Computer Programming Using C
COP 3275 - Summer 2017

Lecture 11: Type cast and Array

By Ahmed E. Khaled

/* Programming */
Recap to previous lecture!

- Integer Types
- Float Types
- Char Type
Type Conversion

- When operands of different types are mixed in expressions, the compiler may have to generate instructions that change the types of some operands so that hardware will be able to evaluate the expression.

- Because the compiler handles these conversions automatically, without the programmer’s involvement, they’re known as *implicit conversions*.

- C also allows the programmer to perform *explicit conversions*, using the *cast operator*. 
char c;
short int s;
int i;
unsigned int u;
long int l;

i = i + c;
    /* c is converted to int */

i = i + s;
    /* s is converted to int */

u = u + i;
    /* i is converted to unsigned int */

l = l + u;
    /* u is converted to long int */
unsigned long int ul;
float f;
double d;
long double ld;

ul = ul + l;
    /* l is converted to unsigned long int */

f = f + ul;
    /* ul is converted to float */

d = d + f;
    /* f is converted to double */

ld = ld + d;
    /* d is converted to long double */
char c;
int i;
float f;
double d;

i = c;
    /* c is converted to int   */

f = i;
    /* i is converted to float   */

d = f;
    /* f is converted to double */
Casting (explicit conversion)

- A cast expression has the form
  \( (\text{type-name}) \ expression \)

\textit{type-name} specifies the type to which the expression should be converted.

```plaintext
float f_value = 100.0;
int i_value = 10;
i_value = (int) f_value;
```
Casting

• Cast expressions also let us force the compiler to perform conversions.

• Example:

```c
float quotient;
int dividend, divisor;

quotient = dividend / divisor;
```

• To avoid truncation during division, we need to cast one of the operands:

```c
quotient = (float) dividend / (float) divisor;
```
Arrays in C
first step towards *Data structures* !
• So far, the only variables we’ve seen are *scalar:* capable of holding *a single data item.*

• C also supports *aggregate* variables, which can store *collections of values.*

• There are two kinds of aggregate variable in standard C: *arrays* and *structures.*
One-Dimensional Arrays

• **array** is a *data structure* containing a number of data values, all of which have the *same type*.

  • These values, known as *elements*, can be individually selected *by their position* within the array.

  • The simplest kind of array has just one dimension (1D).

• Array A:

  
  | 0 | 1 | 2 | …… | n-1 |

  

One-Dimensional Arrays

- To declare an array, we must specify the type of the array’s elements and the number of elements:
  
  ```
  type name [ size ] ;
  ```

  - Int
  - Char
  - Float
  - Double
  - ....

  Size is the number of elements (integer constant expression)

  The name follows the naming rules of identifiers
Array size

int abc [10];

• The elements of an array of length $n$ are indexed from 0 to $n - 1$.

• The elements of abc are:

Array Subscripting

• To access an array element, write the array name followed by an integer value in square brackets.

• This is referred to as *subscripting* or *indexing* the array.

```c
int abc [3];
abc[0] = 100;
int x = abc[0];
```

• In general, if an array contains elements of type $T$, then each element of the array is treated as if it were a variable of type $T$. 
• Develop a program that initializes array of size 50 with value of 20.

```c
for (i = 0; i < 50; i++){
    a[i] = 20;
}
```

• Develop a program that reads user input for an array of size 50.

```c
for (i = 0; i < 50; i++){
    scanf("%d", &a[i]);
}
```
• Develop a program that calculate the summation of array elements, array size is 50.

```c
int i;
int sum = 0;
for (i = 0; i < N; i++){
    sum += a[i];
}
```