RIT

SWEN-340

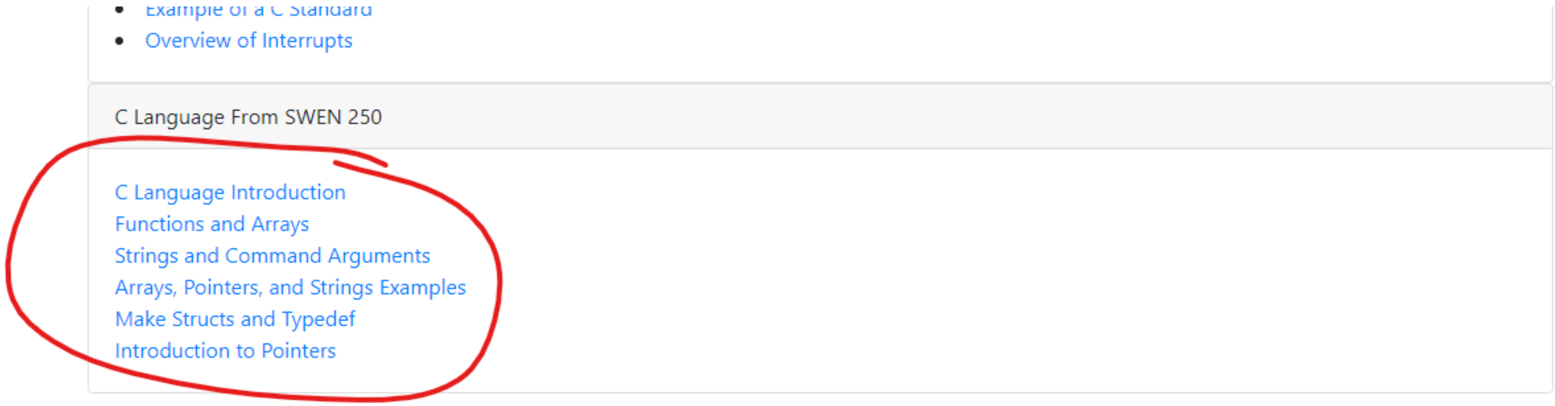
Software Design for Computing Systems

Mitesh Parikh

Jan 25, 2022

C-Review

- Review SWEN-250 material/slides
- Available on Resources page

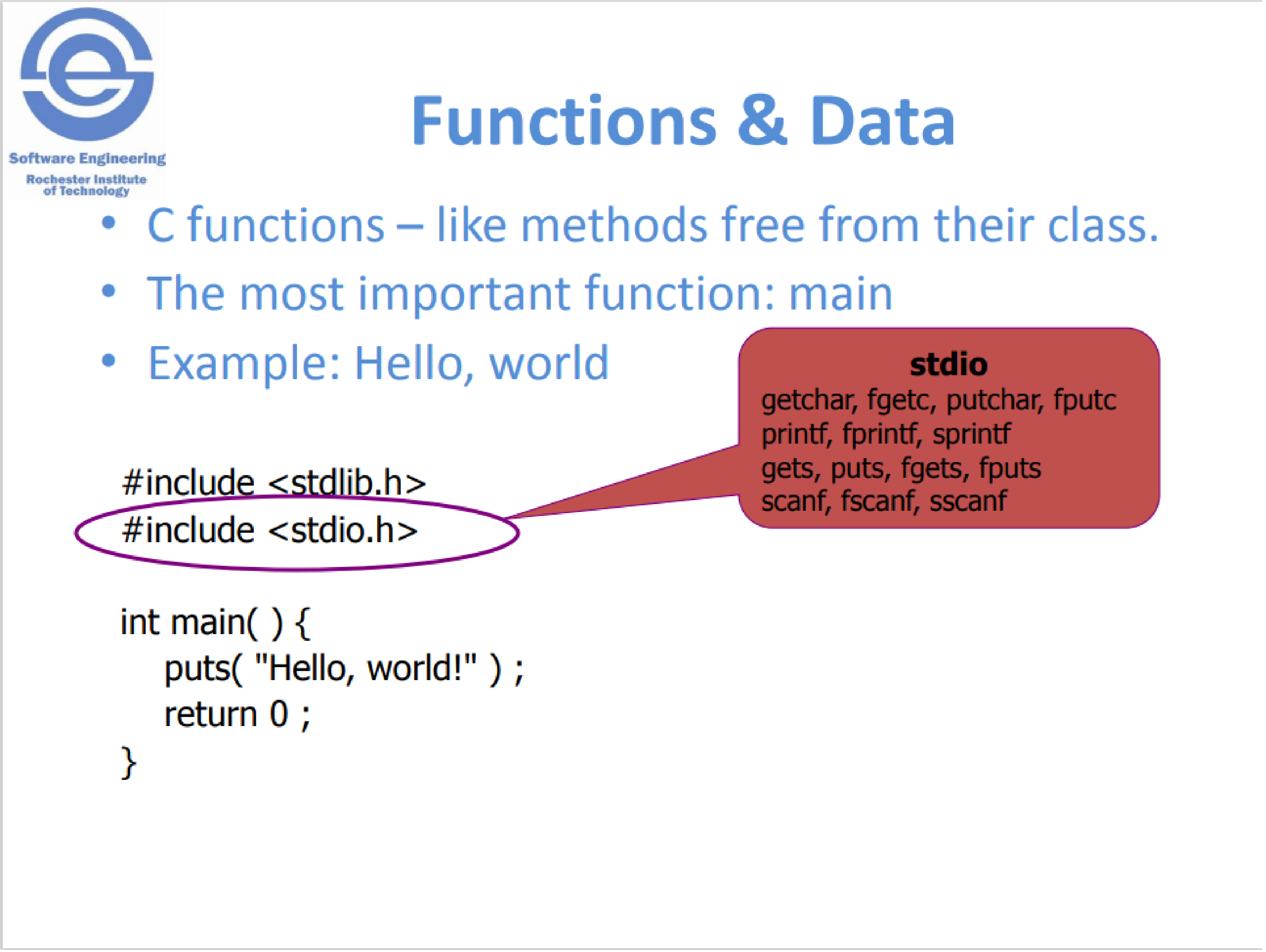


- Example of a C Standard
- Overview of Interrupts

C Language From SWEN 250

C Language Introduction
Functions and Arrays
Strings and Command Arguments
Arrays, Pointers, and Strings Examples
Make Structs and Typedef
Introduction to Pointers

Data Types



Software Engineering
Rochester Institute of Technology

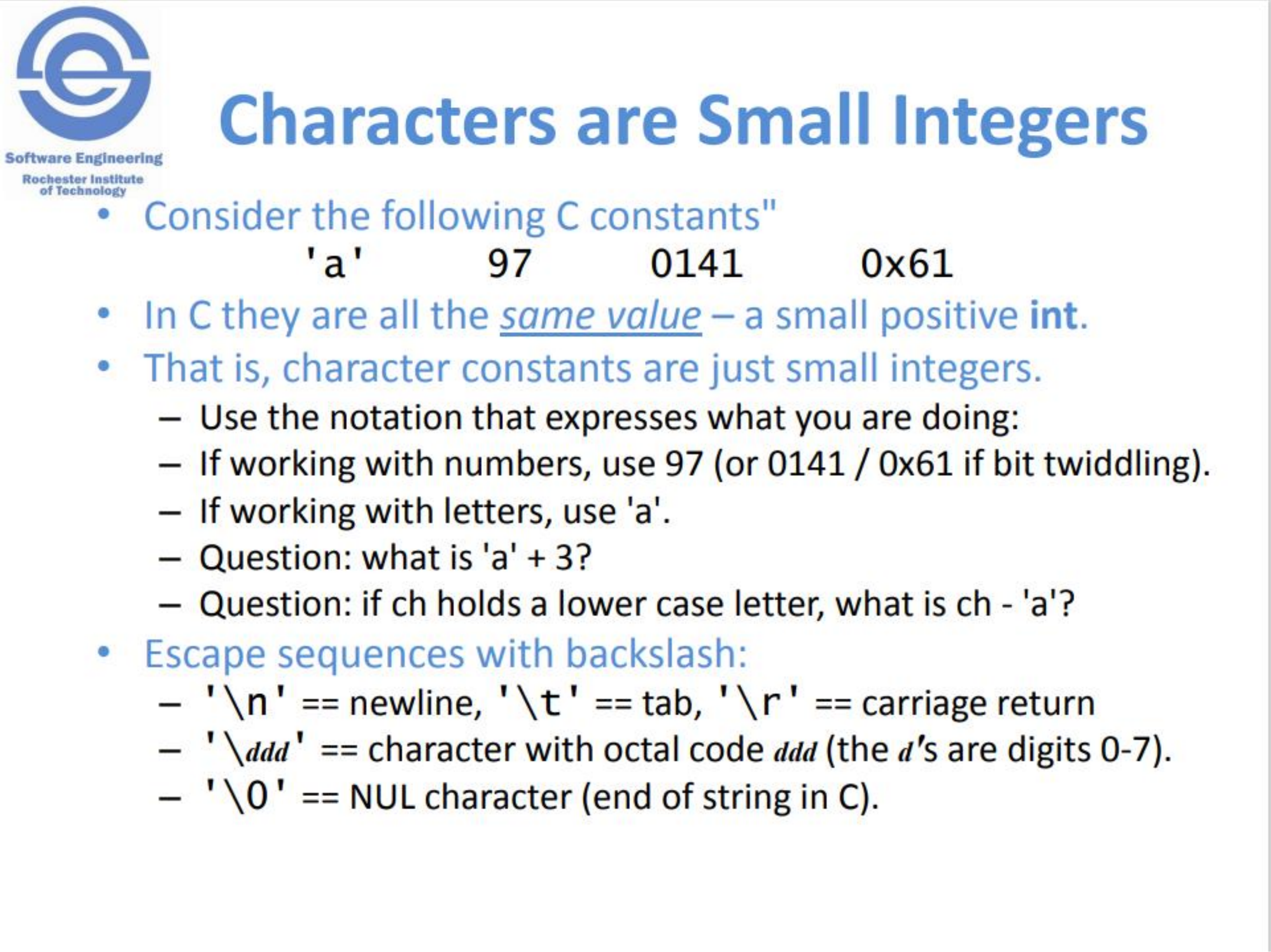
Functions & Data

- C functions – like methods free from their class.
- The most important function: main
- Example: Hello, world

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

int main( ) {
    puts( "Hello, world!" ) ;
    return 0 ;
}
```

stdio
getchar, fgetc, putchar, fputc
printf, fprintf, sprintf
gets, puts, fgets, fputs
scanf, fscanf, sscanf

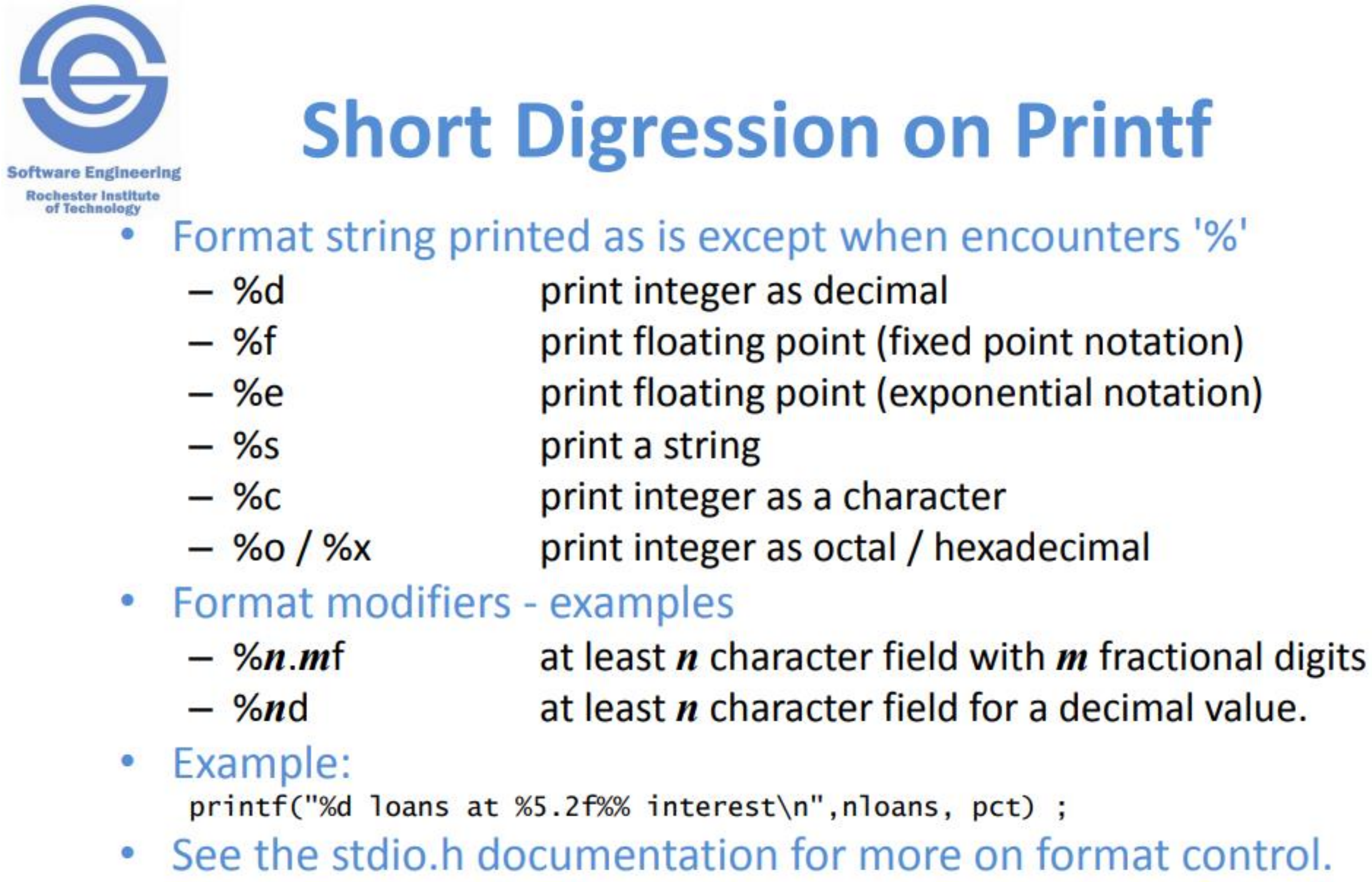


Characters are Small Integers

- Consider the following C constants"

  `'a'          97          0141          0x61`

- In C they are all the <u>*same value*</u> – a small positive **int**.
- That is, character constants are just small integers.
  - Use the notation that expresses what you are doing:
  - If working with numbers, use 97 (or 0141 / 0x61 if bit twiddling).
  - If working with letters, use 'a'.
  - Question: what is 'a' + 3?
  - Question: if ch holds a lower case letter, what is ch - 'a'?
- Escape sequences with backslash:
  - `'\n'` == newline, `'\t'` == tab, `'\r'` == carriage return
  - `'\`*ddd*`'` == character with octal code *ddd* (the *d*'s are digits 0-7).
  - `'\0'` == NUL character (end of string in C).



Short Digression on Printf

- Format string printed as is except when encounters '%'

– %d	print integer as decimal
– %f	print floating point (fixed point notation)
– %e	print floating point (exponential notation)
– %s	print a string
– %c	print integer as a character
– %o / %x	print integer as octal / hexadecimal

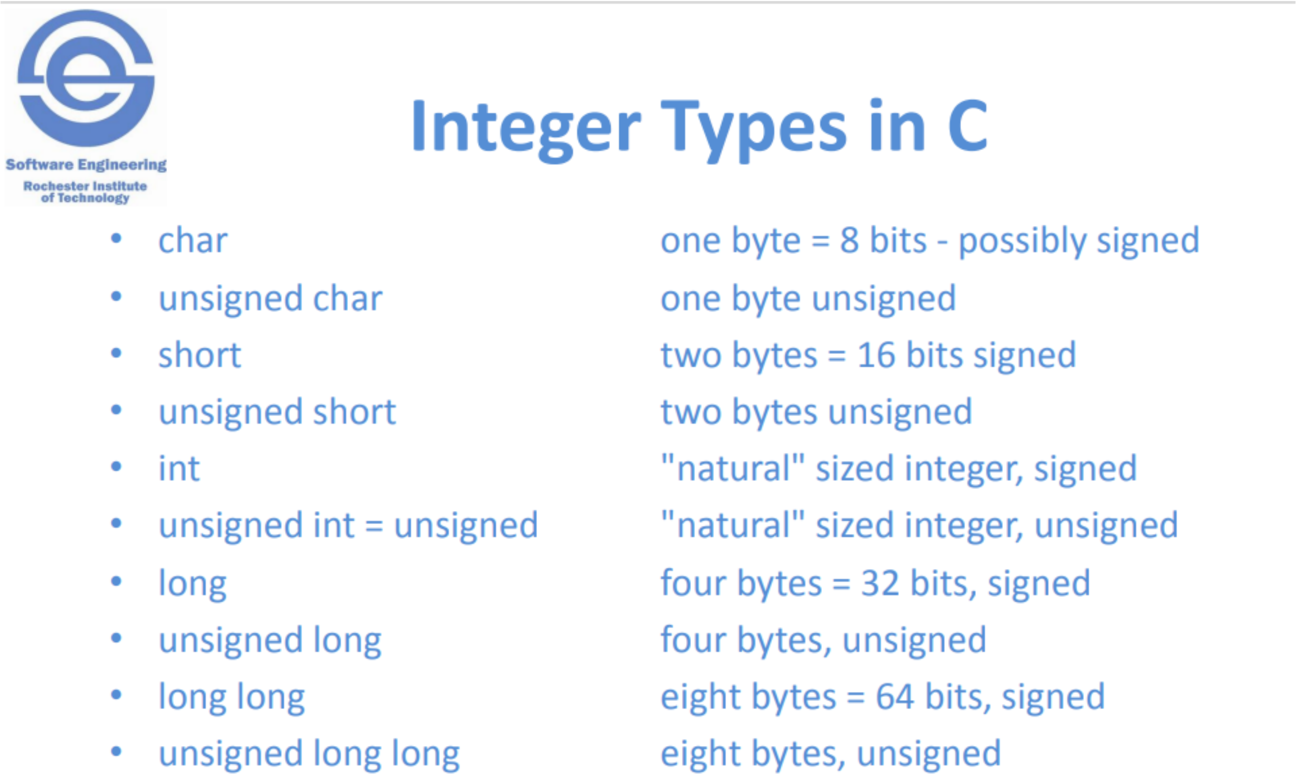
- Format modifiers - examples

– %***n***.***m***f	at least ***n*** character field with ***m*** fractional digits
– %***n***d	at least ***n*** character field for a decimal value.

- Example:

```
printf("%d loans at %5.2f%% interest\n",nloans, pct) ;
```

- See the stdio.h documentation for more on format control.

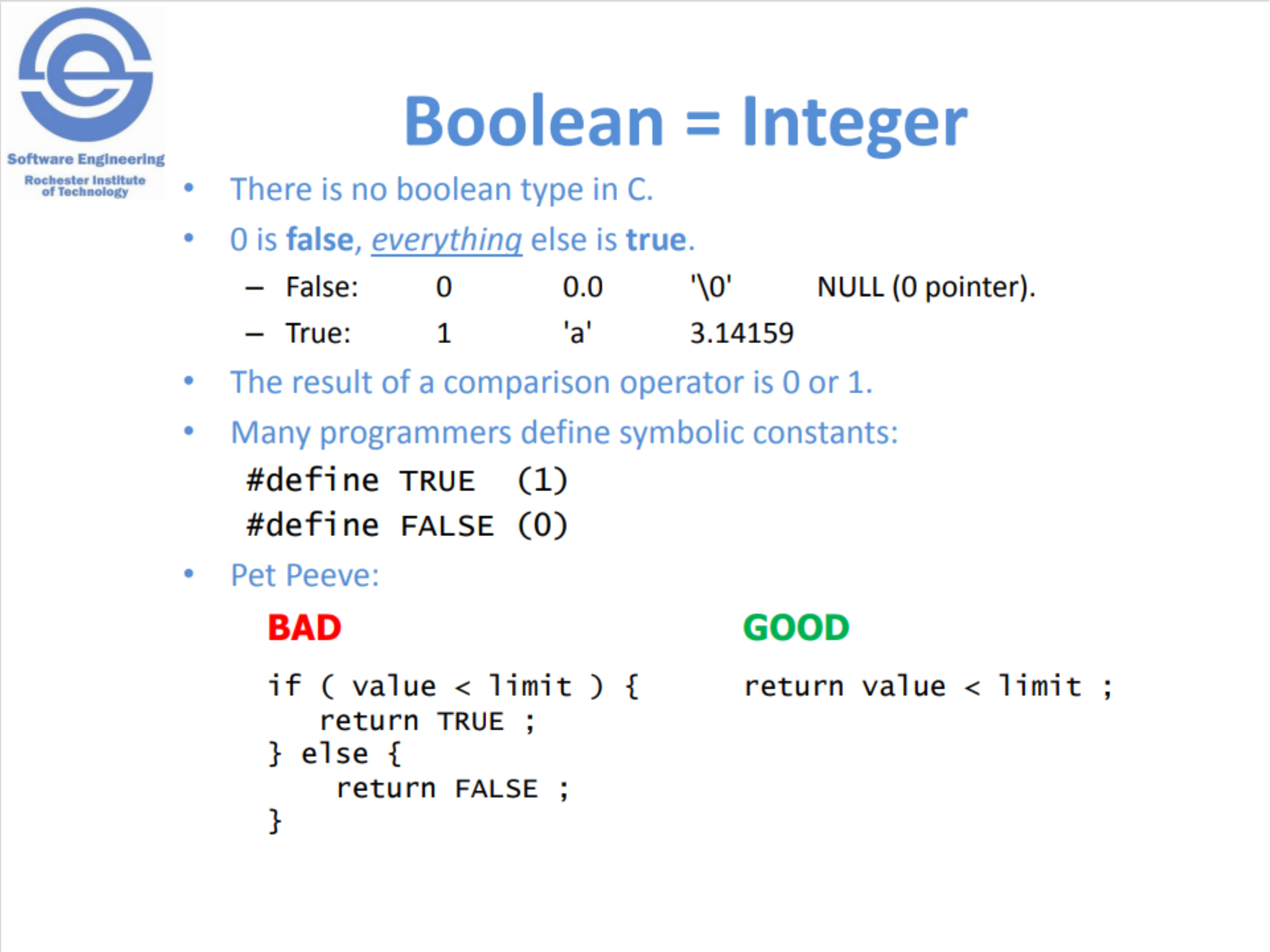


Software Engineering
Rochester Institute
of Technology

Integer Types in C

char	one byte = 8 bits - possibly signed
unsigned char	one byte unsigned
short	two bytes = 16 bits signed
unsigned short	two bytes unsigned
int	"natural" sized integer, signed
unsigned int = unsigned	"natural" sized integer, unsigned
long	four bytes = 32 bits, signed
unsigned long	four bytes, unsigned
long long	eight bytes = 64 bits, signed
unsigned long long	eight bytes, unsigned

- *“Natural” size is the width of integer that is processed most efficiently by a particular hardware*
- *32-bit Integers are "natural" for many 64-bit platform*



Boolean = Integer

- There is no boolean type in C.
- 0 is **false**, *everything* else is **true**.
  - False: 0 0.0 '\0' NULL (0 pointer).
  - True: 1 'a' 3.14159
- The result of a comparison operator is 0 or 1.
- Many programmers define symbolic constants:

```
#define TRUE  (1)
#define FALSE (0)
```

- Pet Peeve:

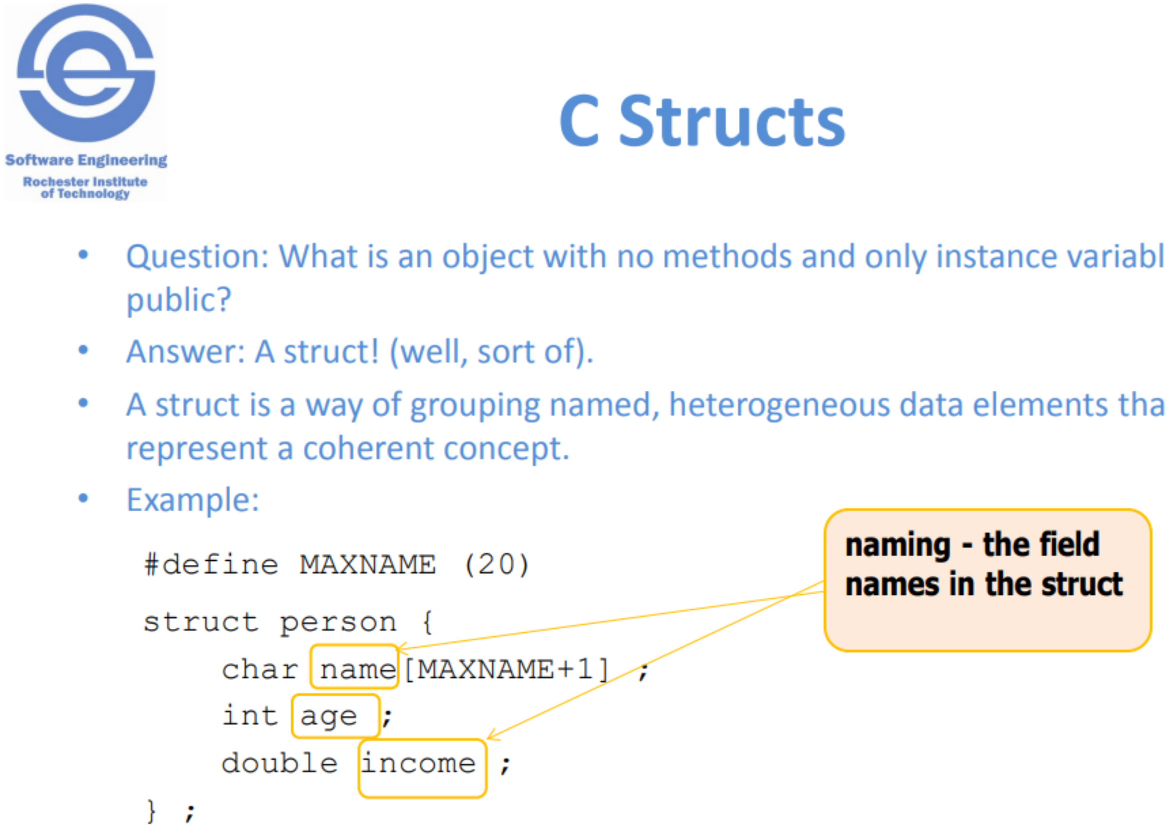
BAD

```
if ( value < limit ) {
   return TRUE ;
} else {
    return FALSE ;
}
```

GOOD

```
return value < limit ;
```

Structs & Pointers



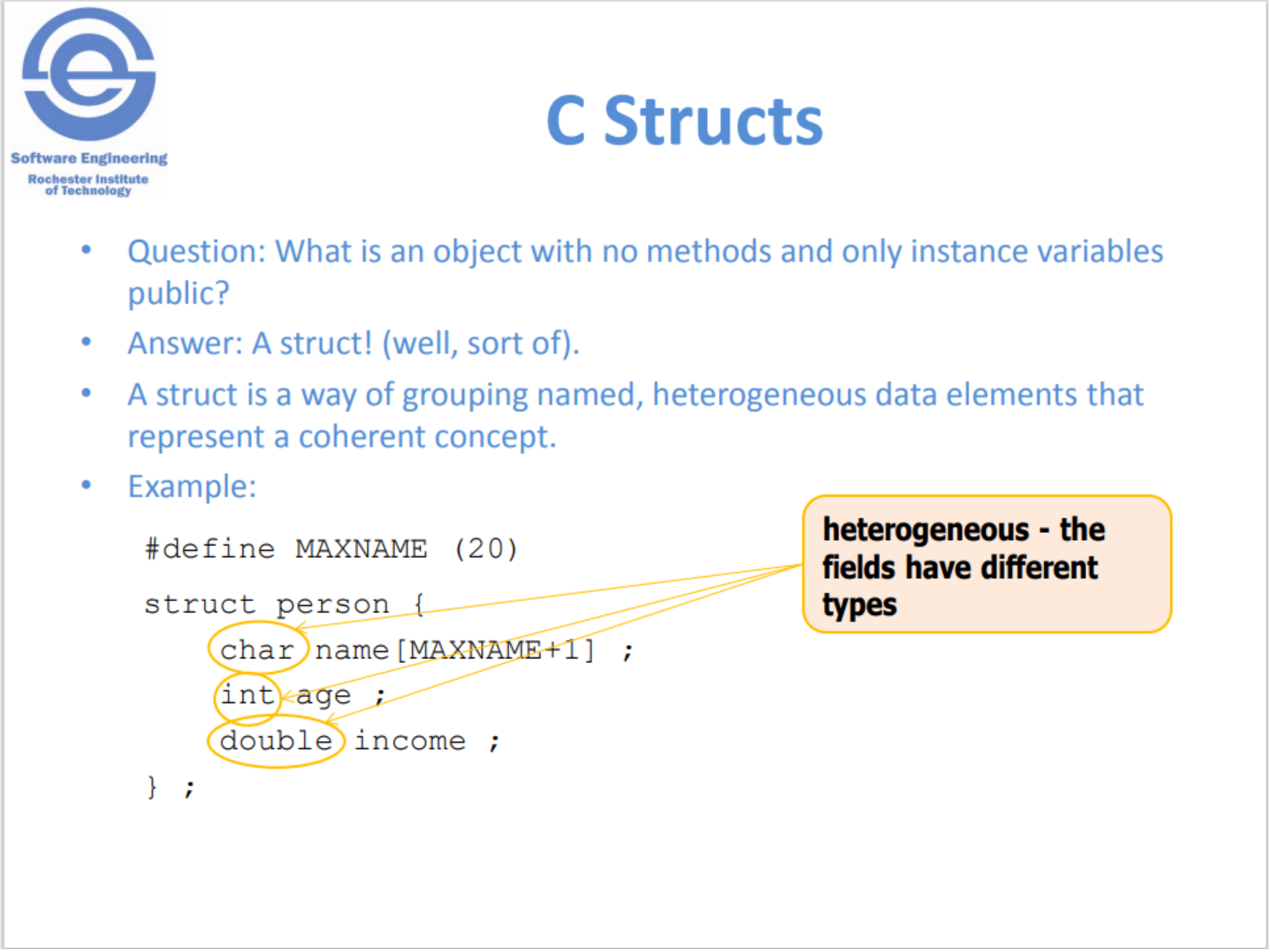
C Structs

- Question: What is an object with no methods and only instance variables public?
- Answer: A struct! (well, sort of).
- A struct is a way of grouping named, heterogeneous data elements that represent a coherent concept.
- Example:

```
#define MAXNAME (20)

struct person {
     char name[MAXNAME+1] ;
     int age ;
     double income ;
} ;
```

naming - the field names in the struct



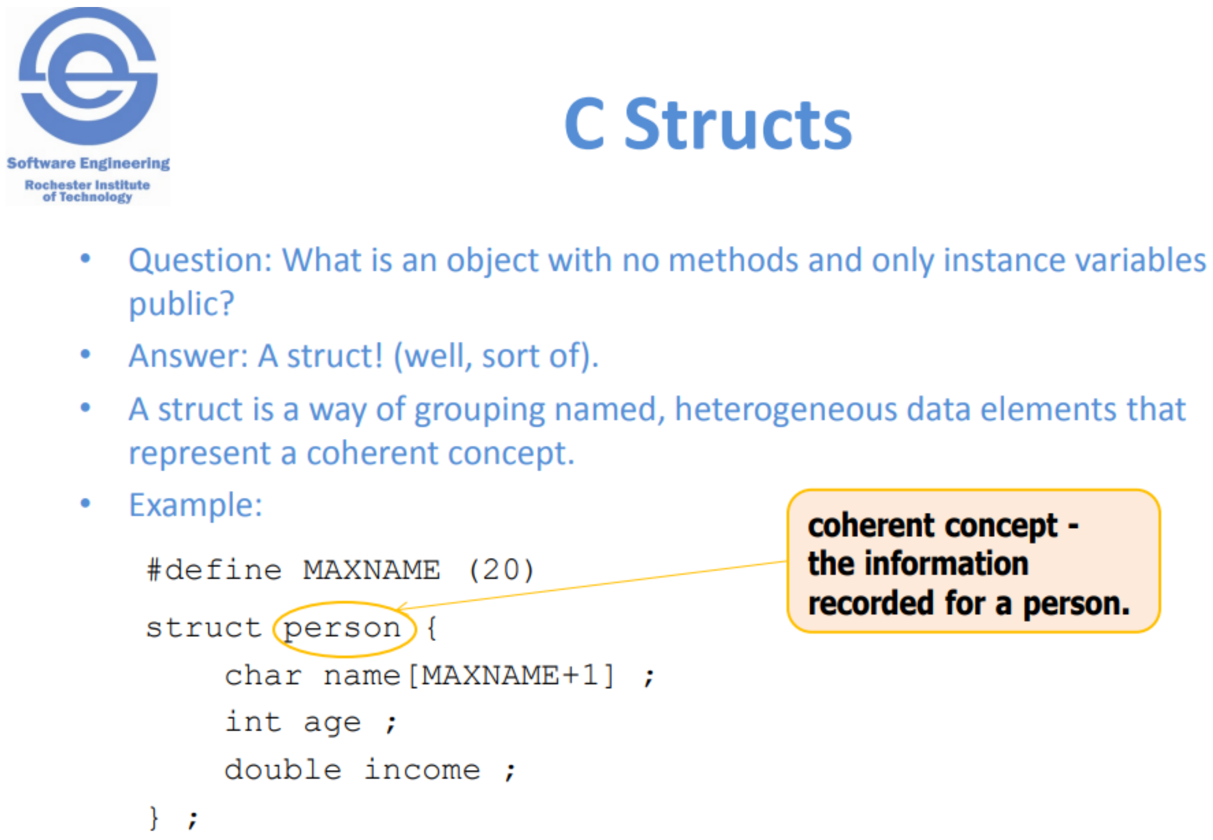
Software Engineering
Rochester Institute of Technology

C Structs

- Question: What is an object with no methods and only instance variables public?
- Answer: A struct! (well, sort of).
- A struct is a way of grouping named, heterogeneous data elements that represent a coherent concept.
- Example:

```
#define MAXNAME (20)
struct person {
    char name[MAXNAME+1] ;
    int age ;
    double income ;
} ;
```

heterogeneous - the fields have different types



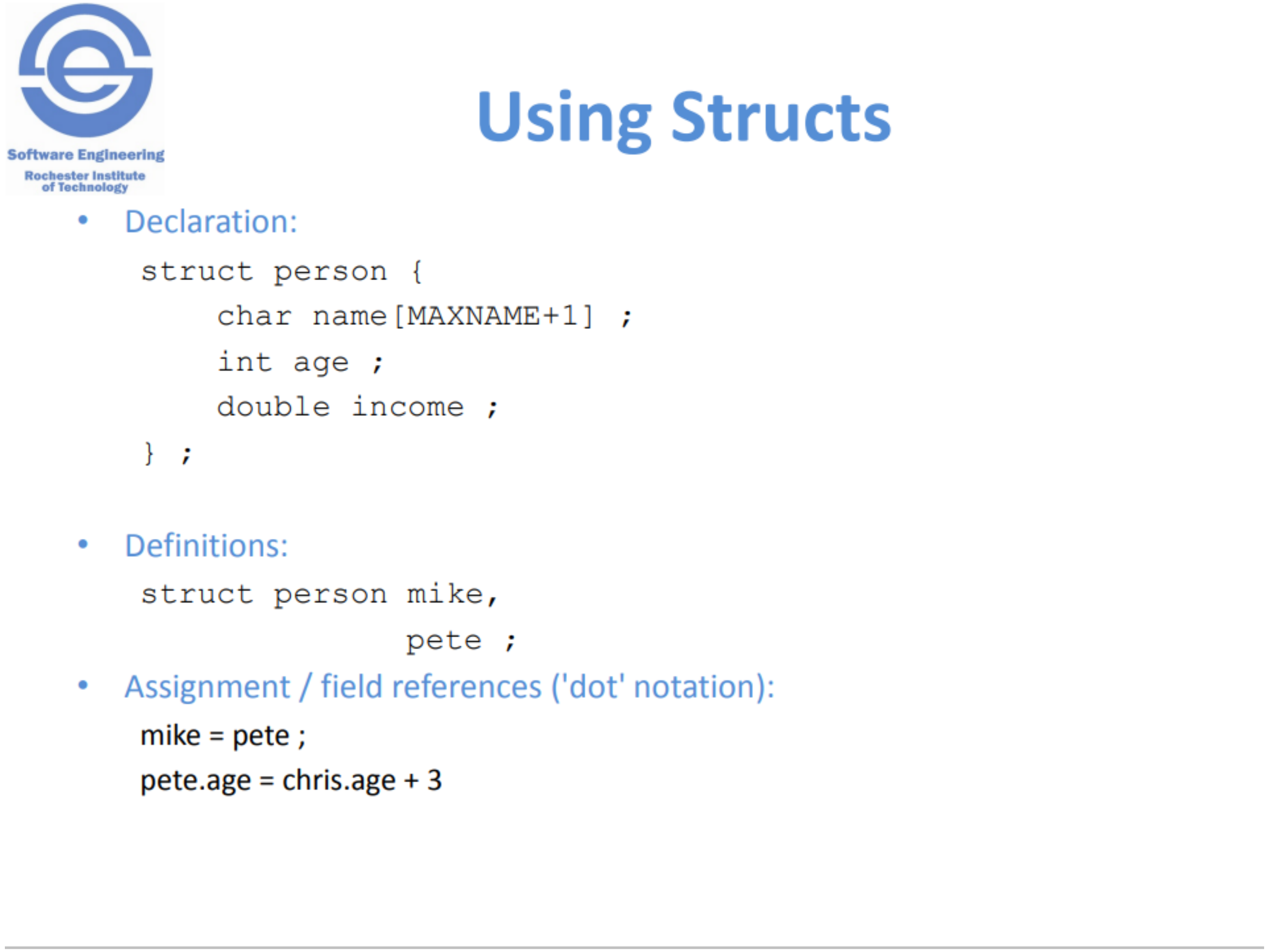
Software Engineering
Rochester Institute of Technology

C Structs

- Question: What is an object with no methods and only instance variables public?
- Answer: A struct! (well, sort of).
- A struct is a way of grouping named, heterogeneous data elements that represent a coherent concept.
- Example:

```
#define MAXNAME (20)
struct person {
    char name[MAXNAME+1] ;
    int age ;
    double income ;
} ;
```

coherent concept - the information recorded for a person.



Software Engineering
Rochester Institute of Technology

Using Structs

- Declaration:

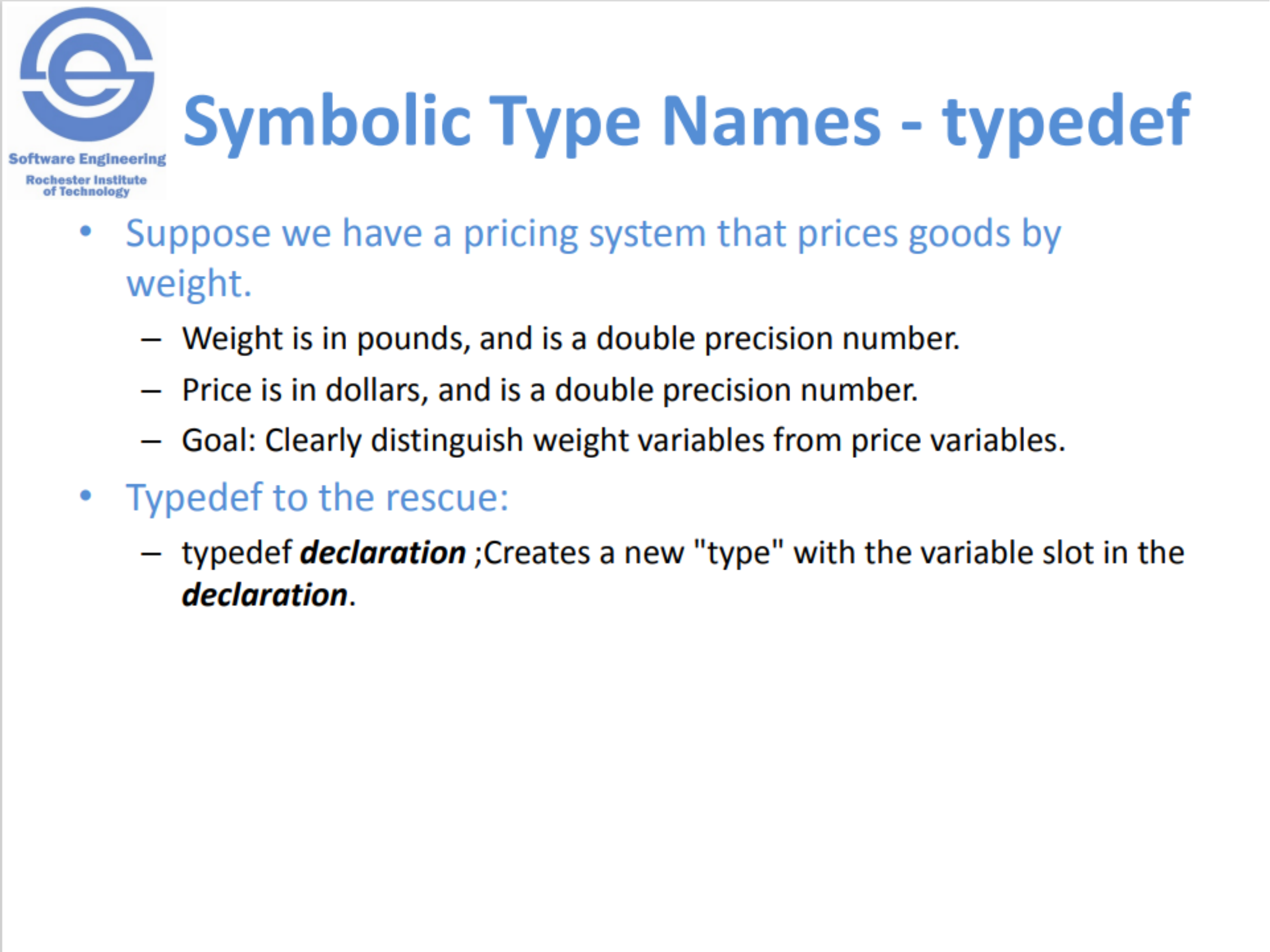
```
struct person {
    char name[MAXNAME+1] ;
    int age ;
    double income ;
} ;
```

- Definitions:

```
struct person mike,
              pete ;
```

- Assignment / field references ('dot' notation):

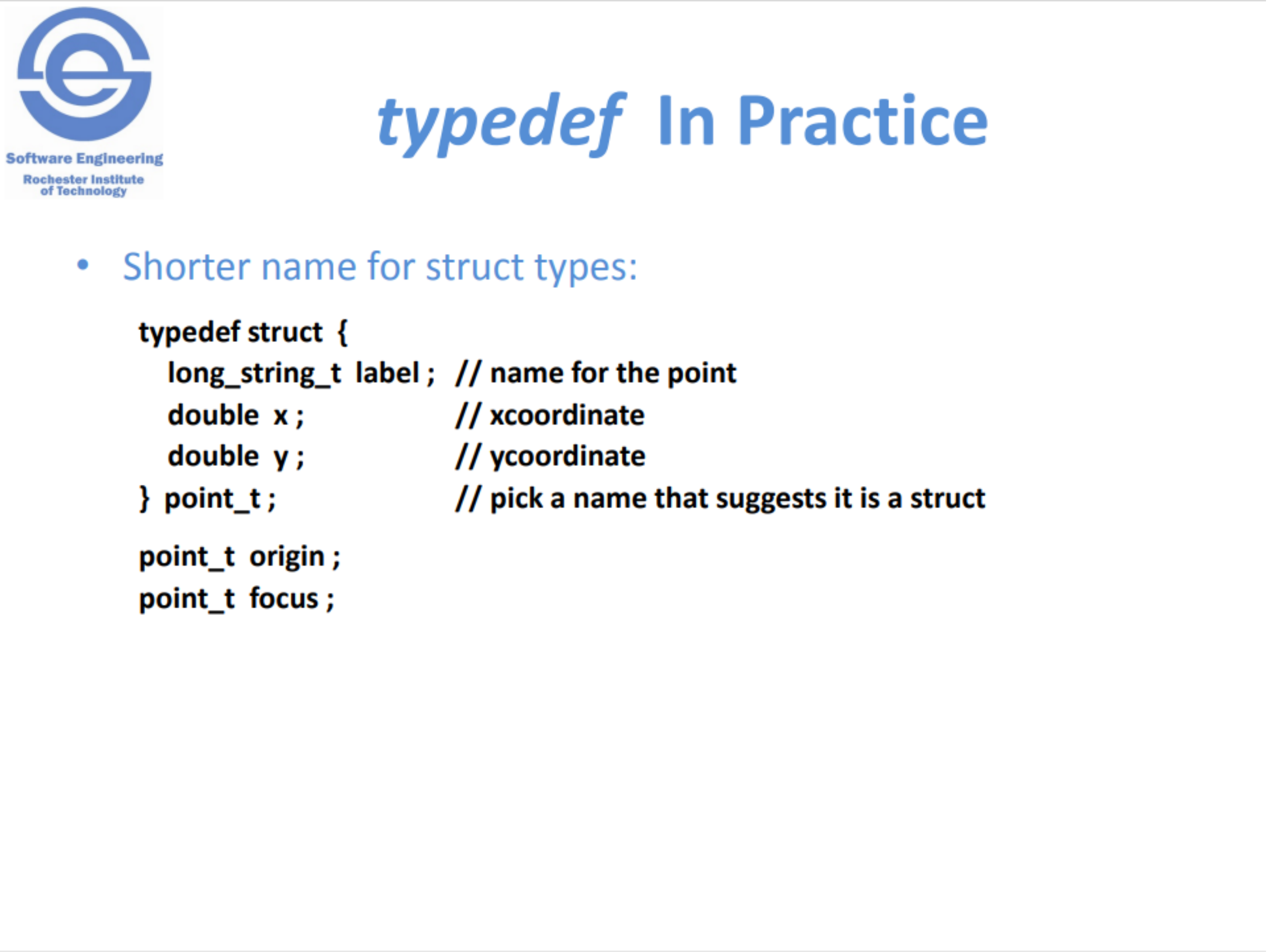
```
mike = pete ;
pete.age = chris.age + 3
```



Software Engineering
Rochester Institute
of Technology

Symbolic Type Names - typedef

- Suppose we have a pricing system that prices goods by weight.
  - Weight is in pounds, and is a double precision number.
  - Price is in dollars, and is a double precision number.
  - Goal: Clearly distinguish weight variables from price variables.
- Typedef to the rescue:
  - typedef ***declaration*** ;Creates a new "type" with the variable slot in the ***declaration***.



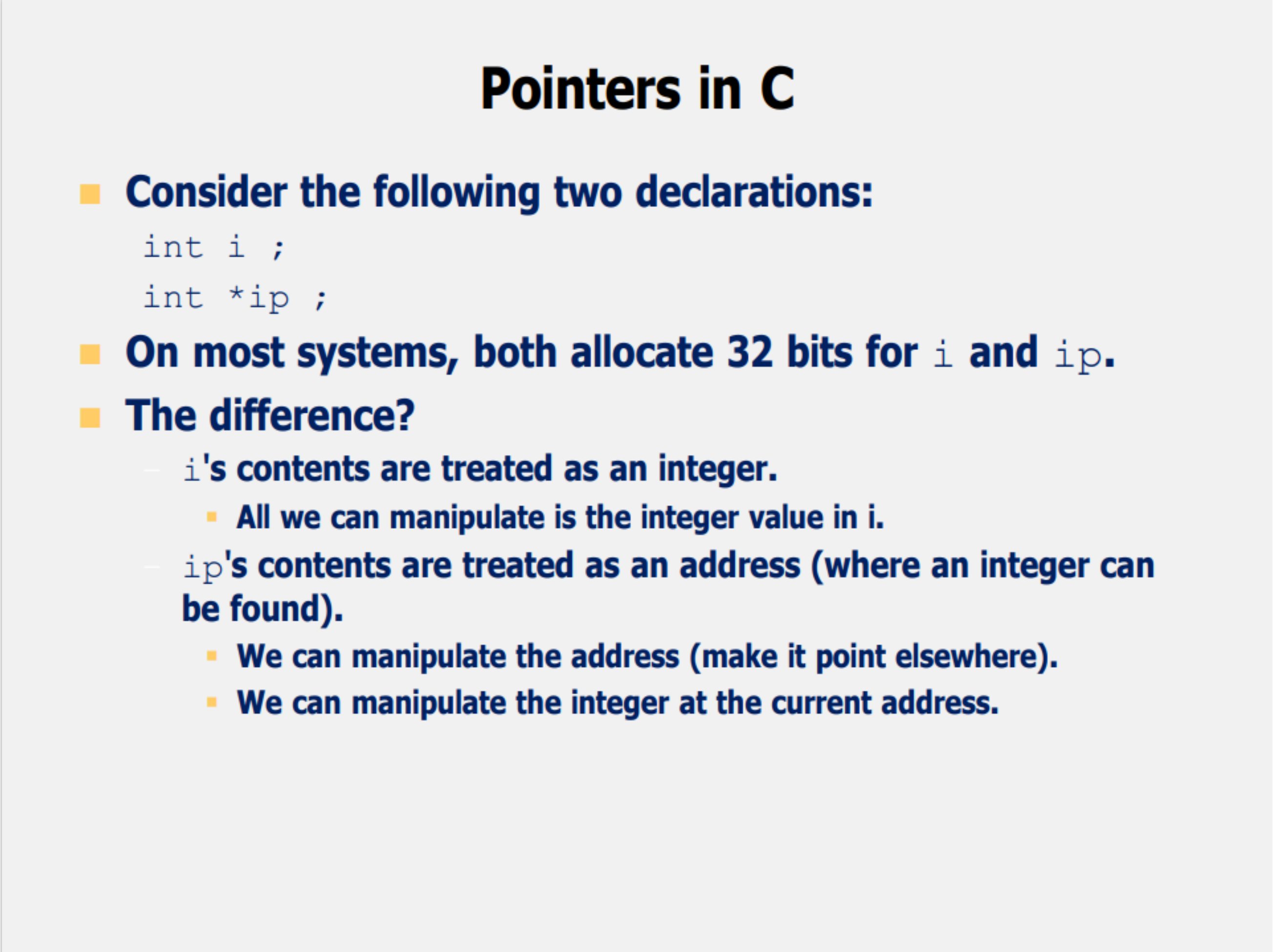
Software Engineering
Rochester Institute
of Technology

typedef In Practice

- Shorter name for struct types:

```
typedef struct  {
   long_string_t  label ;  // name for the point
   double  x ;             // xcoordinate
   double  y ;             // ycoordinate
}  point_t ;               // pick a name that suggests it is a struct

point_t  origin ;
point_t  focus ;
```

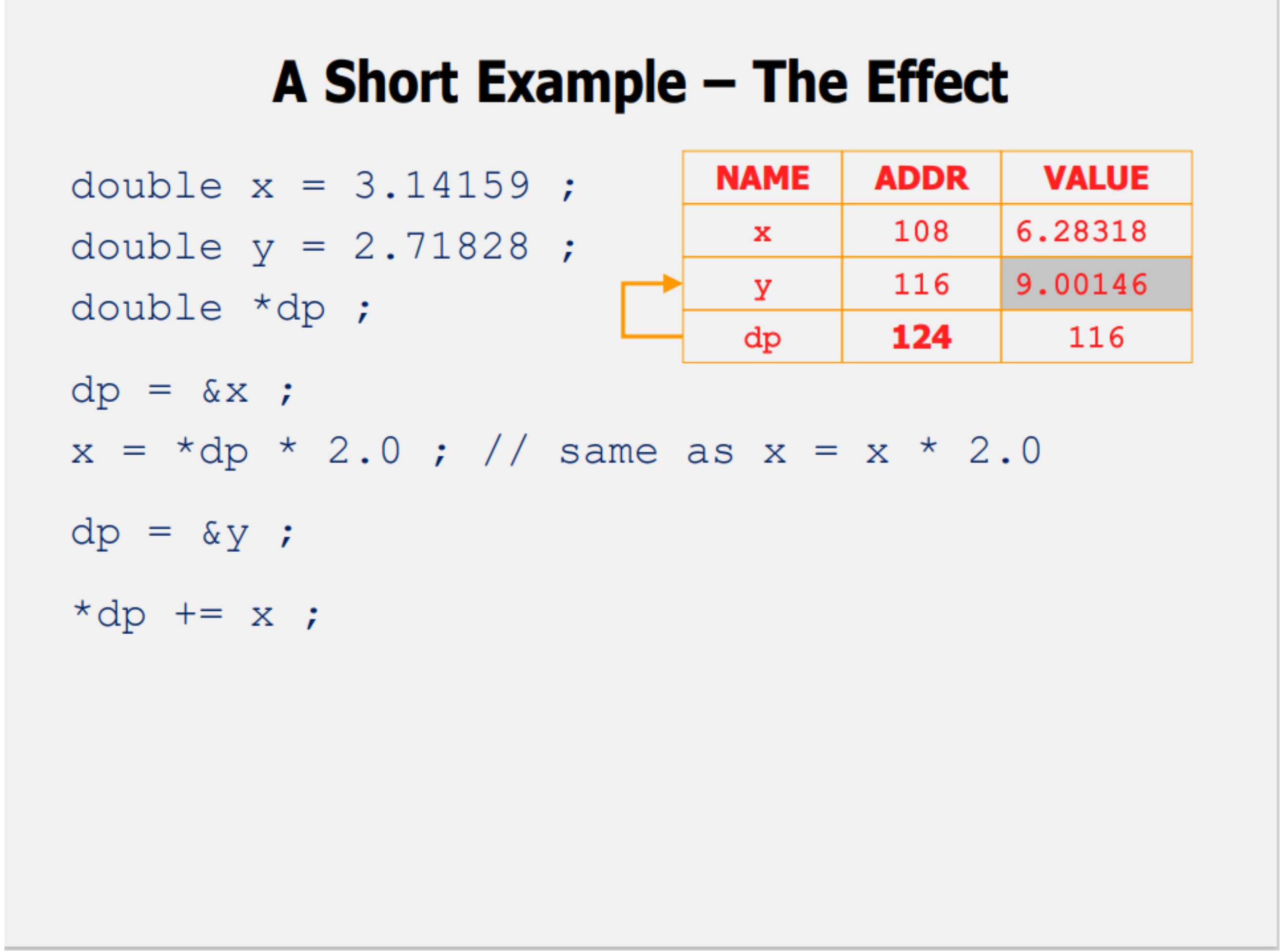


Pointers in C

- Consider the following two declarations:

```
int i ;
int *ip ;
```

- On most systems, both allocate 32 bits for `i` and `ip`.
- The difference?
  - `i`'s contents are treated as an integer.
    - All we can manipulate is the integer value in i.
  - `ip`'s contents are treated as an address (where an integer can be found).
    - We can manipulate the address (make it point elsewhere).
    - We can manipulate the integer at the current address.



A Short Example – The Effect

```
double x = 3.14159 ;
double y = 2.71828 ;
double *dp ;

dp = &x ;
x = *dp * 2.0 ; // same as x = x * 2.0

dp = &y ;

*dp += x ;
```

NAME	ADDR	VALUE
x	108	6.28318
y	116	9.00146
dp	**124**	116

Pointers - Dot vs Arrow Operator

- **The . (dot) operator and the -> (arrow) operator are used to reference individual members of structures**
- **The dot operator is applied to the actual object**
- **The arrow operator is used with a pointer to an object**

Example:

```
struct Employee {
  char first_name[16];
  int  age;
}  emp;
```

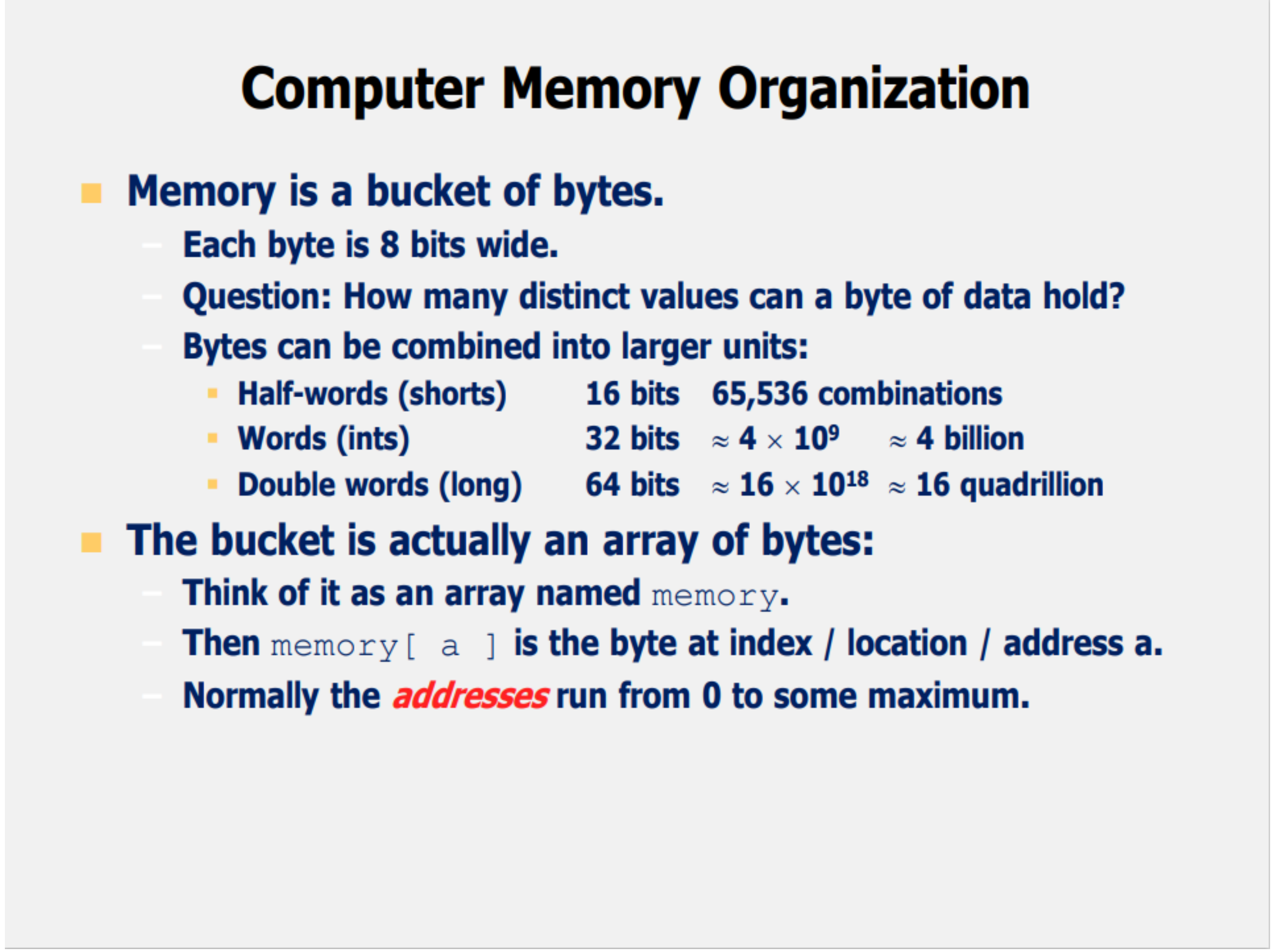
- **Use of (.) dot operator: To assign the value “Alex" to the first_name member of object emp, you would write something as follows –**

```
strcpy(emp.first_name, “Alex");
```

- **Use of (->) arrow operator: If p_emp is a pointer to an object of type Employee, then to assign the value “Alex" to the first_name member of object emp, you would write something as follows –**

```
strcpy(p_emp->first_name, “Alex");
```

Memory Organization



Computer Memory Organization

- Memory is a bucket of bytes.
  - Each byte is 8 bits wide.
  - Question: How many distinct values can a byte of data hold?
  - Bytes can be combined into larger units:

Half-words (shorts)	16 bits	65,536 combinations	
Words (ints)	32 bits	$\approx 4 \times 10^9$	$\approx$ 4 billion
Double words (long)	64 bits	$\approx 16 \times 10^{18}$	$\approx$ 16 quadrillion

- The bucket is actually an array of bytes:
  - Think of it as an array named `memory`.
  - Then `memory[ a ]` is the byte at index / location / address a.
  - Normally the *addresses* run from 0 to some maximum.

Integer Types – Size and Range

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Reference: https://www.tutorialspoint.com/cprogramming/c_data_types.htm

Floating Point Types – Size and Range

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Struct - Memory Layout

- Struct members located in memory in order which the members are declared
- The memory address of the first member will be the same as the address of the Struct itself

Example:

```
struct Employee {
          char id;                  // 8 bits – 1 byte
          int  age;                 // 32 bits – 4 bytes
          char location;            // 8 bits – 1 byte
          short shift;              // 16 bits – 2 bytes
}  emp;
```

How many bytes do we need to store this Struct?

Memory Organization on 32-bit system

Byte 3	Byte 2	Byte 1	Byte 0
Unused	Unused	Unused	ID
Age	Age	Age	Age
Shift	Shift	Unused	Location

12 bytes used instead of 8 Bytes

Re-write Struct to Optimize Memory Layout

```
struct Employee {
        int  age;               // 32 bits – 4 bytes
         short shift;           // 16 bits – 2 bytes
        char id;                // 8 bits – 1 byte
        char location;          // 8 bits – 1 byte
}  emp;
```

How many bytes do we need to store this Struct?

Memory Organization on 32-bit system

Byte 3	Byte 2	Byte 1	Byte 0
Age	Age	Age	Age
Unused	Location	Shift	Shift