

# Identifiers and Expressions

CSE 130: Introduction to C Programming  
Spring 2005

1

# Punctuation in C

- Statements are terminated with a ‘;’
- Groups of statements are enclosed by curly braces: ‘{’ and ‘}’
- Commas separate function arguments
- Whitespace is ignored

2

# Variables

- Remember that variables are named blocks of memory
- Variables have two properties:
  1. name — a unique identifier
  2. type — what sort of value is stored

3

# Identifiers

- Identifiers give unique names to various objects in a program
- An identifier may contain letters, digits, and the underscore character (‘\_’)
- An identifier must begin with a letter or \_
- Identifiers should be meaningful (and nouns)
- Style convention: the second and subsequent words in an identifier are capitalized

4

# Identifier Examples

- Good Identifiers
  - tax\_rate
  - taxRate
  - level4score
- Bad Identifiers
  - 1stName /\* starts with a digit \*/
  - %discount /\* contains invalid character \*/

5

# Keywords

- Some words may not be used as identifiers
- These words have special meaning in C
- C has 32 reserved words
- Ex. for, if, while, switch

6

## Reserved Words in C

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

7

## Data Types

- `int` — stores integer values (ex. 5)
- `float` — stores decimal values (ex. 3.14)
- `double` — stores larger decimal values than `float` (double the precision of a `float`)
- `char` — stores an integer representing a character (ex. 'A')
- Also `short`, `unsigned`, and `long`

8

## Type Size Hierarchy

`char`->`short`->`int`->`unsigned`->`long`->`float`->`double`

- Size (storage capacity) increases from left to right
- Ex. `char`: 1 byte, `int`: 4 bytes, `double`: 8 bytes
- Actual size varies based on machine architecture
- Use `sizeof()` to get actual size

9

## Type Conversion

- A value can always be stored in a variable of a larger (further right) type
  - Ex. an `int` can be stored in a `float`
- If you want to go the other way (to store a value in a smaller type), you must cast:

```
int x = (int)3.14159;
```
- Casting can lose (truncate) data

10

## Conversion Examples

```
double average = 100.0/8.0
average = 100.0/8
average = 100/8
int sumGrades = 100.0/8.0
sumGrades = (int)100.0/8.0
sumGrades = (int)(100.0/8.0)
double fiftyPercent = 50/100;
fiftyPercent = 50.0/100.0;
```

11

## Operators in C

- Some characters have special meanings
  - `+`, `-`, `*`, `/`, and `%` are arithmetic operators
  - `&&`, `||`, `!`, `==`, and `!=` are logical operators
- A character's meaning may depend on context
  - Ex. `printf("%d", a);` vs. `b = c % 4;`

12

## Integer Arithmetic in C

- Addition, subtraction, and multiplication work as you would expect
- Division (/) returns the whole part of the division (the quotient)
  - $12 / 3 = 4$      $15 / 2 = 7$
- Modulus (%) returns the remainder
  - $12 \% 3 = 0$      $15 \% 2 = 1$

13

## Shorthand Operators

- Some operators are shortcuts for others
  - Ex. +=, -=, \*=, /=, %=, ++, --
- $x += 5;$  is the same as  $x = x + 5;$
- $y++;$  is the same as  $y = y + 1;$

14

## Increment (++) and Decrement (--) Operators

- When used by themselves,  $y++;$  and  $++y;$  have identical results
- In an expression, they have different results
- The relative order of the operator matters:
  - $y++:$  use  $y$ 's value, then increment it.
  - $++y:$  increment  $y$ , then use new value.
- The same is true for decrement (--)

15

## Operator Precedence

- Precedence rules specify the order in which operators are evaluated
- Remember PMDAS:
  - Parentheses, Multiplication, Division, Addition, Subtraction
- Associativity determines left-right order

16

## Operator Precedence

Operator	Associativity
() ++ -- (postfix)	left to right
!	right to left
* / %	left to right
+ -	left to right
< <= > >=	left to right
== !=	left to right
&&	left to right
	left to right
?:	right to left
= += -= *= /= %=	right to left

17

## Precedence Examples

- $3 - 8 / 4$
- / has the highest precedence, so we compute  $8 / 4$  first, then subtract the result from 3
- Equivalent expression:  $3 - (8 / 4)$
- What is the value of  $3 * 4 + 18 / 2$ ?

18

## Precedence Examples

- $5 - 3 + 4 + 2 - 1 + 7$
- + and - have equal precedence, so this expression is evaluated left to right:
  - $(((((5 - 3) + 4) + 2) - 1) + 7)$
- Innermost parentheses are evaluated first

19

## Parentheses

- Parentheses can be used to force a different order of evaluation:
  - $12 - 5 * 2 = 2$
  - $(12 - 5) * 2 = 14$

20

## Expression Examples

- What do the following expressions evaluate to?
  - $1 + 2 * 3$
  - $(1 + 2) * 3$
  - $13 \% 5$
  - $23 \% 4 * 6$

21

## More Expression Examples

- $27.0 / 6.0$
- $27.0 / 6$
- $27 / 6$
- Given: `int x = 5;`
  - `int y = x++ * 6`
  - `int y = ++x * 6`

22

## Constants

- A constant is a value that cannot change
- Ex. numbers (42, 23, 3.14)
- Variables can be declared as constants using the “const” keyword:  
`const double pi = 3.1415926;`
- Strings (sequences of characters enclosed in double quotes) are also constants.

23

## User Input

- The `printf()` function is used to display output on the screen
- The `scanf()` function is used to read input from the keyboard
  - `scanf()` is also defined in `stdio.h`

24

## Using scanf()

- scanf() reads in data and stores it in one or more variables

- Ex.

```
int userAge;  
scanf(" %d", &userAge);
```

25

## scanf() Usage

- The first argument (the control string) contains a series of placeholders
- These are like the ones that printf() uses
- %d = int, %f = float, %c = char, etc.
- Spaces are used to separate placeholders and absorb whitespace
- “ %d” absorbs leading spaces and reads an integer value

26

## scanf() Usage, Pt. 2

- The remaining arguments to scanf() are a comma-separated list of variable names
- Input is stored in these variables
- Each variable name is preceded by &
  - &i means “the memory address associated with variable i”
  - We’ll talk more about this later on

27

## scanf() Examples

```
int a, b, c;  
char d;  
scanf(" %d %d", &a, &b);  
scanf(" %c %d", &d, &c);
```

28

## Next Time

- Relational Operators
- Conditional (selection) statements

29