CS 262 Lecture 2: Fundamentals

GEORGE MASON UNIVERSITY

Overview of Lecture 2

Variables and Constants

- Assignment operators
- Relational operators
- Logical operators

Data Types

- Primitive types (native to C)
- Different ways of representing data

Assignment Operators

Standard Libraries

- Libraries built into C
- Contains a lot of basic functionality

Printing Formatted Data

- Printing variables of different types
- Aligning printed data

Reminders

Notices

The first questions for the practice midterm are posted in Canvas

Anonymous instructor feedback form is posted in Canvas

Identifiers (Variable Names)

Rules for identifiers

Must start with a letter or an underscore _

note – C is case-sensitive

Identifiers can only contain letters, numbers, or underscores

Identifiers cannot be keywords (`if, for, while`)

Good Practices

Use descriptive variable names

Be consistent with convention (camelCase vs underscores)

These examples will work

```
int underscores_work = 1
int thisIsCamelCase = 1
char descriptive_name = 'A'
```

These examples won't work

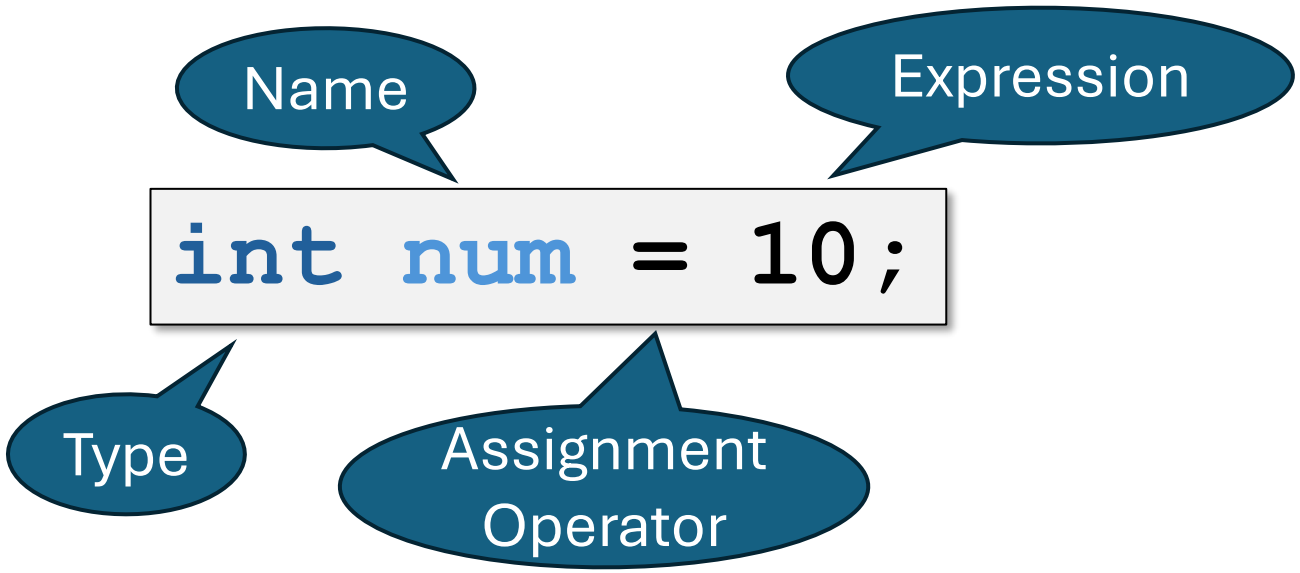
```
int 5thVariable = -1
int spaces dont work = -1
char h$%ello^ = 'X'
```

Declaring Variables

Rules for variables in C

All variables must be declared with a type

If we don't initialize it, it will contain a 'garbage' value. Believe it or not, this relates to C being fast and efficient



Constants

C has two styles of constants

Style 1 (most common)

```
// Example: #define IDENTIFIER value
#define PI 3.14
```

Style 2

```
// Example: const int identifier = value
const float pi = 3.14;
```

Constants

Constants can greatly increase readability

Bad programming practice

```
if (value > 378) { ...
// 378 seems random, hard to read
// we call these magic numbers. They are best avoided
```

Good programming practice

```
if (value > MAX_VALUE) { ...
// MAX_VALUE is a descriptive, self-explanatory name
```

Data Types

There are three primitive type categories in C

`void`

No values and No Operations
Used in Parameters and Returns

`int`

Stores integers
Can be **Signed** or **Unsigned**

`float`

Approximates Real Numbers (Fractional Values)
Large Ranges, but it does **Round**

Data Types

There are four different sizes of integer in C

Note: The actual sizes of each are system-dependent

`char` **Typical size: 1 byte**

`short int` **Typical size: 2 bytes**

`int` **Typical size: 4 bytes**

`long int` **Typical size: 8 bytes**

Data Types

There are three different sizes of floats in C

Note: The actual sizes of each are system-dependent

`float` **Typical size: 4 bytes**

`double` **Typical size: 8 bytes**

`long double` **Typical size: 16 bytes**

Doubles and long doubles offer more accurate approximations

Signed vs Unsigned Types - Range

Signed Integers

Can represent both positive and negative values

Unsigned Integers

Can only represent positive values

4 Bit Integer	Minimum Unsigned Value	Maximum Unsigned Value	Minimum Signed Value	Minimum Signed Value
bbbb (4 bits)	0000 (0)	1111 (15)	1000 (-8)	0111 (7)

What do you think happens if we try to go above the max values?

Signed vs Unsigned Types - Range

How do we check the size of a variable type?

```
sizeof(variable_type) // returns the size of the type passed in
```

Reference table for integer type ranges

4 Bit Integer	Minimum Unsigned Value	Maximum Unsigned Value	Minimum Signed Value	Minimum Signed Value
1 byte	0	15	-8	7
2 bytes	0	65,535	-32,768	32,767
4 bytes	0	4,294,967,295	-2,147,483,648	2,147,483,647
8 bytes	0	2^(64) - 1	-2^(63)	2^(63) - 1

More On Declaring Variables

Type	Format	Example
char	Single quotes	`char ch = 'q';`
int	Plain integer value	`int x = 123;`
unsigned int	Add U to the end	`int y = 12345U;`
long	Add L to the end	`long z = 12345L;`
float	Add F to the end	`float a = 12.34F;`
double	Plain decimal number	`double b = 12.34;`
Octal Value	Put 0 in the front	`int oct = 052;`
Hex Value	Put 0x in front	`int hex = 0x3F;`
Binary Value	Put b in the front	`int bin = 0b10110;`

Assignment Operators

Just like you remember them in other languages. These let you assign values to variables

Symbol	Basic assignment operator	Example
=	Basic assignment operator	`var1 = 3 // var1 is 3`
+=	Addition assignment	`var1 += 2 // var1 is now 5`
-=	Subtraction assignment	`var1 -= 1 // var1 is now 4`
*=	Multiplication Assignment	`var1 *= 10 // var1 is now 40`
/=	Division Assignment	`var1 /= 2 // var1 is now 20`
%=	Modulus Assignment	`var1 %= 3 // var1 is now 2`

Assignment Operators

Let's Practice

Pay close attention – These resemble what you will see on quizzes and exams

```
int value = 1;
value += 5;
value *= 3;


// value = _____
```

```
int val1 = 1;
int val2 = 3;
val2 += val1;
val1 *= val2;

// val2 = _____
// val1 = _____
```

Standard C Libraries

All functions in C need a Library to be Included

Library	Description
stdio.h	Basic Input and Output Functions / File Functions
stdlib.h	Memory Functions, Searching, Conversions
string.h	String Functions
math.h	Math Functions and Math Constants
time.h	Time and Date Functions

Standard C Libraries

stdio.h

Function	Description
printf(...)	Writes Formatted Data to the Output Device (screen)
scanf(...)	Reads Formatted Data from the Input Device (keyboard) into Variables
sscanf(...)	Reads Formatted Data from an Input String into Variables
fgets(...)	Reads User Input from stdin (keyboard) into a String

math.h

(Needs a special compile option to use. Add `-lm` during gcc)

Function	Description
sqrt(val)	Returns the Square Root of the Parameter
ceil(val)	Returns the Ceiling (Round val up to the next Integer)
floor(val)	Returns the Floor (Round val down to the next Integer)
pow(d1, d2)	Raises d1 to the power of d2 and returns the result

Standard C Libraries

stdlib.h

Function	Description
rand()	Returns a random value as an int. (Random in range of an int)
srand(val)	Initializes the Random Number Generator with input val.

string.h

Function	Description
strlen(str)	Returns the number of characters in the String
strcpy(s1, s2)	Copies string s2 into string s1. (strncpy is the Safer version)
strcat(s1, s2)	Concatenates s2 to the end of s1. (strncat is the Safer version)

Printing Formatted Data

The `printf` function will print out formatted data

```
int printf("hello\n")
```

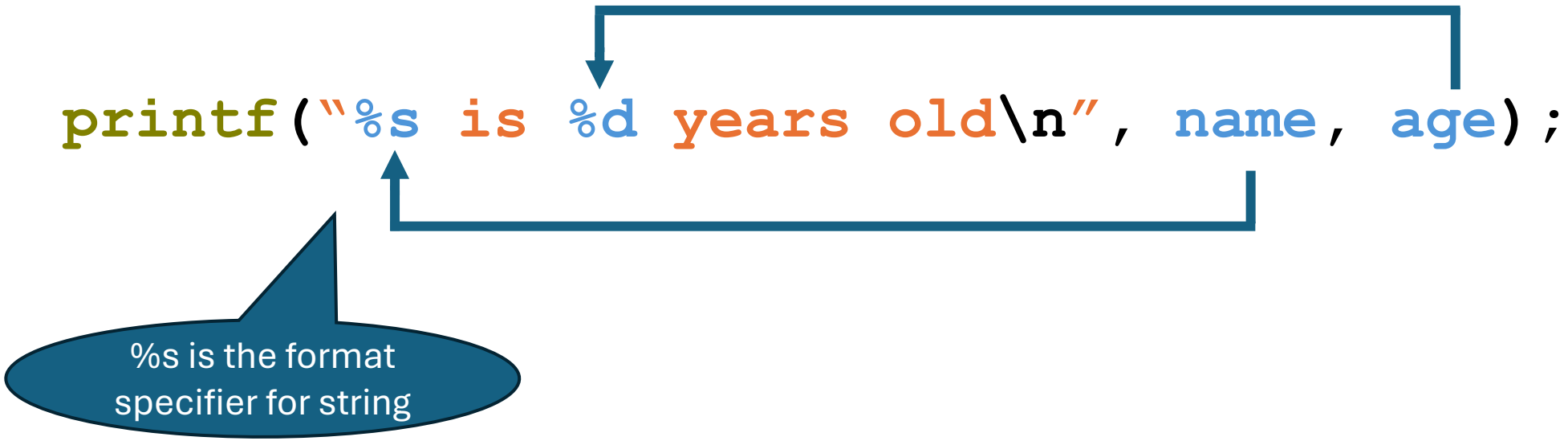
For values, we use something called Conversion characters – Placeholders to represent values that will be substituted into a string that tell the system how to format the value

```
printf("You are %d years old\n", age);
```

%d is the format specifier for int

Printing Formatted Data

We can have multiple conversion characters in a print statement



Printing Formatted Data

% Conversion Codes for `printf` and `scanf`

% Code	Description	Example
c	Character	`printf("Letter %c\n", 'A'); // "Letter A"`
d	Integer	`printf("She is %d\n", 20); // "She is 20"`
u	Unsigned Int	`printf("He is = %u\n", 10); // "He is 10"`
f	Floating-Point	`printf("Pi is %f\n", 3.14); // "Pi is 3.14"`
s	String	`printf("I am %s\n", "hungry"); // "I am hungry"`

And for Floating-Point Values

% Code	Description	Example
h	short int (short)	`printf("%hu\n", 3); // "3"`
l	long int (long)	`printf("%li\n", 42); // "42"`
l	double	`printf("%lf\n", 4.12); // "4.12"`

Printing Formatted Data – Escape Codes

Escape Sequences for Strings

Code	Description	Example
\b	Backspace	`printf("Hi!\b"); // "Hi"`
\t	Tab	`printf("Hi\t!"); // "Hi  !"`
\n	New Line	`printf("Hi\n!"); // "Hi` `!"`
\r	Carriage Return	`printf("Hi!\rMy"); // "My!"`
\"	Quotation	`printf("Hi \"Me\" !"); // "Hi "Me" !"`
\'	Apostrophe	`printf("It\'s Me !"); // "It's Me !"`
\0	Null Character	`printf("Hi!\0"); // "Hi!"`
\\	Backslash	`printf("Hi\\!"); // "Hi\!"`

Printing Formatted Data

`printf` lets us specify width and precision

This prints 2 numbers after the decimal

```
float pi = 3.1416f;
printf("Pi to 2 decimal places = %.2f", pi);
```

```
int num = 3;
printf("int with width 5: %5d", num);
```

This specifies a width of 5 (will always take up 5 spaces

```
Number with width 5:     3
```

Printing Formatted Data

printf also lets us add padding: the addition of extra data to achieve a specific size

Example: Adding 0 before the width will pad with that many 0s

```
float pi = 3.14;
printf("pi padded with some 0s: %020.2f", pi);
```

```
Pi padded with some 0s: 00000000000000003.14
```