Introduction to C Programming

Section 5

sizeof

- The sizeof keyword returns the number of bytes of the given expression or type
  - returns an unsigned integer result

- sizeof variable_Identifier;
- sizeof (variable_Identifier);
- sizeof (Data_Taype);

- Example:

```
int x;
printf("size of x = %d", sizeof x);//4
printf("size of long long = %d", sizeof(long long));//8
printf("size of x = %d", sizeof (x));//4
```

Type Casting

- What is the casting?
  - When the type of variable and value are not the same
  - Example: Assigning double value to integer variable

Implicit casting

- Implicit
  - We don’t say it
  - But we do it
  - carried out by compiler automatically

```
char f2= 50e6; /*cast from double to char */

 int i= 98.01;/*cast from double to int */

float f = 65.6;

    int i = f;  //f = (int)f;

    char c = i;  // c = (char)i;
```

Explicit casting

- Explicit
  - We say it
  - And we do it
  - carried out by programmer using casting

```
int i=(int)98.1;/*cast from double to int */
char c = (char) 90;/* cast from int to char */
int k, i = 7;
float f = 10.14;
char c = 'B';
k = (i + f) % 3; // error
k = (int)(i + f) % 3;
```

Casting effects

- Casting from small types to large types
  - There is not any problem
  - No loss of data

```
int i;
short s;
float f;
double d;
s = 'A';    // s = 65
i = 'B';    // i = 66
f = 4566;   // f = 4566.0
d = 5666;   // d = 5666.0
```

Casting effects

- Casting from large types to small types
  - Data loss is possible
  - Depends on the values

```
float f = 65536;        // 65536.0
double d = 65536;       // 65536.0
short s = 720;          // 720

char c = (char) 65536;      // c = 0
short s = (short) 65536;    // s = 0
int i = 1.22;       // i = 1
int i = 1e23;       // i = ???
```

Casting effects

- Casting to Boolean
  - If value is zero > false
  - If values is not zero > true

```
bool b2 = 'a', b3 = -9, b4 = 4.5;   //true
bool b5 = 0, b6 = false; b7 = '\0'; //false
```

Constant Variables

- Constants
  - Do not want to change the value
  - Example: pi = 3.14
- We can only *initialize* a constant variable
  - We MUST initialize the constant variables (why?!)
-

Constant

- **const [Data_Type] Identifier = constant_value;**
  - const p = 3; // const int p = 3;
  - const p;
    p = 3.14; // compile error
  - const p = 3.14; // p = 3 because default is int
  - const float p = 3.14;

Constant

```
const int STUDENTS = 38;
const long int
MAX_GRADE =20;
int i;
i = MAX_GRADE;
STUDENTS = 39;//ERROR
```

Definitions

- Another tool to define constants
  - Definition is not variable
    - We define definition, don't declare them
  - Pre-processor replaces them by their values before compiling

```
#define STUDENTS  38  int
main(void){
int i;
i = STUDENTS
STUDENTS = 90; //ERROR! What compiler sees: 38 = 90
```

Constant

```
#define   Identifier    constant_value
#define   PI     3.14
#define   ERROR   "Disk error "
#define   ONE     1
#define   TWO    ONE + ONE
```

```c
#include <stdio.h>  // (preprocessor )

#define   PI    3.14  // PI  constant
(preprocessor )

//  calculating  area of circle
int main()
{
    /* variable definition */
    float  Radius;
    float  Area = 0;
    // get radius of circle form user
    printf("Enter Radius :\n");
    scanf("%f",  &float );
    // calculating  area of circle
    Area  =  PI  *  Radius  *  Radius;
    printf("Area  =  %f",  Area );

    system("Pause");
    return 0;
}
```