CSE240 – Introduction to Programming Languages (online)

Lecture 06:
Programming with C | Primitive Data Types, Arrays and Strings, and Constants

Javier Gonzalez-Sanchez

javiergs@asu.edu
javiergs.engineering.asu.edu
Office Hours: By appointment

jgs

Primitive Data Types

Primitive Data Types and Modifiers

jgs

C defines five basic types

- **`int`** - integer, a whole number.
- **`float`** - floating point value i.e., a number with a fractional part.
- **`double`** - a double-precision floating point value.
- **`char`** - a single character.
- **`void`** - valueless special purpose type which we will examine closely in later sections.

C++ will add

- **`bool`** – boolean values

Primitive types can be modified using one or more of these modifiers

- **`signed`,**
- **`unsigned`,**
- **`short`,**
- **`long`**

Basic Data Types and Ranges in C

jgs

Type	Guaranteed minimum range	≥ bits
char	-127 to 127 or 0 to 255	8
signed char	-127 to 127	8
unsigned char	0 to 255	8
int	-32,768 to 32,767	16
signed int	same as int	16
unsigned int	0 to 65,535	16
short int	-32,768 to 32,767	16
signed short int	same as short int	16
unsigned short int	unsigned int	16
long int	±2,147,483,647	32
signed long int	same as long int	32
unsigned long int	0 to 4,294,967,295	32
float	6 decimal digits of precision	32
double	10 decimal digits of precision	64
long double	10 decimal digits of precision	64
bool (C++ only)	true/false	1 (8)

Basic Data Types and Ranges in C


```
unsigned int x = 65000;

int x = 32767;
```

	Guaranteed minimum range	≥ bits
	-127 to 127 or 0 to 255	8
	-127 to 127	8
	0 to 255	8
int	-32,768 to 32,767	16
signed int	same as int	16
unsigned int	0 to 65,535	16
short int	-32,768 to 32,767	16
signed short int	same as short int	16
unsigned short int	unsigned int	16
long int	±2,147,483,647	32
signed long int	same as long int	32
unsigned long int	0 to 4,294,967,295	32
float	6 decimal digits of precision	32
double	10 decimal digits of precision	64
long double	10 decimal digits of precision	64
bool (C++ only)	true/false	1 (8)

Basic Data Types and Ranges in C

jgs

Type		≥ bits
char		8
signed char		8
unsigned char		8
int		16
signed int		16
unsigned int		16
short int		16
signed short int		16
unsigned short int		16
long int		32
signed long int		32
unsigned long int	0 to 4,294,967,295	32
float	6 dec	32
double	10 de	64
long double	10 de	64
bool (C++ only)	true/false	1 (8)

```c
#include <stdio.h>

int main() {
  char age = 90;
  if (age)
    printf("%d \n", age);
  else
    printf("zero is false");
  age = 0;
  if (age)
    printf("%d \n", age);
  else
    printf("zero is false");
}
```

```
90
zero is false
```

Arrays

Arrays

Homogeneous collection of data elements (all have the same type). Individual elements are accessed by an **index.**

```
int a[3], i, j, k; // a is not initialized
int ma[2][3] = {{4, 2, 3}, {7, 8, 9}};
int b[4] = {2, 3, 9, 4}; // b is initialized
a[0] = 2; // index starts from 0
a[1] = 3;
a[2] = 9;
i = sizeof a; // # of bytes used by a is 12
j = sizeof b; // # of bytes used by b is 16
k = sizeof ma;  // # of bytes used by ma is 24
```

A string is an array of characters.

There is not a class String (like in Java) or a primitive data type for strings

In other words, an array of characters can be seen as:

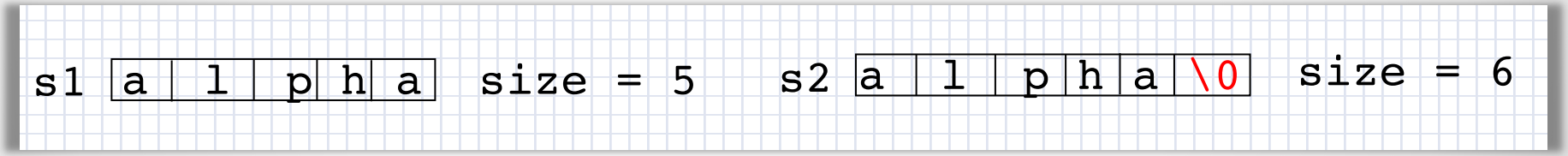
- An array of characters
- An string

Array and String

- There are two ways of **initializing** an array of characters in declaration:

```
char s1[] ={'a', 'l', 'p', 'h', 'a'}; // as an array of char
char s2[] ="alpha"; // as a string
```

- These two initializations have different results in memory:



- where **'\0'** is the null terminator (null character), marking the end of a string. It is automatically added to the end of a *string*.

Array and String

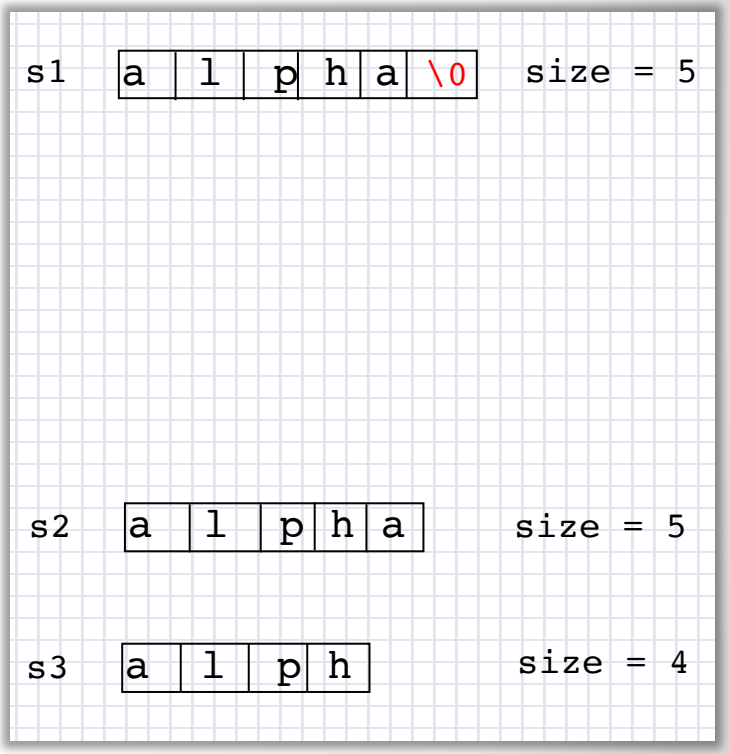
- To have the same result, we could do:

```
// char s1[] = "alpha";
char s1[]={'a', 'l', 'p', 'h', 'a', '\0'};
```

- If the size of the array is specified and there is not enough space, the null character will not be attached to the string.

```
char s2[5] = "alpha";

char s3[4] = "alpha";
```



Arrays and Strings

```
#include <stdio.h>

main() {

  char s1[ ] = "hello";

  printf("%s \n", s1);

  for (int i = 0; i < 5; i++)
    printf("%c", s1[ i ]);
  printf("\n");

  return 0;
}
```

```
hello
hello
```

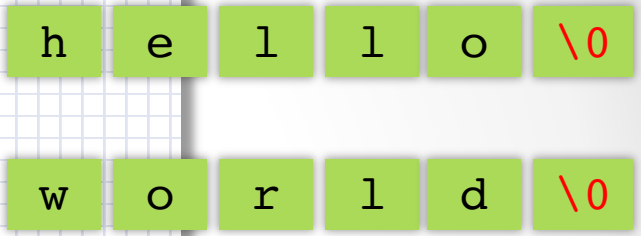
Array and Strings

```
#include <stdio.h>
#include <string.h>

main() {
  int i;
  char s1[ ] = "hello", s2[] = "world";

  for (i = 0; i < 5; i++)
    printf("%c", s1[ i ]);
  printf("\n");
  printf("s1 = %s, size = %d\n", s1, sizeof s1);

  for (i = 0; i < 5; i++)
    printf("%c", s2[ i ]);
  printf("\n");
}
```




```
hello
s1 = hello, size = 6
world
```

Constants

Constants

C allows two ways to define a constant:

- **const** It is equivalent to **final** in Java. This way is slower since it have to read memory.

```
const int i = 5;
i = i + 2;  // this line will cause a compilation error
```

- **#define** It substitute values for constant definitions in compilation time. Provide a faster execution.

```
#define PI 3.14

int x = PI * 5;
```

CSE240 – Introduction to Programming Languages (online)

Javier Gonzalez-Sanchez
javiergs@asu.edu
Fall 2017