CSI 333 – Lecture 4
Introduction to C: Part III
/* Prints the no. of bytes for various data types. */
#include <stdio.h>

int main(void) {
    printf("Size of char = %d\n", sizeof(char));
    printf("Size of int = %d\n", sizeof(int));

    printf("Size of float = %d\n", sizeof(float));
    printf("Size of double = %d\n", sizeof(double));

    printf("Size of short = %d\n", sizeof(short));
    printf("Size of long = %d\n", sizeof(long));
    return 0;
}

Output of program (on itsunix):

Size of char = 1
Size of int = 4
Size of float = 4
Size of double = 8
Size of short = 2
Size of long = 4
Strings as Character Arrays

**Strings:**

- Can be implemented as arrays of type `char`. (Pointer version to be discussed later.)
- Each string **must end** with the ‘\0’ character (whose ASCII code is 0).

**Example:**

```c
char greet[6] = "Hello";
```

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>H</td>
<td>e</td>
<td>l</td>
<td>l</td>
<td>o</td>
</tr>
<tr>
<td>5</td>
<td>\0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

`greet`
Notes on the Example:

- The size of the greet array must be 6 to allow for the ’\0’ character.

- The elements of the array can be accessed separately. For example,

  ```c
  printf("%c %c\n", greet[0], greet[1]);
  ```

  produces the following output.

  H e

- Note the use of format "%c" to print one character.
Instead of

```c
char greet[6] = "Hello";
```

one can also have

```c
char greet[] = "Hello";
```

The system automatically sets the size of the array to be 6 (to allow for ‘\0’).

**Reading a string from stdin:**

- Use `scanf` with "%s" format.
- Using "%s" format skips leading white space characters (if any). The string ends when a white space character (or EOF) is seen.
- Function `scanf` automatically inserts the ‘\0’ character at the end of the string.
Example: Consider the following C program segment:

```c
#define SIZE 6
char word[SIZE];
scanf("%s", word); /* Note the usage. */
```

Suppose the input typed by the user is:

This is ICSI 333.

After the call to `scanf`, the variable `word` stores the string "This" as shown below.

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>h</td>
<td>i</td>
<td>s</td>
<td>\0</td>
<td></td>
</tr>
</tbody>
</table>
```

word
Strings using Character Arrays (continued)

**Printing a string to stdout:**

- Use `printf` with "%s" format.
- Prints all the characters in the string up to the first ’\0’ character.

**Example:** The following C program segment

```c
char new_greet[] = "Hello World";
printf("%s\n", new_greet); /* Note the usage. */
```

produces the following output:

```
Hello World
```
The C String Library (<string.h>)

Ref: Chapter 8 of Deitel & Deitel.

**Function strlen:**

- Returns the length of the string parameter passed to the function.
- Length does **not** include the ‘\0’ character.

**Example:**

```c
#include <stdio.h>
#include <string.h>
char new_greet[] = "Hello World";
printf("Length = %d\n", strlen(new_greet));
```

**Output:**

```
Length = 11
```
**Function** `strcmp`:
- Compares two strings.
- Returns 0 if the strings are identical; otherwise, returns a non-zero value.

**Code segment 1:**

```c
char greet_one[] = "Hello World";
char greet_two[] = "Hello WORLD";

if (strcmp(greet_one, greet_two) == 0)
    printf("Strings are identical.\n");
else
    printf("Strings are not identical.\n");
```

**Output:**

Strings are not identical.
Code segment 2:

```c
char greet_one[] = "Hello World";

if (strcmp(greet_one, "Hello World") == 0)
    printf("Strings are identical.\n");
else
    printf("Strings are not identical.\n");
```

Output:

Strings are identical.

Important Note:

Direct comparison of two strings using "==" won’t work.
For example, the following code segment does not compare the two strings defined earlier:

```c
if (greet_one == greet_two) {
    .
    .
}
```

**Function `strcpy`**:

- Copies one string into another.
- Usage: `strcpy(dest_str, source_str)`
- Also copies the ‘\0’ character.
- The destination string must have enough room for the source string and the ‘\0’ character.
The C String Library (continued)

**Code segment:**

```c
#define LEN 12
char greet_one[] = "Hello World";
char new_str[LEN];
strcpy(new_str, greet_one); printf("%s\n", new_str);
strcpy(new_str, "Good Bye"); printf("%s\n", new_str);
```

**Output:**

Hello World
Good Bye

**Important Note:**

You **cannot** copy a string by a statement of the form

```c
new_str = greet_one;
```
Structures (or Records)

- Similar to classes in Java (or C++) except that there are no member functions (methods).

**Example:**

```c
#define MAX 25
struct employee {
    char name[MAX]; int age; float salary;
};
struct employee x;
```

**Remarks:**

- The `struct` definition above actually defines a type (namely, `struct employee`) and not a variable.
- The variable `x` defined on the next line is of type `struct employee`.
- Use the “dot” notation to access the components (which are data members) of a `struct`. 
Example:

```c
struct employee x, y;

strcpy(x.name, "Norma Baker");
x.age = 34;
scanf("%f", &x.salary);

strcpy(y.name, "Robert Smith");
y.age = x.age + 5;
y.salary = 0.90 * x.salary;
```
**Keyword** typedef:

- A mechanism for creating synonyms for previously defined types.

**Example:**

```c
typedef struct employee EmpRec;
```

After the above type definition, EmpRec can be used wherever struct employee is needed. For example,

```c
EmpRec x, y;
```

declares variables `x` and `y` to be of type `struct employee`. 
Arrays of Structures

With the declaration

```c
#define SIZE 10
int values[SIZE];
```

each element of `values` stores an integer; that is, the type of `values[i]` is `int`.

One can also have arrays where each element is a `struct`.

**Example:**

```c
#define MAX 25
#define SIZE 100
struct employee { 
    char name[MAX]; int age; float salary;
};
struct employee toy_dept[SIZE];
```
With the above declaration, any element of the array `toy_dept` is of type `struct employee`.

The fields of the struct stored in any element of the array can be accessed as follows:

```c
    toy_dept[i].salary = 70000.0;
    strcpy(toy_dept[i].name, "Adam Smith");
    scanf ("%d", &toy_dept[i].age);
```

**Program Example:** See Handout 4.1.
Structure of a C Program

**Note:** This version takes into account `struct` definitions and `typedef` statements.

1. `#include` statements for standard libraries (e.g. `stdio.h`, `stdlib.h`).
2. Definitions of symbolic constants (using `#define`).
3. Definitions of structures and `typedef` statements.
4. Function prototypes.
5. Code for each of the functions.