DASF004
Basic and Practice in Programming

Lecture 2
Basic Sequencing and Control

Food for your MIND

- Augmented Reality – Display Technology
- How do you see the real world and the virtual world at the same time???
- Hand-held Display
  - “Magnifying glass” approach
- Head-worn Display
-

Monocular Occlusive Display

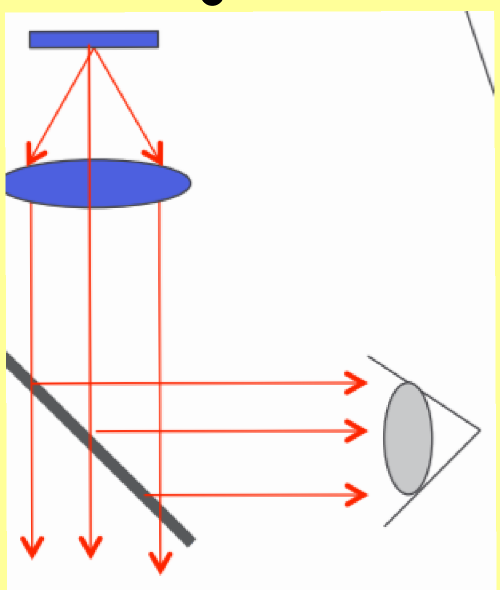
- Monocular – One eye
- Occlusive – Not see-thru
- See the real world with one eye; see the virtual world with the other eye
- The brain combines the two worlds
    - May seems unnatural to you, but it works quite good in practice!
- Cheap

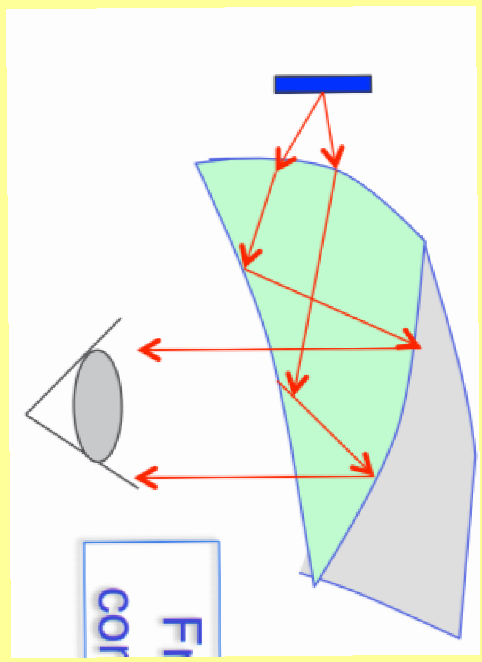
Optical See Through Display

- Real world and virtual world are combined using optics
- Occlusion of virtual world to real world is not possible
-

Optical See Through Display

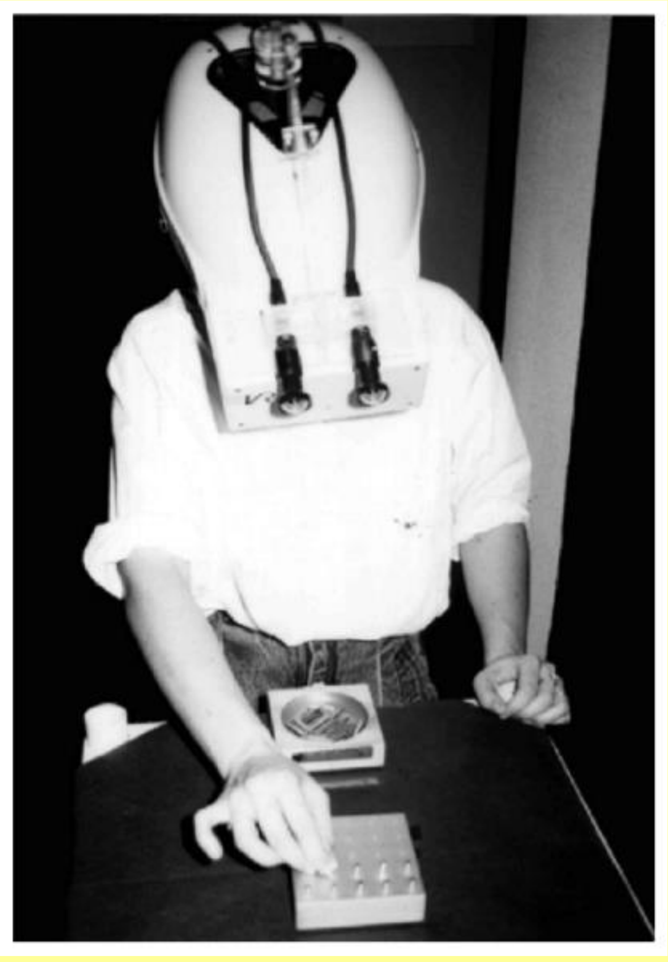
- Real world and virtual world are combined using optics
- Occlusion of virtual world to real world is not possible





Video See Through Display

- Seeing through the camera
- Properties of the naked eyes are reduced to the technical limitations of the cameras
  - Resolution
  - Field-of-view
  - Accommodation (focus)
  - Color

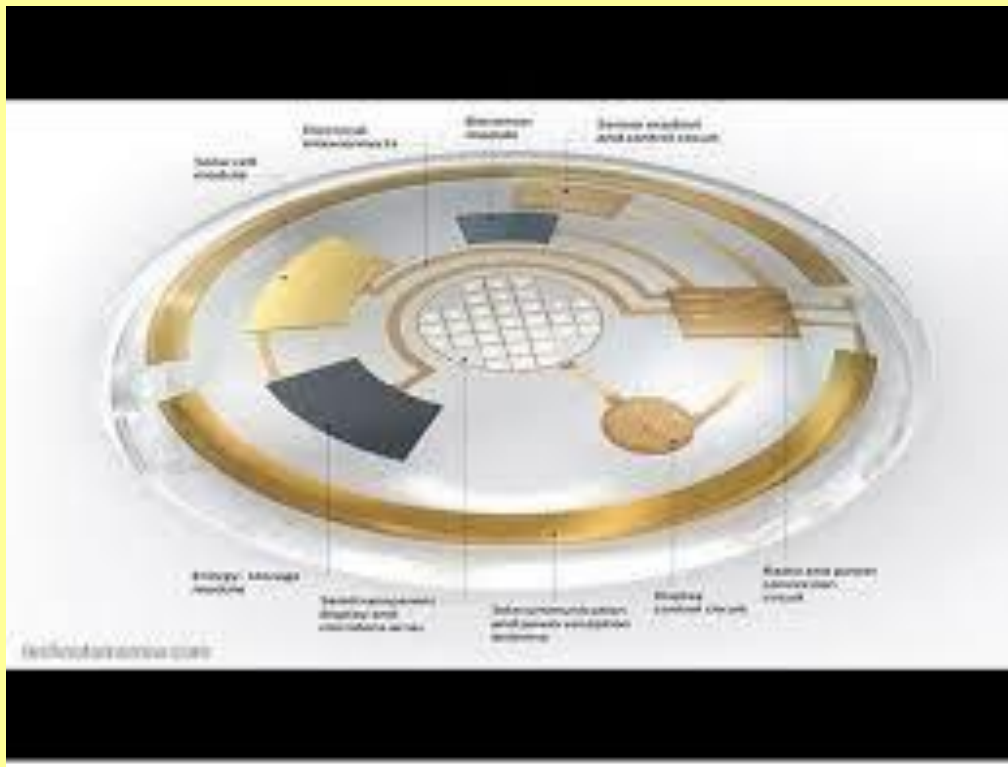
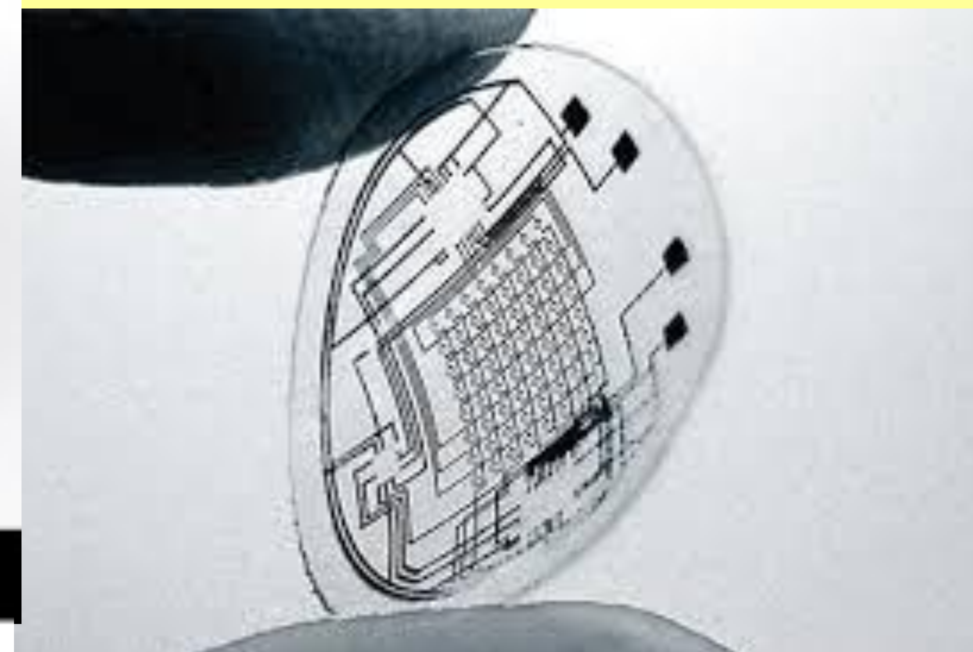


Virtual Retinal Display

- Shooting laser directly into the eye
- “Printing” the graphics on the retina

Bionic Display

- Illustration of concept
- Display for the future

Agenda

Return Value of the main() function

Algorithm

Variables:

- Variable declaration
- Variable type
- Variable assignment
- Increment / Decrement

Arithmetic operators

`stdio.h` library: the **`printf()`** and **`scanf()`** functions

- Type and formating

`if` … **`else`** statement

Comparison operators

Return Value of the main() function

```
int main(void)
{ printf("Hello!");

  return 0;
}
```

- The return value of main() function shows how the program exited.
- The normal exit of program is represented by zero return value.
- If the code has errors, fault etc., it will be terminated by non-zero value.
  - If program's execution fails due to lake of memory, -1 is return.
  - If it fails due to file opening, -2 is return.
  - If it fails due to any invalid input value, -3 return.

Programming Process

- Making goals
  - Understanding of requirements in given problems
- Writing algorithms
  - Writing pseudo codes or flow charts
- Coding
  - Translate the algorithm to C programming language
- Testing
  - Test whether correct outcomes are obtained for possible inputs
- Debugging
  - Modifying programs to correct errors found in testing

Algorithms

- The solution to any computing problem involves executing a series of actions in a specific order.
- A procedure for solving a problem in terms of
  - the actions to be executed, and
  - the order in which these actions are to be executed
- is called an algorithm.
- Correctly specifying the order in which the actions are to be executed is important.

Algorithms

Consider the “rise-and-shine algorithm” followed by one junior executive for getting out of bed and going to work: (1) Get out of bed, (2) take off pajamas, (3) take a shower, (4) get dressed, (5) eat breakfast, (6) carpool to work.
This routine gets the executive to work well prepared to make critical decisions.

Algorithms

Suppose that the same steps are performed in a slightly different order: (1) Get out of bed, (2) take off pajamas, (3) get dressed, (4) take a shower, (5) eat breakfast, (6) carpool to work.
In this case, our junior executive shows up for work soaking wet.
Specifying the order in which statements are to be executed in a computer program is called program control.

Pseudocode

- Pseudocode is an artificial and informal language that helps you develop algorithms.
- Pseudocode is similar to everyday English; it’s convenient and user friendly although it’s not an actual computer programming language.
- Pseudocode programs are *not* executed on computers.
- Rather, they merely help you “think out” a program before attempting to write it in a programming language like C.
- Pseudocode consists purely of characters, so you may conveniently type pseudocode programs into a computer using an editor program.

Pseudocode Example

```
Get first entry;
Call this entry N;
WHILE N is NOT the required entry
DO Get next entry;
     Call this entry N;
ENDWHILE;
```

```
Write “How many pairs of values are to be entered?”
Read numberOfPairs
Set numberRead to 0
WHILE (numberRead < numberOfPairs)
     Write “enter two values separated by a blank; press return”
     Read number1
     Read number2
     IF(number1 < number2)
          Print number1 + “” + number2
     ELSE
          Print number2 + “” + number1
     Increment numberRead
```

Variables and Variables Assignment

- Variable: A memory that stores a value
- Basic variable type:
  - » `int` – integer
  - » `float` – float point number
  - » `char` – a single character
- To declare an integer variable i:
  - » `int i;`
  - » `float j;`
  - » `char k;`
- To assign a value to the variable i:
  - » `i = 1;`
  - » `j = 2.5;`
  - » `k = 'A';`
  - » Note: you can only assign value to a declared variable

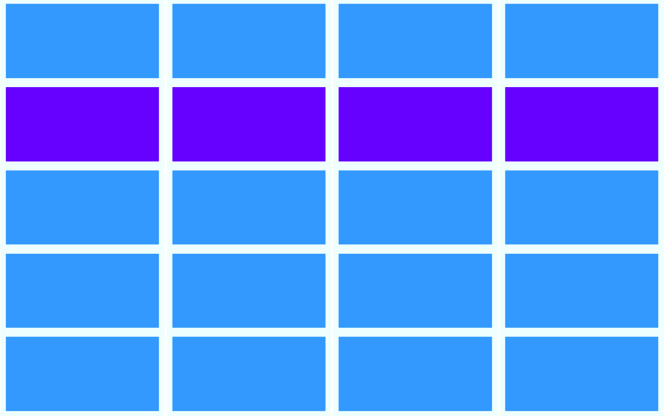
C is case sensitive—uppercase and lowercase letters are different in C, so `a1` and `A1` are different variables.

Variables Declaration

- When a variable is declared:
  - »`int i;`
- A memory with the size of storing an integer is allocated
- A pointer (name of the variable) is pointing to this allocated memory location
  - »

Variables Declaration

- `int i=25;`
- This line of code consist of 4 steps
  - » Step 1: size of an integer is allocated in the memory (size of `int` is 4 byte)
  - »

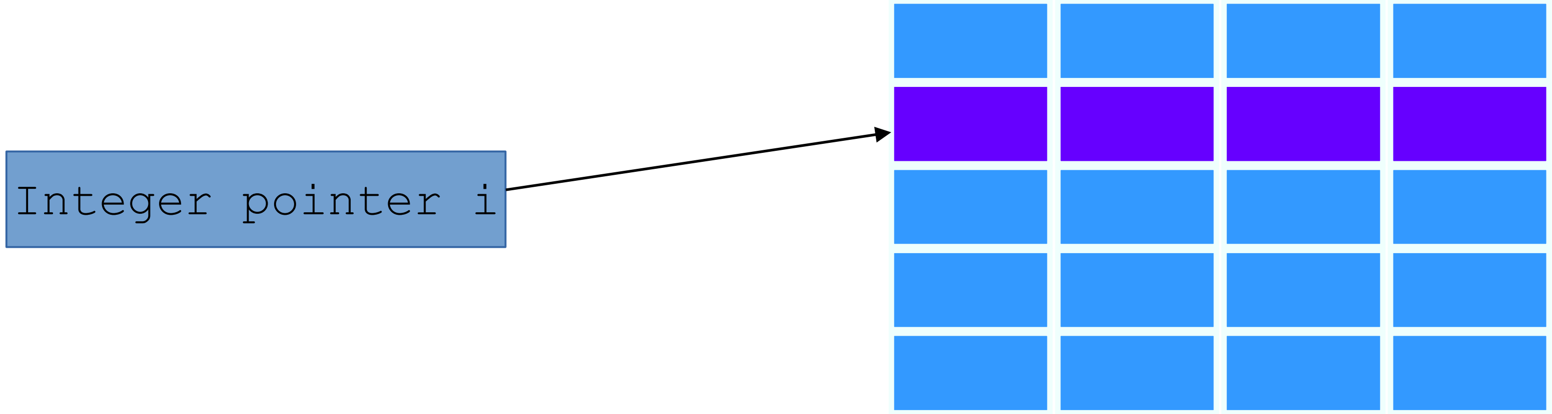


Variables Declaration

- `int i=25;`
- This line of code consist of 4 steps
  - » Step 1: size of an integer is allocated in the memory (size of `int` is 4 byte)
  - » Step 2: a pointer name i is created

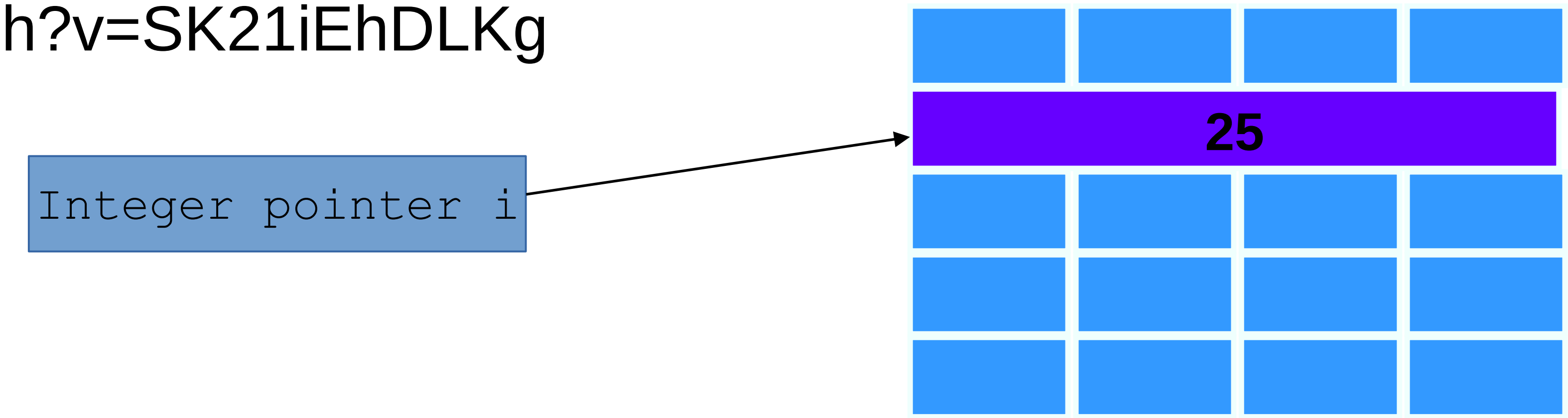
Variables Declaration

- `int i=25;`
- This line of code consist of 4 steps
  - » Step 1: size of an integer is allocated in the memory (size of `int` is 4 byte)
  - » Step 2: a pointer name i is created
  - » Step 3: the pointer i is pointing at the memory location allocated to the integer
  - »



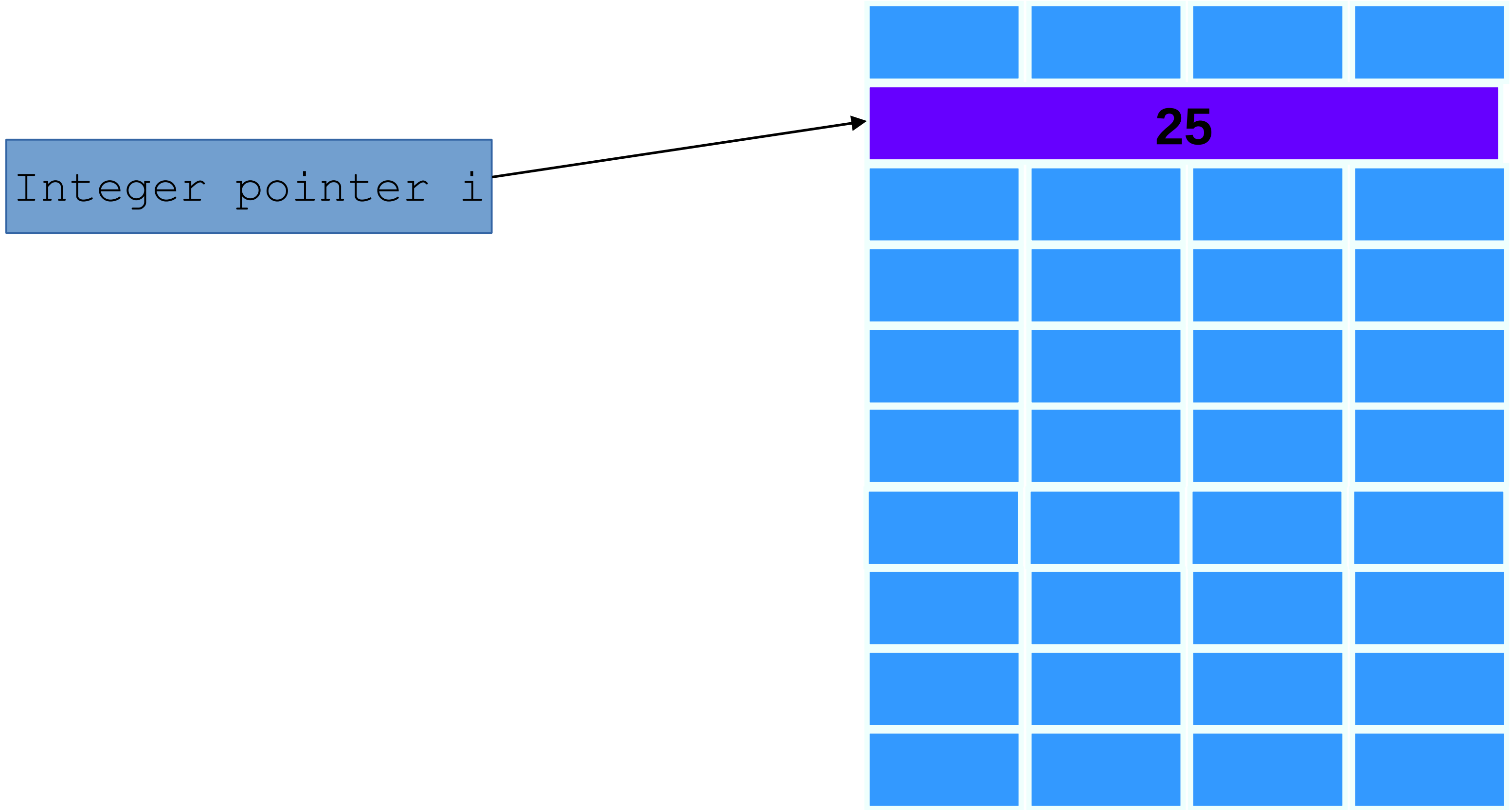
Variables Declaration

- `int i=25;`
- This line of code consist of 4 steps
  - » Step 1: size of an integer is allocated in the memory (size of `int` is 4 byte)
  - » Step 2: a pointer name i is created
  - » Step 3: the pointer i is pointing at the memory location allocated to the integer
  - » Step 4: The value 25 is stored in the memory location
  - »
- Note. Number representation in computer:
- https://www.youtube.com/watch?v=SK21iEhDLKg



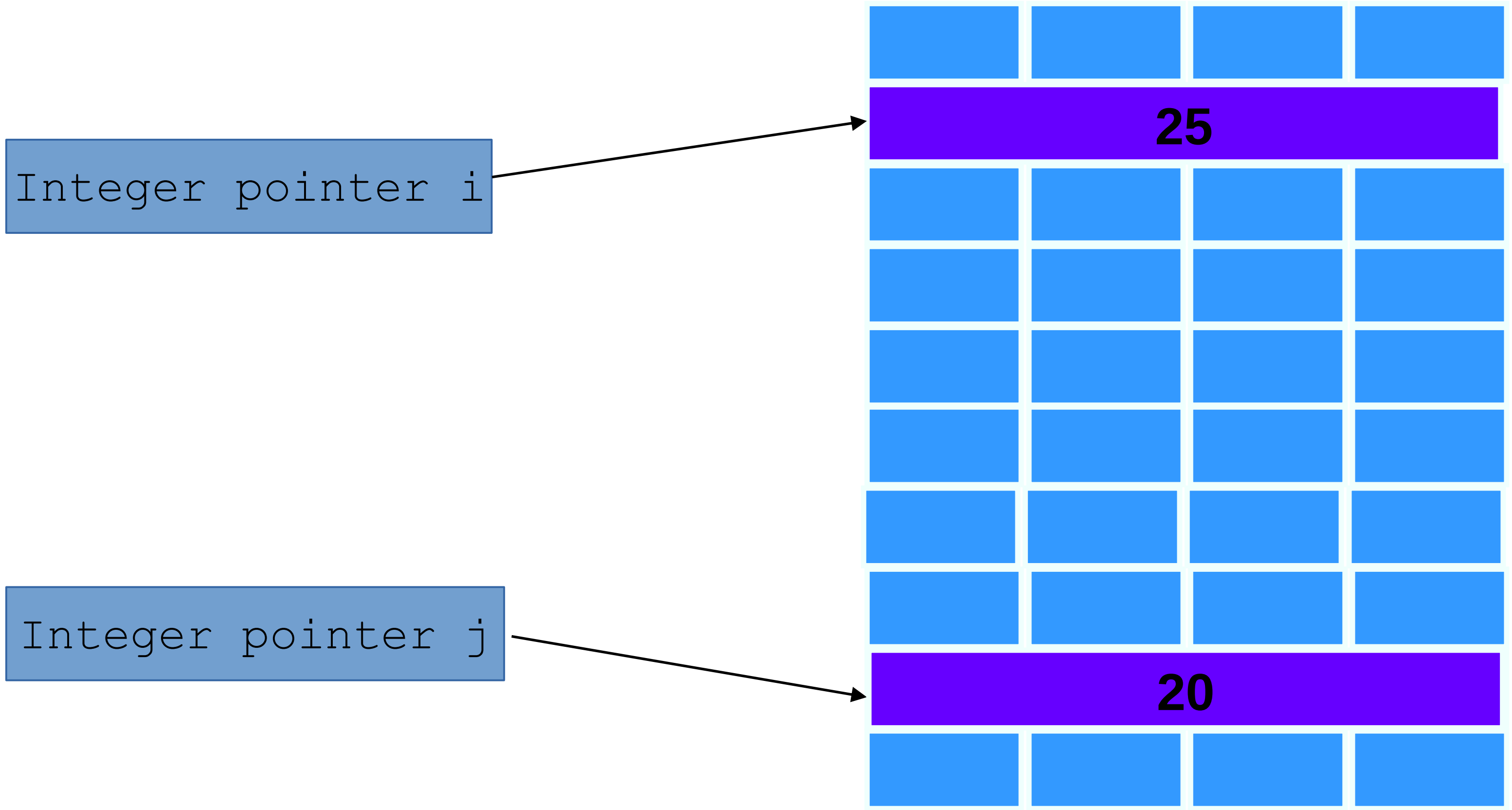
2.4 Memory Concepts

```
int i = 25;
```



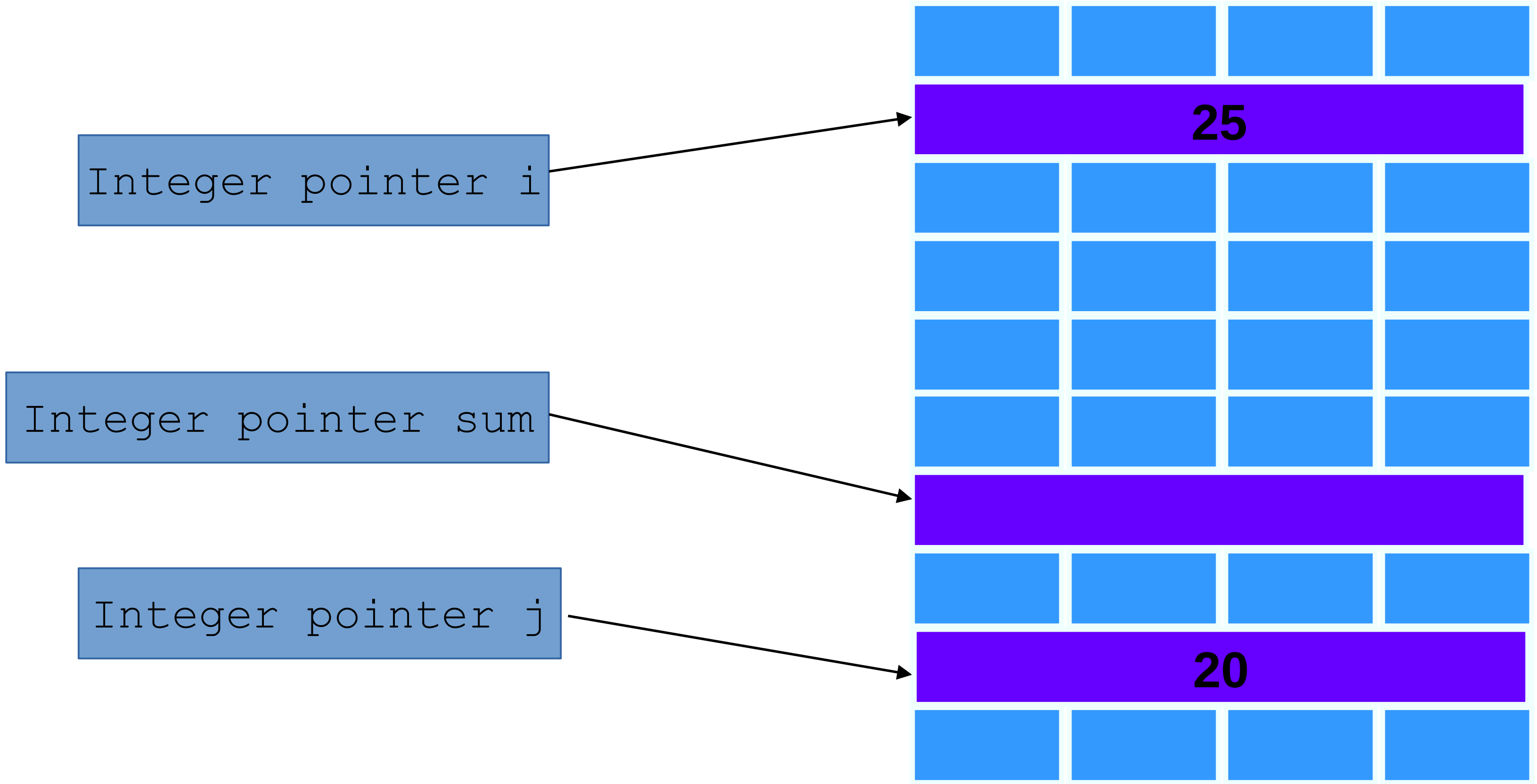
2.4 Memory Concepts

```
int i = 25;
int j = 20;
```



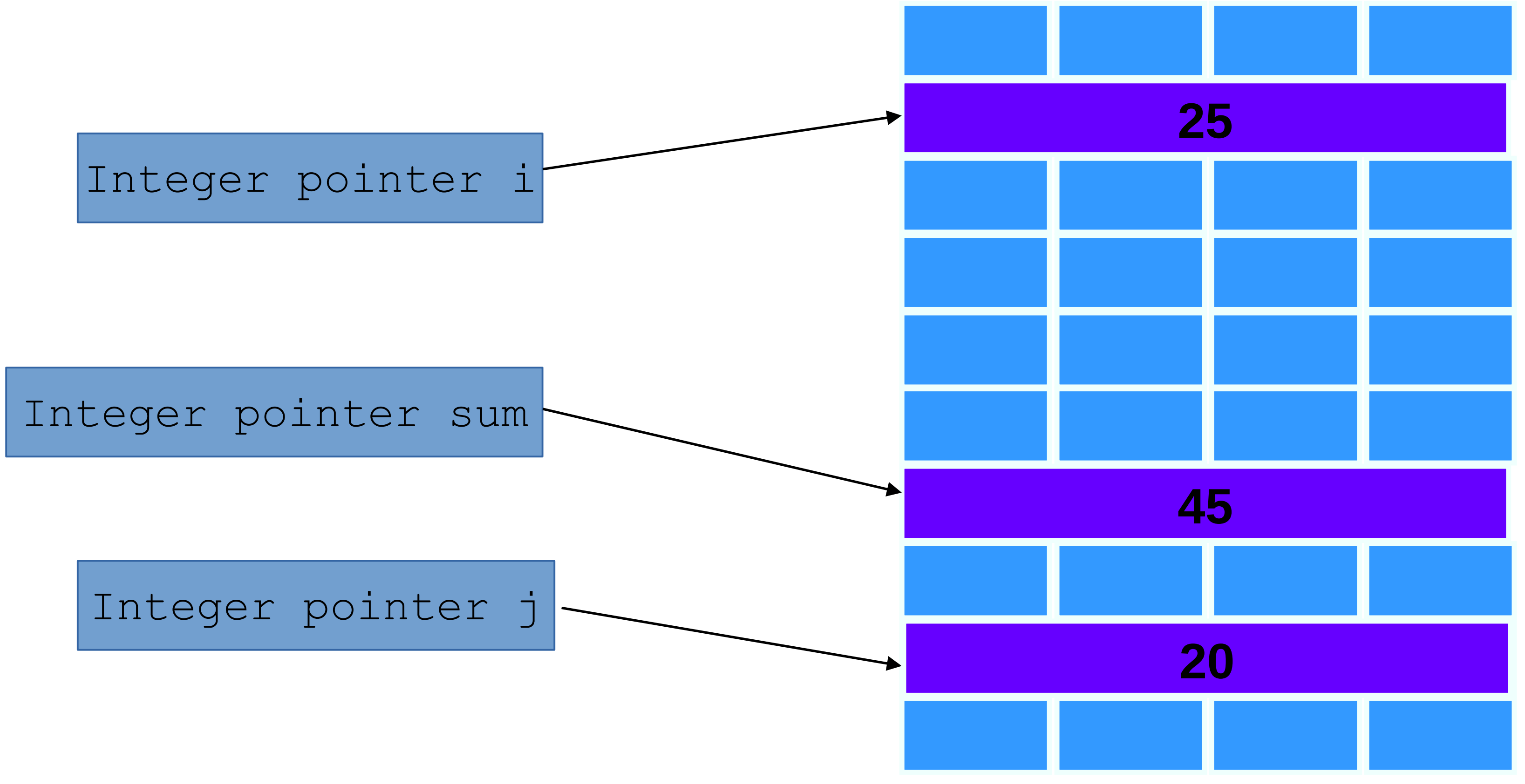
2.4 Memory Concepts

```
int i = 25;
int j = 20;
int sum;
```



2.4 Memory Concepts

```
int i = 25;
int j = 20;
int sum;
sum = i + j;
```



Variables and Variables Assignment

- You can declare a variable and assign its initial value:
  - `int i = 1;`
  - `float j = 2.5;`
  - `char k = 'A';`
- You can reassign a new value to a variable:
  - `int i = 2;`
  - `i = 4;`
- You cannot declare a variable that already exists:
  - `int i = 2;`
  - `i = 4;`
  - `int i = 6; // ERROR!!!`
- Name your variable something meaningful:
  - `int i = 5;                // Meaningless name!`
  - `int midterm_score = 80;// Meaningful name!`

Variables Type

- Basic Variable Type
  - `int, float, char`
    - `3 types of float`
      - `float - 32 bit`
      - `double - 64 bit`
      - `long double - 80 bit`
    - Longer bit means more precision
- Integer Division (Caution!!!)
  - `int x = 7;`
  - `int y = 2;`
  - `float z = x / y;          // z = 3`
- Float Division
  - `float x = 7;`
  - `float y = 2;`
  - `float z = x / y;          // z = 3.5`
- Type casting (Changing type)
  - `int x = 7;`
  - `int y = 2;`
  - `float z = (float) x / (float) y; // z = 3.5`

Type Casting

- Changing the type of a value
- Type casting

```
» int x = 7;
» int y = 2;
» float z = (float) x / (float) y; // z = 3.5
```

Variables Type

- Implicit type casting
- When arithmetic operation is performed on one integer and one float point number, the integer will be casted into float point number

```
»int x = 7;
»float y = 2;
»float z = x / y;  // z = 3.5
```

Variables Type

- More examples:

```
» float x = 2.8;
» int y = (float) x;   // y = 2
»
» float x = 2.2;
» int y = (float) x;   // y = 2

» float x = 2.2;
» int y = x;         // y = 2 (Implicit type casting)
»
» float x = 2.0;
» float y = 7.0;
» int z = y / x;   // z = 3
»
» int x = 2;
» int y = 7;
» float z = (float) (y / x);        // z = 3
» float z2 = (float) y / (float) x; // z2 = 3.5
» float z3 = y / (float) x;         // z3 = 3.5
» float z4 = (float) y / x;         // z4 = 3.5
```

Print out the value of variables

- Use the `printf()` function from `stdio.h` library

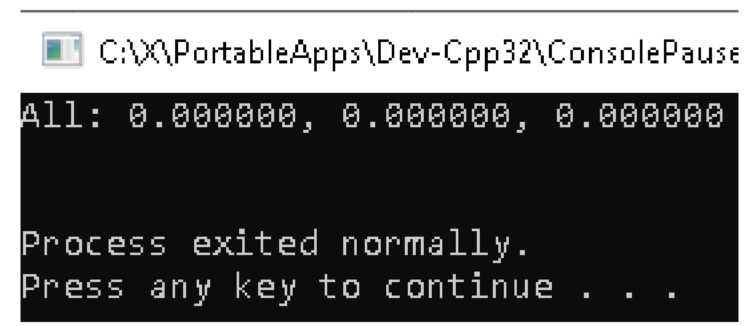
```
int x = 10;
float y = 1.5;
char z = 'A';
printf("The value of x is: %d\n", x);
printf("The value of y is: %f\n", y);
printf("The value of z is: %c\n", z);
printf("All: %d, %f, %c\n", x, y, z);
```

- %d – decimal value
- %f – float point value
- %c – character

Print out the value of variables

- If the type is not matched, an incorrect value will be printed

```
int x = 10;
float y = 1.5;
char z = ‘A’;
printf(“All: %f, %f, %f\n”, x, y, z);
```




```
printf(“All: %d, %d, %d\n”, x, y, z);
```

- %d – decimal value
- %f – float point value
- %c – character

Assignment and Expression

- Arithmetic Operators:
  - `+ - * /` add; sub; multiply; divide
  - `%` modules (i.e. remainder)

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

Fig. 2.9 | Arithmetic operators.

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the *innermost* pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
* / %	Multiplication Division Remainder	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated third. If there are several, they're evaluated left to right.
=	Assignment	Evaluated last.

Fig. 2.10 | Precedence of arithmetic operators.

Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`2 * 5 is` **10**

Step 2. `y = 10 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`10 * 5 is` **50**

Step 3. `y = 50 + 3 * 5 + 7;` *(Multiplication before addition)*

`3 * 5 is` **15**

Step 4. `y = 50 + 15 + 7;` *(Leftmost addition)*

`50 + 15 is` **65**

Step 5. `y = 65 + 7;` *(Last addition)*

`65 + 7 is` **72**

Step 6. `y = 72` *(Last operation—place `72` in `y`)*

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

Assignment operators

- C provides several assignment operators for abbreviating assignment expressions.
- For example, the statement
  - `c = c + 3;`
- can be abbreviated with the addition assignment operator += as
  - `c += 3;`
- The **+=** operator adds the value of the expression on the right of the operator to the value of the variable on the left of the operator and stores the result in the variable on the left of the operator.

Assignment operators

- Any statement of the form
  - *variable = variable operator expression;*
- where *operator is one of the binary operators* +, -, *, / or % *(or others we'll discuss in Chapter 10), can be written in the form*
  - *variable operator= expression;*
- Thus the assignment `c += 3` adds `3` to `c`.
- Figure 3.11 shows the arithmetic assignment operators, sample expressions using these operators and explanations.

Assignment operators

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to `c`
`-=`	`d -= 4`	`d = d - 4`	1 to `d`
`*=`	`e *= 5`	`e = e * 5`	20 to `e`
`/=`	`f /= 3`	`f = f / 3`	2 to `f`
`%=`	`g %= 9`	`g = g % 9`	3 to `g`

Fig. 3.11 | Arithmetic assignment operators.

Increment and decrement operators

- unary increment operator, ++
- unary decrement operator, −−
- If a variable `c` is to be incremented by 1, the increment operator `++` can be used rather than the expressions `c = c + 1` .
- predecrement operators: ++x
- postdecrement operators: x++
- Difference???

Increment and decrement operators

- Preincrementing (predecrementing) a variable causes the variable to be incremented (decremented) by 1, then the new value of the variable is used in the expression in which it appears.
- Postincrementing (postdecrementing) the variable causes the current value of the variable to be used in the expression in which it appears, then the variable value is incremented (decremented) by 1.

Increment and decrement operators

Operator	Sample expression	Explanation
++	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

Fig. 3.12 | Increment and decrement operators

Increment and decrement operators

```c
// Fig. 3.13: fig03_13.c
// Preincrementing and postincrementing.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   int c; // define variable

   // demonstrate postincrement
   c = 5; // assign 5 to c
   printf( "%d\n", c ); // print 5
   printf( "%d\n", c++ ); // print 5 then postincrement
   printf( "%d\n\n", c ); // print 6

   // demonstrate preincrement
   c = 5; // assign 5 to c
   printf( "%d\n", c ); // print 5
   printf( "%d\n", ++c ); // preincrement then print 6
   printf( "%d\n", c ); // print 6
} // end function main
```

Fig. 3.13 | Preincrementing and pos

```
5
5
6

5
6
6
```

Fig. 3.13 | Preincrementing and postincrementing. (Part 2 of 2.)

Taking input from user

- Use the `scanf()` function from `stdio.h` library

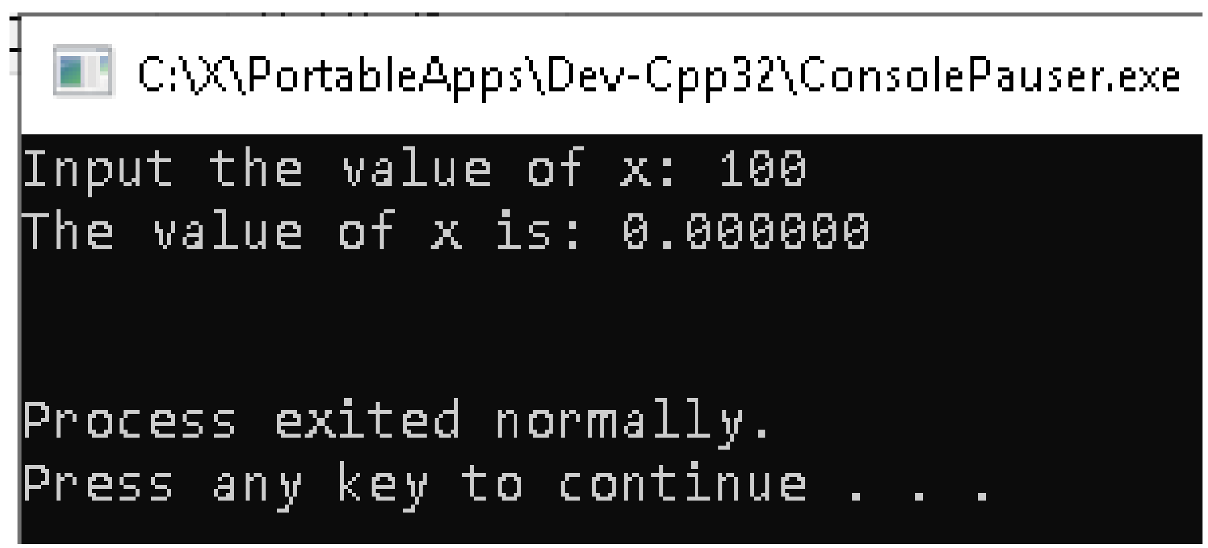
```
int x;                                          // Declare variable
x
printf("Input the value of x: ");         // Prompt user for input
scanf("%d", &x);                          // Assign user input to x
printf("The value of x is: %d\n", x);    // Print out value of x
```

- The ampersand (&) tells `scanf` the location (or address) in memory at which the variable `x` is stored.
- The use of ampersand (&) is often confusing to novice programmers or to people who have programmed in other languages that do not require this notation.
-
- For now, just remember to precede each variable in every call to `scanf` with an ampersand (&).
-
- Also, be careful of the type of the variable.

Taking input from user

❖ The type must match

```
float x;                                // Declare variable x
printf("Input the value of x: ");       // Prompt user for input
scanf("%d", &x);                        // Assign user input to x
printf("The value of x is: %f\n", x);   // Print out value of x
```



```c
// Fig. 2.5: fig02_05.c
// Addition program.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   int integer1; // first number to be entered by user
   int integer2; // second number to be entered by user
   int sum; // variable in which sum will be stored

   printf( "Enter first integer\n" ); // prompt
   scanf( "%d", &integer1 ); // read an integer

   printf( "Enter second integer\n" ); // prompt
   scanf( "%d", &integer2 ); // read an integer

   sum = integer1 + integer2; // assign total to sum

   printf( "Sum is %d\n", sum ); // print sum
} // end function main
```

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Fig. 2.5 | Addition program. (Part 2 of 2.)

`if` statement

- If the condition is met (i.e., the condition is true), the statement in the body of the `if` statement is executed
- If the condition is not met (i.e., the condition is false), the body statement is not executed.
- Whether the body statement is executed or not, after the `if` statement completes, execution proceeds with the next statement after the `if` statement.

```
if(x == 1)
{ printf("The value of x is 1.\n");
}
```

If the body statement contains one line of code only, the brace can be ignored.

```
if(x == 1)
  printf("The value is 1.\n");
```

```
if(score >= 60)
{ printf("Score >= 60.\n");
  printf("Grade is pass.\n");
}
```

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality operators			
$=$	`==`	`x == y`	x is equal to y
$\neq$	`!=`	`x != y`	x is not equal to y
Relational operators			
$>$	`>`	`x > y`	x is greater than y
$<$	`<`	`x < y`	x is less than y
$\geq$	`>=`	`x >= y`	x is greater than or equal to y
$\leq$	`<=`	`x <= y`	x is less than or equal to y

Fig. 2.12 | Equality and relational operators.

```c
// Fig. 2.13: fig02_13.c
// Using if statements, relational
// operators, and equality operators.
#include <stdio.h>

// function main begins program execution
int main( void )
{
   int num1; // first number to be read from user
   int num2; // second number to be read from user

   printf( "Enter two integers, and I will tell you\n" );
   printf( "the relationships they satisfy: " );

   scanf( "%d%d", &num1, &num2 ); // read two integers

   if ( num1 == num2 ) {
      printf( "%d is equal to %d\n", num1, num2 );
   } // end if

   if ( num1 != num2 ) {
      printf( "%d is not equal to %d\n", num1, num2 );
   } // end if
```

Fig. 2.13 | Using if statements, relational operators, and equality operators. (Part 1 of 3.)

```
    if ( num1 < num2 ) {
       printf( "%d is less than %d\n", num1, num2 );
    } // end if

    if ( num1 > num2 ) {
       printf( "%d is greater than %d\n", num1, num2 );
    } // end if

    if ( num1 <= num2 ) {
       printf( "%d is less than or equal to %d\n", num1, num2 );
    } // end if

    if ( num1 >= num2 ) {
       printf( "%d is greater than or equal to %d\n", num1, num2 );
    } // end if
} // end function main
```

Fig. 2.13 | Using `if` statements, relational operators, and equality operators. (Part 2 of 3.)

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

Fig. 2.13 | Using `if` statements, relational operators, and equality operators. (Part 3 of 3.)

The `if…else` selection statement

```
if ( grade >= 60 ) {
   printf( "Passed\n" );
}
else {
   printf( "Failed\n" );
}
```

The `if…else` selection statement

Nested if...else Statements

- Nested `if…else` statements test for multiple cases by placing `if…else` statements inside `if…else` statements.
-
- For example, the following pseudocode statement will print `A` for exam grades greater than or equal to `90`, `B` for grades greater than or equal to `80` (but less than `90`), `C` for grades greater than or equal to `70` (but less than `80`), `D` for grades greater than or equal to `60` (but less than `70`) and `F` for all other grades.

The `if…else` selection statement

```
If student's grade is greater than or equal to 90
  Print "A"
else
  If student's grade is greater than or equal to 80
    Print "B"
  else
    If student's grade is greater than or equal to 70
      Print "C"
    else
      If student's grade is greater than or equal to 60
        Print "D"
      else
        Print "F"
```

The `if…else` selection statement

- This pseudocode may be written in C as

```
if ( grade >= 90 )
   printf( "A" );
else
   if ( grade >= 80 )
      printf("B");
   else
      if ( grade >= 70 )
         printf("C");
      else
         if ( grade >= 60 )
            printf( "D" );
         else
            printf( "F" );
```

The `if…else` selection statement

- You may prefer to write the preceding `if` statement as
- `if ( grade >= 90 )`
  `printf( "A" );`
- `else if ( grade >= 80 )`
  `printf( "B" );`
- `else if ( grade >= 70 )`
  `printf( "C" );`
- `else if ( grade >= 60 )`
  `printf( "D" );`
- `else`
  `printf( "F" );`

`if` statement and `if` … `else` statement

- What is the difference???

```
if(x == 1)
{ printf("x is 1.\n");
}
if(x == 2)
{ printf("x is 2.\n");
}
if(x == 3)
{ printf("x is 3.\n");
}
if(x != 1 && x != 2 && x != 3)
{ printf("x is not 1, nor 2, nor 3.\n");
}
```

```
if(x == 1)
{ printf("x is 1.\n");
}
else if(x == 2)
{ printf("x is 2.\n");
}
else if(x == 3)
{ printf("x is 3.\n");
}
else
{ printf("x is not 1, nor 2, nor 3.\n");
}
```

Q&A?