# Computer Programming & Problem Solving

CS100

**Mrs Sanga G. Chaki**
**Department of Computer Science and Engineering**
**National Institute of Technology, Goa**
**May, 2023**


# Datatypes Revisited

# Why Data Types are needed?

1. Type of variable present in a program

2. Determine the space that a variable occupies in storage – size of the datatype

3. The way in which the stored bit pattern are interpreted is also determined

# Primary Data Types

1. Integer - storing various whole numbers
2. Float - all the real number values or decimal points
3. Character - It refers to all ASCII character sets as well as the single alphabets
4. Double - These include all large types of numeric values that do not come under either floating-point data type or integer data type.
5. Void - refers to no values at all. We mostly use this data type when defining the functions in a program.

# Datatype modifiers

1. **To provide different ranges to existing datatypes**
2. **4 types:**
    a) **Short**
    b) **Long**
    c) **Signed**
    d) **Unsigned**

# Integers

1. In general, space for ints =
    a) 2 memory locations = 2 bytes = 16 bits
    b) Or 4 bytes
2. Variants: to provide integers with different ranges
    a) Short
    b) Long
    c) Signed
    d) Unsigned

# Integers - Variants

**Some rules that are followed:**

**shorts** are at least 2 bytes big
**longs** are at least 4 bytes big
**shorts** are never bigger than **ints**
**ints** are never bigger than **longs**

| Compiler | short | int | long |
|---|---|---|---|
| 16-bit (Turbo C/C++) | 2 | 2 | 4 |
| 32-bit (Visual C++) | 2 | 4 | 4 |

# Integers - Variants

1. Signed – both positive and negative values
2. Unsigned – when we know that the integer can only take positive values.
3. Range for signed = $(- 2^{n-1})$ to $(2^{n-1}-1)$ where n = number of bits the integer takes.
4. Range for unsigned = 0 to $(2^{n}-1)$ where n = number of bits the integer takes.

```
unsigned int  num_students ;
```

For n = 16

1. Signed range: -32768 to +32767
2. Unsigned range: 0 to 65535

# Floats and Doubles

1. **Floats occupy 4 bytes each**
2. **Range: -3.4e38 to +3.4e38.**
3. **Double:**
    a) **When floats are insufficient**
    b) **8 bytes**
    c) **Range: -1.7e308 to +1.7e308**
4. **Long double:**
    a) **Occupy 10 bytes**
    b) **Range: -1.7e4932 to +1.7e4932**

# ASCII Values

1. American Standard Code for Information Interchange, is a character encoding standard for electronic communication.
2. ASCII codes represent text in computers.
3. ASCII has just 128 code points, of which only 95 are printable characters, which severely limited its scope.
4. All modern computer systems instead use Unicode, which has millions of code points, but the first 128 of these are the same as the ASCII set.

# ASCII Values

1. Characters in C language also follow ASCII encoding.
2. What does this mean?
3. Characters from A-Z and a-z all have some ASCII code value, which is an integer number.
4. The code for A is 65
5. The code for B is 66 and so on till 90 for Z
6. The code for a is 97 and z is 122

# Chars – Character Variables

1. Keyword char is used for declaring character type variables

```c
#include <stdio.h>
int main(){
    char a = 'a'; //assigning character value to character
        variable
    printf("Character Value of a: %c\n", a);
    printf("Integer Value of a: %d\n", a);
    a++;
    printf("Value of a after increment is: %c\n", a);
    // c is assigned ASCII values which corresponds to the
    // character 'c'
    // a-->97 b-->98 c-->99
    char c;
    c = 99;  //assigning integer value to character variable
    printf("Value of c: %c", c);
    return 0;
}
```

# Chars – Character Variables


Output

```
/tmp/x35XuYsWBc.o
Character Value of a: a
Integer Value of a: 97
Value of a after increment is: b
Value of c: c
```

# ASCII Values - Example

```c
#include <stdio.h>
int main() {
    int i = 65, j ;
    for(i=65;i<=90;i++){
        printf("\ni = %d and i = %c",i,i);
        // printf("\ni = %c",i);
    }
    return 0;
}
```

```
/tmp/yv3pz3Cw2j.o
i = 65 and i = A
i = 66 and i = B
i = 67 and i = C
i = 68 and i = D
i = 69 and i = E
i = 70 and i = F
i = 71 and i = G
i = 72 and i = H
i = 73 and i = I
i = 74 and i = J
i = 75 and i = K
```

# Chars - Variants

```
char ch = 'A' ;
```

1. **What happens here?**
2. **The binary equivalent of the ASCII value of ‘A’ (i.e. binary of 65) gets stored in ch.**
3. **If 65 can be stored, logically we should be able to store -65 as well.**
4. **Where do we need signed chars?**

# Chars - Variants

**1. Signed – 1 byte**

a) **Range: -128 to +127**

**2. Unsigned – 1 byte**

a) **Range: 0 to 255**

**3. Why do we need this?**

# Summary

| Data Type | Range | Bytes | Format |
|---|---|---|---|
| signed char | -128 to + 127 | 1 | %c |
| unsigned char | 0 to 255 | 1 | %c |
| short signed int | -32768 to +32767 | 2 | %d |
| short unsigned int | 0 to 65535 | 2 | %u |
| signed int | -32768 to +32767 | 2 | %d |
| unsigned int | 0 to 65535 | 2 | %u |
| long signed int | -2147483648 to +2147483647 | 4 | %ld |
| long unsigned int | 0 to 4294967295 | 4 | %lu |
| float | -3.4e38 to +3.4e38 | 4 | %f |
| double | -1.7e308 to +1.7e308 | 8 | %lf |
| long double | -1.7e4932 to +1.7e4932 | 10 | %Lf |

Note: The sizes and ranges of int, short and long are compiler dependent. Sizes in this figure are for 16-bit compiler.