Lecture 4

Fundamental data types, Chapter 2

Objectives

- The comma operator
- The **`sizeof`** operator
- The header **<limits.h>**
- The **`typedef`** statement
- Table of type conversation code
- Increment and decrement of unary operators
- Problem solving.

Category	Type	Description	Bytes	Minimum	Maximum
Integers	int (or signed int)	Signed integer (of at least 16 bits)	4 (2)	-2147483648	2147483647
	unsigned int	Unsigned integer (of at least 16 bits)	4 (2)	0	4294967295
	char	Character (can be either signed or unsigned depends on implementation)	1		
	signed char	Character or signed tiny integer (guarantee to be signed)	1	-128	127
	unsigned char	Character or unsigned tiny integer (guarantee to be unsigned)	1	0	255
	short (or short int) (or signed short) (or signed short int)	Short signed integer (of at least 16 bits)	2	-32768	32767
	unsigned short (or unsigned shot int)	Unsigned short integer (of at least 16 bits)	2	0	65535
	long (or long int) (or signed long) (or signed long int)	Long signed integer (of at least 32 bits)	4 (8)	-2147483648	2147483647
	unsigned long (or unsigned long int)	Unsigned long integer (of at least 32 bits)	4 (8)	0	same as above
	long long (or long long int) (or signed long long) (or signed long long int)	Very long signed integer (of at least 64 bits)	8	-2^{63}	$2^{63}-1$
	unsigned long long (or unsigned long long int)	Unsigned very long integer (of at least 64 bits)	8	0	$2^{64}-1$
Real numbers	float	Floating-point number, ≈7 digits (IEEE 754 single-precision floating point format)	4	3.4e38	3.4e-38
	double	Double precision floating-point number, ≈15 digits (IEEE 754 double-precision floating point format)	8	1.7e308	1.7e-308
	long double	Long double precision floating-point number, ≈19 digits (IEEE 754 quadruple-precision floating point format)	12 (8)		
Wide Characters	wchar_t	Wide (double-byte) character	2 (4)	Wide Characters	wchar_t

- The **comma** operator (,)

The comma (,) operator is:

- Left associative: the expressions are evaluated from left to right.
- It is used to merge several expressions to form a single expression.
- The type and the value of the entire expression are those of the last expression evaluated.

Example:

```
#include <stdio.h>
int main()
 {
 int a;
 a=10, a=a+50, printf(" The sum is %d.\n", a);
 return 0;
 }
```

- The **comma** operator (`,`)

- The **comma** operator (,)

- Left associative: the expressions are evaluated from left to right.

- The **comma** operator (,)

- Left associative: the expressions are evaluated from left to right.
- It is used to merge several expressions to form a single expression.

- The **comma** operator (,)

- Left associative: the expressions are evaluated from left to right.
- It is used to merge several expressions to form a single expression.
- The type and the value of the entire expression are those of the last expression evaluated.

- The **comma** operator (,)

- Left associative: the expressions are evaluated from left to right.
- It is used to merge several expressions to form a single expression.
- The type and the value of the entire expression are those of the last expression evaluated.

Example:

- The **comma** operator (,)

- Left associative: the expressions are evaluated from left to right.
- It is used to merge several expressions to form a single expression.
- The type and the value of the entire expression are those of the last expression evaluated.

Example:

```
#include <stdio.h>
int main()
 {
 int a;
 a=10, a=a+50, printf(" The sum is %d.\n", a);
 return 0;
 }
```

The **sizeof** operator gets the size of the operand bytes.

Example: Write C code to find out the size of the operands.

```
#include <stdio.h>
int main() {
 printf("sizeof(char) is %d bytes.\n", sizeof(char));
 printf("sizeof(short) is %d bytes\n", sizeof(short));
 printf("sizeof(int) is %d bytes\n", sizeof(int));
 printf("sizeof(long) is %d bytes\n", sizeof(long));
 printf("sizeof(long long) is %d bytes\n", sizeof(long long));
  printf("sizeof(float) is %d bytes.\n", sizeof(float));
  printf("sizeof(double) is %d bytes.\n", sizeof(double));
  printf("sizeof(long double) is %d bytes.\n",
sizeof(long double));
 return 0; }
```

The header **<limits.h>**

The **library macros values** are implementation-specific and defined with the **#define directive:**

MacroValueDescription:

CHAR_BIT8

SCHAR_MIN-128

The **C99 standard** also specifies the <stdint.h> header file, providing names and limits for explicitly-sized platform-independent integer data types such as **int32_t** for a 32-bit signed integer.

The header **<limits.h>** tests the length of the integers.

Example: Write C code to test the limits of the integers.

```
#include <stdio.h>
#include <limits.h>
int main() {
  printf("int max = %d\n", INT_MAX);
  printf("int min = %d\n", INT_MIN);
  printf("unsigned int max = %u\n", UINT_MAX);

   printf("long max = %ld\n", LONG_MAX);
   printf("long min = %ld\n", LONG_MIN);
   printf("unsigned long max = %lu\n", ULONG_MAX);

  printf("long long max = %lld\n", LLONG_MAX);
  printf("long long min = %lld\n", LLONG_MIN);
  printf("unsigned long long max = %llu\n", ULLONG_MAX);

    printf("Bits in char = %d\n", CHAR_BIT);
    printf("char max = %d\n", CHAR_MAX);
    printf("char min = %d\n", CHAR_MIN);
  printf("signed char max = %d\n", SCHAR_MAX);
  printf("signed char min = %d\n", SCHAR_MIN);
  printf("unsigned char max = %u\n", UCHAR_MAX);
return 0; }
```

The header **<limits.h>** tests the length of the integers.

Example: Write C code to test the limits of the integers.

```
#include <stdio.h>
#include <limits.h>
int main() {
  printf("int max = %d\n", INT_MAX);
  printf("int min = %d\n", INT_MIN);
  printf("unsigned int max = %u\n", UINT_MAX);

   printf("long max = %ld\n", LONG_MAX);
   printf("long min = %ld\n", LONG_MIN);
   printf("unsigned long max = %lu\n", ULONG_MAX);

  printf("long long max = %lld\n", LLONG_MAX);
  printf("long long min = %lld\n", LLONG_MIN);
  printf("unsigned long long max = %llu\n", ULLONG_MAX);

    printf("Bits in char = %d\n", CHAR_BIT);
    printf("char max = %d\n", CHAR_MAX);
    printf("char min = %d\n", CHAR_MIN);
  printf("signed char max = %d\n", SCHAR_MAX);
  printf("signed char min = %d\n", SCHAR_MIN);
  printf("unsigned char max = %u\n", UCHAR_MAX);
return 0; }
```

Example:

Write C code to print the system limitations.

```
#include <stdio.h>
#include <limits.h>
#include <float.h>
int main(void)    {
    //Print integer type maximums.
    printf("short maximum: %i \n",SHRT_MAX);
    printf("int maximum: %i \n",INT_MAX);
    printf("long maximum: %li \n\n",LONG_MAX);
    //Print float precision, range, maximum.
    printf("float precision digits: %i \n",FLT_DIG);
    printf("float maximum exponent: %i \n", FLT_MAX_10_EXP);
    printf("float maximum: %e \n\n",FLT_MAX);
    //Print double precision, range, maximum.
    printf("double precision digits: %i \n",DBL_DIG);
    printf("double maximum exponent: %i \n", DBL_MAX_10_EXP);
    printf("double maximum: %e \n\n",DBL_MAX);
    //Print long precision, range, maximum.
    printf("long double precision digits: %i \n",LDBL_DIG);
    printf("long double maximum exponent: %i \n", LDBL_MAX_10_EXP);
    printf("long double maximum: %Le \n\n",LDBL_MAX);
    return 0;       }
```

The **typedef** statement

- To assign an **"unsigned int"**
- The **typedef** statement is used to create a new name for an existing type.

Example:

- Create a new type for "***unsigned int***"
- Immediately after **#include**, place **typedef**:

```
typedef unsigned int uint;
```

Many C compilers define **size_t** as a **typedef** of **unsigned int** or `typedef unsigned int type_t;`

Table of type conversation code

Type	Type Conversion Code	Type & Format
Integers	%d (or %i)	(signed) int
	%u	unsigned int
	%o	int in octal
	%x, %X	int in hexadecimal (%X uses uppercase A-F)
	%hd, %hu	short, unsigned short
	%ld, %lu	long, unsigned long
	%lld, %llu	long long, unsigned long long
Floating-point		
	%f	float in fixed notation
	%e, %E	float in scientific notation
	%g, %G	float in fixed/scientific notation depending on its value
	%f, %lf (printf), %lf (scanf)	double: Use %f or %lf in printf(), but %lf in scanf().
	%Lf, %Le, %LE, %Lg, %LG	long double
Character	%c	char
String	%s	string

Field Width

```
int number = 456789;
printf("number=%d.\n", number);
        //number=456789.
printf("number=%8d.\n",number);
        //number= 456789.
printf("number=%3d.\n", number);
        //Very short, so it is ignored.
        //number=456789.
```

Precision (decimal places) for floating-point numbers

```
double value = 123.14159265;
printf("value=%lf;\n", value);
   //value=123.141593;
printf("value=%6.2lf;\n", value);
   //value=123.14;
printf("value=%9.4lf;\n", value);
   //value= 123.1416;
printf("value=%3.2lf;\n", value);
   // Since the field-width is too short, it is ignored.
   //value=123.14;
```

Increment /decrement of unary operators

```
#include <stdio.h>
int main() {
 int grade = 20;          // declare and assign
 printf("%d\n", grade); // 20
grade++;                  // increase by 1 (post-increment)
printf("%d\n", grade);   // 21
 ++grade;                 // increase by 1 (pre-increment)
 printf("%d\n", grade); // 22
grade = grade + 1;        // also increase by 1 (grade+=1)
printf("%d\n", grade);   // 23
 grade--;                 // decrease by 1 (post-decrement)
 printf("%d\n", grade); // 22
--grade;                  // decrease by 1 (pre-decrement)
printf("%d\n", grade);   // 21
 grade = grade - 1;       // also decrease by 1 (grade-=1)
 printf("%d\n", grade); // 20
return 0;}
```

Increment/decrement of unary operators, cont.

- The increment/decrement unary operator is placed before the operand (prefix operator), or after the operand (postfix operator).
- **`++var` is a `pre-increment` `var`**
  Use the new value of *`var:`* `y=++x;` same as `x=x+1; y=x;`
- **`var++` is a `post-increment`**
  Use the old value of **`var`**, then increment **`var`**
  `y = x++;` same as `oldX=x; x=x+1; y=oldX;`
- **`--var` is a `pre-decrement`**
  `y = --x;` same as `x=x-1; y=x;`
- **`var--` is a `post-decrement`**
  `y = x--;` same as `oldX=x; x=x-1; y=oldX;`

Increment/decrement of unary operators, cont.

```
#include <stdio.h>
int main() {
 int x = 5;
 printf("  %d\n\n", x++);
// x=5; increment x=6; prints old x=5.
 x = 5;
 printf("  %d\n", ++x);
// increment x=6; prints x is 6.
 return 0;}
```

#define token [value]

Defining a constant : `#define token [value]`

- When defining a constant, a value for that constant may not be provided, the token will be replaced with blank text and "defined" for the purposes of #ifdef and ifndef.

- If a value is provided, the token will be replaced with the remainder of the text on the line (See C preprocessor for the list of gotchas).

Defining a parameterized macro:

`#define token(<arg> [, <arg>s ... ]) statement`

`#define MAX(a, b) ((a) > (b) ? (a) : (b))`

To define a multiline macro, each line before the last is to end with a \, which will result in a line continuation.

Example:

Write a C program to use 0 and 1 to find for the leap year.

```
#include <stdio.h>
int main(){
 char feb;
 int days;
 printf("Enter 0 if the year is not a leap year
otherwise enter 1: ");
 scanf("%c",&feb);
 days=(feb=='0')?28:29;
//If (feb=='l') is true, days are equal to 29.
//If (feb=='0') is false, days are equal to 28.
 printf("Number of days in February= %d",days);
 return 0;}
```

To do for practice:

Write a C program to create the following output:

```
*
**
***
****
*****
```

Example:

Write a C program to print all 256 ASCII values.

```
#include<stdio.h>
main()
{
  int i;
  char ch;
  for(i=0; i<256; i++)
    {
    printf("%c ", ch);
    ch = ch + 1;
    }
}
```

To do for practice:

Write a C program to create the following output:

```
**********
****    ****
***       ***
**          **
*             *
```

Example:

Write a C program to print the first 7 natural numbers using a “for” loop

```
int main()
{
    int i = 1;
    for (i = 1; i <= 7; i++)
        {
        printf(" %d ", i);
        }
    return (0);
}
```

Example:

Write a C program to print the first 10 natural numbers using a "while" loop.

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

Example:

Write a C program to print the first 12 natural numbers using a “do-while” loop.

```
#include<stdio.h>
int main()
{
   int i = 1;
   do
     {
     printf(" %d ", i);
     i++;
     }
       while (i <= 12);
   return (0);
}
```

Example:

Write a C program to convert an integer from decimal number system(base-10) to binary number system(base-2). Size of integer is assumed to be 32 bits.

```
#include <stdio.h>
int main() {
  int n, c, k;
  printf("Enter an integer in decimal number system\n");
  scanf("%d", &n);
  printf("The integer %d in binary number system is:\n", n);
  for (c=31;c>=0;c--) {
    k = n >> c; //Number is shifted by 1 bit
    if (k & 1) //It is either 1 or 0, depending on the least significant bit of k:
if the last bit is 1, the result of k & 1 is 1; otherwise, it is 0. This is a bitwise AND
operation.
      printf("1");
    else
      printf("0"); }
  printf("\n");
  return 0;}
```

Example:

Write a C program to generate a table of conversions from Fahrenheit to Kelvin for values from 0 degrees F to 200 degrees Fahrenheit. It allows the user (user defined program) to enter the increment between lines.

```
#include <stdio.h>
int main(void)
{
    //Define and initialize the variables.
    double farenheit=0, increment=0, kelvin;
    //Prompt the user for increment.
    while (increment <= 0)
    {
        printf("Enter increment for table:");
        scanf("%lf",&increment);
    }
    //Print the title and the table.
    printf("Farenheit to Kelvin \n");
```

cont.

Example:

Write a C program to generate a table of conversions from Fahrenheit to Kelvin for values from 0 degrees F to 200 degrees Fahrenheit. It allows the user (user defined program) to enter the increment between lines.

```
do
    {
    kelvin = (5.0/9.0)*(farenheit + 459.67);
    printf("%4.2f F        %4.2f K \n",farenheit,kelvin);
    farenheit += increment;
    }
    while (farenheit <= 200.0);
    //Exit program.
    return 0;
}
```

To do for practice:

The C program below uses linear interpolation to compute the freezing temperature of seawater.

```
#include <stdio.h>
#include <math.h>
int main(void) {
//Declare variables.
double a, f_a, b, f_b, c, f_c;
//Get user input from the keyboard.
printf ("Use ppt for salinity values. \n");
printf ("Use degrees F for temperatures. \n");
printf ("Enter first salinity and freezing temperature: \n");
scanf ("%lf %lf",&a,&f_a);
printf ("Enter second salinity and freezing temperature: \n");
scanf ("%lf %lf",&c,&f_c);
printf ("Enter new salinity: \n");
scanf ("%lf",&b);
//Use linear interpolation to compute new freezing temperature.
f_b = f_a + (b-a)/(c-a)*(f_c - f_a);
//Print new freezing temperature.
printf("New freezing temperature in degrees F: %4.1f \n",f_b);
return 0;  }
```

cont.

To do for practice, cont.

1. Modify the program for the linear interpolation so that it determines the freezing temperatures to go with the following salinity measurements in ppt:
   3 8.5 19 23.5 26.8 30.5

1. Modify the program for the linear interpolation so that it converts and prints the new temperature in degrees Centigrade. (Recall the relation between the temperature in degrees Fahrenheit and the temperature in degrees Centigrade.)