Basis and Practice in Programming

Chapter 4: Character strings & formatted I/O

Prof. Tamer ABUHMED
College of Computing and Informatics

성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

Lecture Objectives

- Explain Boolean data type
- Explain how to handle strings
- Explain how to define constants in C
- Explain data overflow problem
- Explain data type conversion (Casting)

The Boolean Data Type `_Bool`

- A `_Bool` variable is defined in the language to be large enough to store just the values 0 and 1.
- The precise amount of memory that is used is unspecified.
- `_Bool` variables are used in programs that need to indicate a Boolean condition.
- By convention, 0 is used to indicate a false value, and 1 indicates a true value. When assigning a value to a `_Bool` variable, a value of 0 is stored as 0 inside the variable, whereas any nonzero value is stored as 1.
- To make it easier to work with `_Bool` variables in your program, the standard header file `<stdbool.h>` defines the values `bool`, `true`, and `false`:

```
bool endOfData = false;
```

- The `_Bool` type has been added by C99.
- Some compilers (Borland C, Turbo C, Visual C) don't support it

Save more than the size: Integer overflow

- What happens if an integer tries to get a value too big for its type (out of range)?

```
#include <stdio.h>
int main(void) {
     int i = 2147483647;
     printf("%i %i %i\n", i, i+1, i+2);
     return 0;
}
```

Program output:

```
2147483647 -2147483648 -2147483647
```

Explanation:
On this computer, int is stored on 32 bits: the first bit represents the sign, the rest of 31 bits represent the value.
Biggest positive int value here: 231-1 = 2147483647

Save more than the size
Floating point round-off error

```
#include <stdio.h>
int main(void)
{
    float a,b;
    b = 2.0e20 + 1.0;
    a = b - 2.0e20;
    printf("%f \n", a);
    return 0;
}
```

Program output:

```
4008175468544.000000
```

Explanation: the computer doesn't keep track of enough decimal places ! The number 2.0e20 is 2 followed by 20 zeros and by adding 1
you are trying to change the 21st digit. To do this correctly, the program would need to be able to store a 21-digit number. A float number is typically just six or seven digits scaled to bigger or smaller numbers with an exponent.

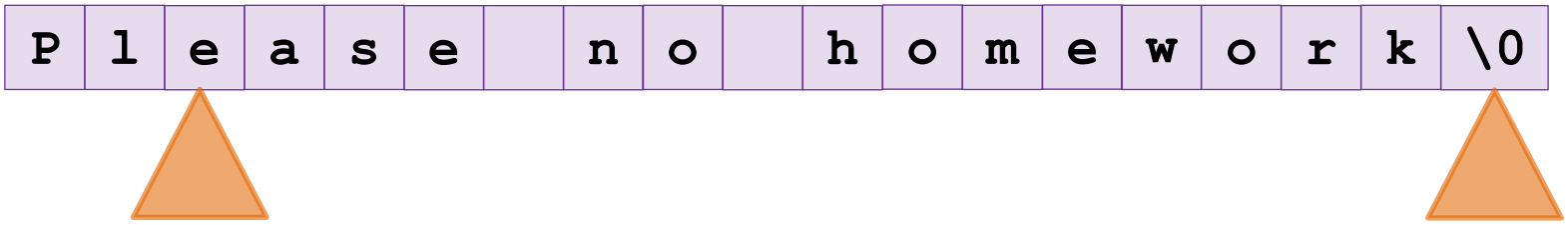
String

- Character
  - Occupies one byte in memory

```
char toll;
toll = 'A';
```

- String
  - It is a sequence (i.e., array) of characters
  - Computer terminates the string in memory by a NULL character '\0'

```
char comment[30] = "Please no homework";
```



Handling Strings

- A short example

```
# include <stdio.h>
# include <string.h>      // for strlen() prototype

void main()
{
    float weight;
    int size, letters;
    char name[40];          // name is an array of 40 chars

    printf("Hi! What's your first name?\n");
    scanf("%s", name);
    printf("%s, what's your weight in kg?\n", name);
    scanf("%f", &weight);
    size = sizeof(name);
    letters = strlen(name);

    printf("Hello %s,\n", name);
    printf("your name has %d letters,\n", letters);
    printf("and your weight is %.2f kg.\n", weight);

}
```

Output

```
Hi! What's your first name?
Songbae
Songbae, what's your weight in kg?
75
Hello Songbae,
your name has 7 letters,
and your weight is 75.00 kg.
```

Handling Strings – contd.

- Another example

```
# include <stdio.h>
# include <string.h>      /* provides strlen() prototype */
# define  PRAISE "What a super marvelous name!"

void main(void)
{
    char name[40];

    printf("What's your name?\n");
    scanf("%s", name);
    printf("Hello, %s. %s\n", name, PRAISE);
    printf("Your name of %d letters occupies %d memory cells.\n",
         strlen(name), sizeof name);

}
```

Output

What's your name?
YoungJae
Hello YoungJae. What a super marvelous name!
Your name of 8 letters occupies 40 memory cells

Constants

- We define a constant using the "# define" directive
  - # define name value
  - Examples
    - # define PI 3.14159 // no semicolon
      - The compiler substitutes 3.14159 whenever PI is used in your code
    - # define CH 'A'
      - The compiler substitutes 'A' whenever CH is used in your code
    - # define name "Huey Freeman"
      - The compiler substitutes "Huey Freeman" whenever name is used in your code
    - # define age 19
      - The compiler substitutes 19 whenever age is used in your code
    - # define PRINT(x) printf("x = %d\n", x)
      - The compiler substitutes printf("x = %d\n", x) whenever PRINT(x) is used in your code

Constants – contd.

- Again! An example ☺

```
# include <stdio.h>
# define PI 3.14159    // constant
# define PRINT(x, y) printf("circumference = %1.2f, area = %1.2f\n", x, y)

void main()
{
    float area, circum, radius;      // variable declaration

    printf("What is the radius of your pizza?\n");
    scanf("%f", &radius);
    area = PI * radius * radius;     // calculate area of the pizza
    circum = 2.0 * PI *radius;       // calculate circum of the pizza
    printf("Your basic pizza parameters are as follows:\n");

    PRINT(circum, area);

}
```

Output

What's is the radius of your pizza?
0.5
Your basic pizza parameters are as follows:
circumference = 3.14, area = 0.79

Constants – contd.

- Again! An example ☺

```
# include <stdio.h>
//# define PI 3.14159     // constant
# define PRINT(x, y) printf("circumference = %1.2f, area = %1.2f\n", x, y)

void main()
{    const float PI = 3.14159;
     float area, circum, radius;       // variable declaration

     printf("What is the radius of your pizza?\n");
     scanf("%f", &radius);
     area = PI * radius * radius;      // calculate area of the pizza
     circum = 2.0 * PI *radius;        // calculate circum of the pizza
     printf("Your basic pizza parameters are as follows:\n");

     PRINT(circum, area);

}
```

Output

What's is the radius of your pizza?
0.5
Your basic pizza parameters are as follows:
circumference = 3.14, area = 0.79

Constants – contd.

- Conversion specification modifiers for printf()

Conversion specifiers	
Conversion specification	**Output**
%d	Decimal integer
%c	Single character
%f	Floating-point number
%p	A pointer (memory address)
%s	Character string
%u	Unsigned decimal integer
%o	Octal integer
%x	Hexadecimal integer

A Comment on scanf()

- When the argument is STRING (array of characters)
  - There will be no “&” preceding the variable name
    - This is because the array’s name includes the address! (we will study it later)

```
...
char name[40];

printf("Please enter your first name.\n");
scanf("%s", name);
...
```

- When the argument is a single variable (not an array!)
  - There must be the “&” before the variable name
    - This is because variable’s name does not include the address

```
...
int age;

printf("Please enter your age.\n");
scanf("%d", &age);
...
```

Let's check what is wrong!

- It's time to find few errors
  - **There are 12 errors and you need to find the error line and error details**

```
1   define B Huey
2   define X 10
3   main(int)
4   {
5    int age;
6    char name;
7    printf("My name is %c, please enter your first name.", B);
8    scanf("%s", name);
9    printf("All right, %c, what's your age?\n", name);
10   scanf("%f", age);
11   xp = age + X;   // your age after 10 years
12   printf("after %d years, you will be %d years old.\n", xp);
13  return 0;
```

Integer and Floating-Point Conversions

- Assign an integer value to a floating variable: does not cause any change in the value of the number; the value is simply converted by the system and stored in the floating
- Assign a floating-point value to an integer variable: the decimal portion of the number gets truncated.
- Integer arithmetic (division):
  - int divided to int => result is integer division
  - int divided to float or float divided to int => result is real division (floating-point)

Integer and Floating-Point Conversions

```c
// Basic conversions in C
#include <stdio.h>
int main (void)
{
  float f1 = 123.125, f2;
  int i1, i2 = -150;
  char c = 'a';
  i1 = f1; // floating to integer conversion
  printf ("%f assigned to an int produces %i\n", f1, i1);
  f1 = i2; // integer to floating conversion
  printf ("%i assigned to a float produces %f\n", i2, f1);
  f1 = i2 / 100; // integer divided by integer
  printf ("%i divided by 100 produces %f\n", i2, f1);
  f2 = i2 / 100.0; // integer divided by a float
  printf ("%i divided by 100.0 produces %f\n", i2, f2);
  f2 = (float) i2 / 100; // type cast operator
  printf ("(float) %i divided by 100 produces %f\n", i2, f2);
  return 0;
}
```

The Type Cast Operator

- `f2 = (float) i2 / 100; // type cast operator`
- The type cast operator has the effect of converting the value of the variable i2 to type float for purposes of evaluation of the expression.
- This operator does NOT permanently affect the value of the variable i2;
- **The type cast operator has a higher precedence than all the arithmetic operators except the unary minus and unary plus**.

- Examples of the use of the type cast operator:
- `(int) 29.55 + (int) 21.99` results in `29 + 21`
- `(float) 6 / (float) 4` results in `1.5`
- `(float) 6 / 4` results in `1.5`

Lecture Keywords

- Keywords
  - _Bool data type
  - String
  - Array (overview)
  - scanf()
  - # define
  - const
  - Data type conversion

Lecture Summary

- Explain Boolean data type
- Explain how to handle strings
- Explain how to define constants in C
- Explain data overflow problem
- Explain data type conversion (Casting)