, B);

A = B = C = 0;
printf("%d  %d  %d\n", A, B, C);

A *= B-= C+=3;
printf("%d  %d  %d\n", A, B, C);

A = (B=5, C=7);
printf("%d  %d  %d\n", A, B, C);

getchar();
}
```

Increment and Decrement Operations

Post

Var++   Var--

Pre

++Var   --Var

Increment and Decrement Example

```
#include <ansi_c.h>
void main()
{
int A, B, C;

A = 5;
B = A++;
printf("%d  %d\n", A, B);

A = 5;
B = ++A;
printf("%d  %d\n", A, B);

A = 5;
B = A-- * 3;
printf("%d  %d\n", A, B);

A = 8;
B = 8;
printf("%d  %d\n", A++, ++B);

getchar();
}
```

Conditional Expressions

expr1 ? expr2 : expr3

if (exp1 == TRUE)
    Expr2;
else
    Expr3;

Conditional Expressions Example

```
#include <ansi_c.h>
void main()
{
int A, B, C;

A = 1;
B = 2;
printf("%d\n", A>B ? A : B);

A = 50;
B = 30;
C = A>B ? A : B;
printf("%d\n", C);

A = -10;
C = A>=0 ? A : -A;
printf("%d\n", C);

A = 1;
printf(A ? "Hello True\n" : "Hello False\n");
A = !A;
printf(A ? "Hello True\n" : "Hello False\n");

getchar();
}
```