Ho Chi Minh City University of Technology
Faculty of Computer Science and Engineering

Chapter 3: Variables and Basic Data Types

Introduction to Computer Programming
(C language)

Nguyễn Tiến Thịnh, Ph.D.

Email: ntthinh@hcmut.edu.vn

2022 – 2023, Semester 1

Course Content

- C.1. Introduction to Computers and Programming
- C.2. C Program Structure and its Components
- **C.3. Variables and Basic Data Types**
- C.4. Selection Statements
- C.5. Repetition Statements
- C.6. Functions
- C.7. Arrays
- C.8. Pointers
- C.9. File Processing

References

- [1] "*C: How to Program*", 7th Ed. – Paul Deitel and Harvey Deitel, Prentice Hall, 2012.
- [2] "*The C Programming Language*", 2nd Ed. – Brian W. Kernighan and Dennis M. Ritchie, Prentice Hall, 1988
- and others, especially those on the Internet

Content

- Introduction
- Data and Data Types
- enum Data Type
- struct Data Type
- Variables and Variable Declaration
- Constant Definition
- Expressions
- Operators
- Summary

Introduction

- Given a set of n positive numbers, find the smallest one.

(Chapter 1 – Pseudo code)

```
Algorithm findMinNumber
  Input: positiveNumber[n] which is an array of n positive double values
  Output: minNumber which is the smallest one whose type is double
  Purpose: find the smallest number in a collection
  Precondition: n data inputs are positive.
Begin Algorithm
        Check positiveNumber[n] contains only positive values
        minNumber = positiveNumber[1]
        iteration = 2
        While (iteration <= n)
        Begin While
                If (minNumber <= positiveNumber[iteration]) Then
                        iteration = iteration + 1
                Else
                Begin
                        minNumber = positiveNumber[iteration]
                        iteration = iteration + 1
                End
        End While
End Algorithm
```

Introduction

- Given a set of n positive numbers, find the smallest one.

(Chapter 1 – Real code in C)

```
void main() {
    double positiveNumber[10] = {2, 1, 3, 10, 8, 3, 4, 5, 9, 12};
    int n = 10;
    double minNumber = positiveNumber[0];
    int iteration = 1;
    while (iteration < n) {
        if (minNumber <= positiveNumber[iteration])
            iteration = iteration + 1;
        else {
            minNumber = positiveNumber[iteration];
            iteration = iteration + 1;
        }
    }
}
```

Introduction

- Given a set of n positive numbers, find the smallest one.

(Chapter 1 – Real code in C)

```
void main() {
    double positiveNumber[10] = {2, 1, 3, 10, 8, 3, 4, 5, 9, 12};
    int n = 10;
    double minNumber = positiveNumber[0];
    int iteration = 1;                    /* Variable declarations */
    while (iteration < n) {
        if (minNumber <= positiveNumber[iteration])
            iteration = iteration + 1;
        else {
            minNumber = positiveNumber[iteration];
            iteration = iteration + 1;
        }
    }
}
```

Introduction

- Given a set of n positive numbers, find the smallest one.

(Chapter 1 – Real code in C)

```
void main() {
    INPUT
    OUTPUT
    SUPPORTING ONES    /* Variable declarations */
    while (iteration < n) {
        if (minNumber <= positiveNumber[iteration])
            iteration = iteration + 1;
        else {
            minNumber = positiveNumber[iteration];
            iteration = iteration + 1;
        }
    }
}
```

Introduction

- Handle data in a C program
  - Input
  - Output
  - Supporting ones

What?

→ Declare, Store, Process (Manipulate)?

→ Who declares?

→ Who stores?

Where?

→ Who processes (manipulates)?

Introduction

- Handle data in a C program
  - Input
  - Output
  - Supporting ones

What?
Values + Types
(Our real world)

→ Declare, Store, Process (Manipulate)?

→ Who declares? - We

→ Who stores? - Computer

Where?
Memory
(Computer's world)

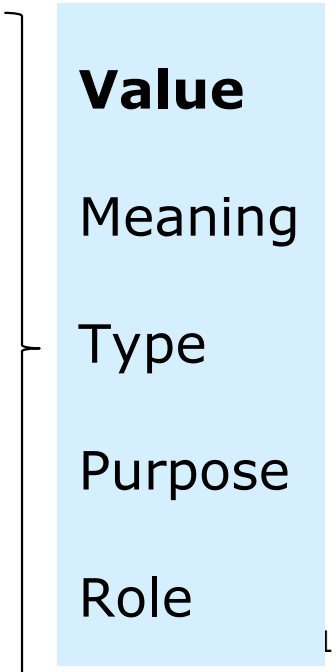
→ Who processes (manipulates)? - Computer

Data and Data Types

- Data
  - "information, especially facts or numbers, collected for examination and consideration and used to help decision-making, or information in an electronic form that can be stored and processed by a computer" – Cambridge Dictionary
  - Examples:
    - 9.5
    - "Introduction to Computer Programming"
    - 'A'
    - ("59800172", "Chau", "IT", 1998)
    - (2, 5, 10, 3, 8, 9, 4, 1, 7, 6)
    - 1
    - 1, 2, 3, …
    - …

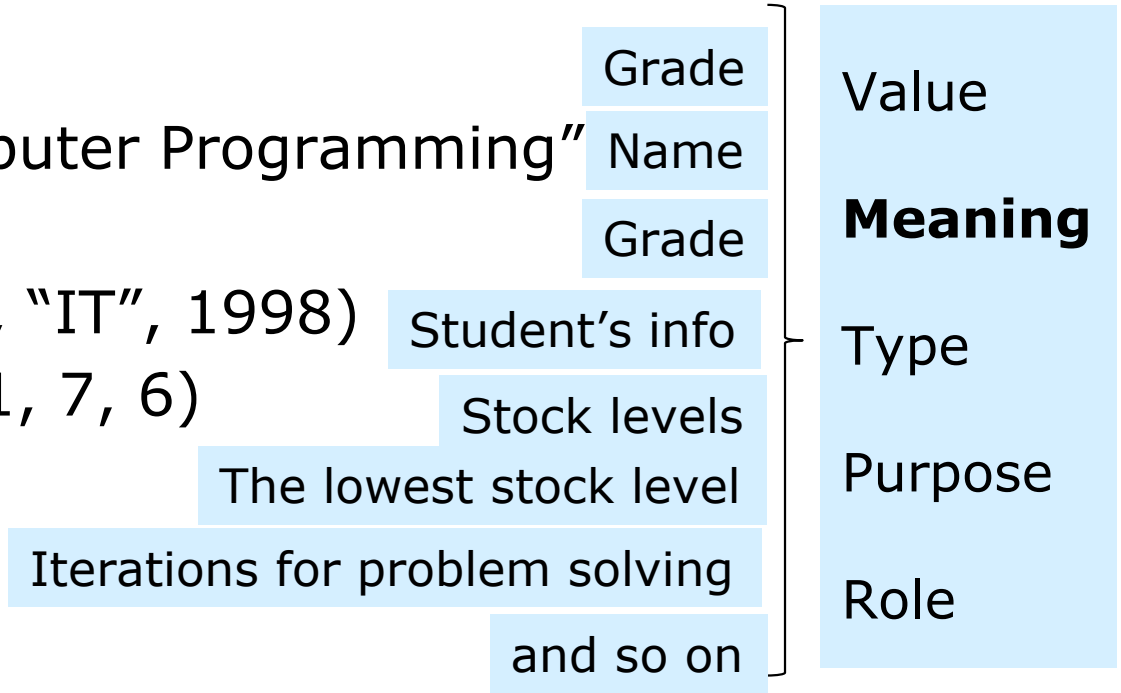
Data and Data Types

- Data
  - "information, especially facts or numbers, collected for examination and consideration and used to help decision-making, or information in an electronic form that can be stored and processed by a computer" – Cambridge Dictionary
  - Examples:
    - 9.5
    - "Introduction to Computer Programming"
    - 'A'
    - ("59800172", "Chau", "IT", 1998)
    - (2, 5, 10, 3, 8, 9, 4, 1, 7, 6)
    - 1
    - 1, 2, 3, …
    - …



Data and Data Types

- Data
  - "information, especially facts or numbers, collected for examination and consideration and used to help decision-making, or information in an electronic form that can be stored and processed by a computer" – Cambridge Dictionary
  - Examples:
    - 9.5
    - "Introduction to Computer Programming"
    - 'A'
    - ("59800172", "Chau", "IT", 1998)
    - (2, 5, 10, 3, 8, 9, 4, 1, 7, 6)
    - 1
    - 1, 2, 3, …
    - …



Data and Data Types

- Data
  - "information, especially facts or numbers, collected for examination and consideration and used to help decision-making, or information in an electronic form that can be stored and processed by a computer" – Cambridge Dictionary

Types (atomic, non-atomic):
- Real numbers
- Sequences of characters
- Characters
- Records
- Sets of elements
- Integer numbers
- Natural numbers
- …

→ Value sets + data manipulations
→ Storage

puter Programming"

, "IT", 1998)

1, 7, 6)

Grade
Name
Grade
Student's info
Stock levels
The lowest stock level
Iterations for problem solving
and so on

Value
Meaning
Type
Purpose
Role

Data and Data Types

- Data
  - "information, especially facts or numbers, collected for examination and consideration and used to help decision-making, or information in an electronic form that can be stored and processed by a computer" – Cambridge Dictionary
- Data Types
  - Represent data processed in a computer-based program
  - Specify what kind of data is to be stored in memory
    - How much memory should be allocated
    - How to interpret the bits stored in the memory
  - Specify what operations are allowed for data manipulation

Data Types in C

- Built-in data types (primitive/fundamental)
  - char (signed char), unsigned char
  - short int, unsigned short, int, unsigned int, long int, unsigned long int, long long int, unsigned long long
  - float, double, long double
  - void
  - enum (enumerated data associated with integers)
- Derived data types
  - arrays [] of objects of a given type
  - pointers * to objects of a given type
  - structures struct containing objects of other types
  - union containing any one of several objects of various types

Built-in Data Types in C

Type	Keyword	Size	Range
Characters	char	1 byte	-128 to 127
Unsigned characters	unsigned char	1 byte	0 to 255
Short integer numbers	short	2 bytes	-32768 to 32767
Unsigned short integer numbers	unsigned short	2 bytes	0 to 65535
Integer numbers	int	4 bytes	-2,147,483,648 to 2,147,483,647 (2 bytes: -32768 to 32767)
Unsigned integer numbers	unsigned int	4 bytes	0 to 4,294,967,295
Long integer numbers	long	4 bytes	-2,147,483,648 to 2,147,483,647
Unsigned long integer numbers	unsigned long	4 bytes	0 to 4,294,967,295
Single-precision floating-point numbers (6 digits of precision)	float	4 bytes	-3.4e+38 to 3.4e+38
Double-precision floating-point numbers (15 digits of precision)	double	8 bytes	-1.797693e+308 to 1.797693e+308
?	long long	8 bytes	?
?	unsigned long long	8 bytes	?
?	long double	12 bytes	?
Typeless	void	1 byte	N/A
Enumerated data	enum	4 bytes	?

Size varies from system to system!

sizeof (type_name) should be used.

Built-in Data Types in C

Data type	printf conversion specification	scanf conversion specification
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
unsigned short	%hu	%hu
short	%hd	%hd
char	%c	%c

Fig. 5.5. Promotion hierarchy for data types ([1], p. 150)

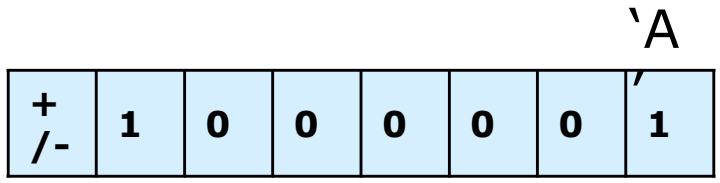
Built-in Data Types in C

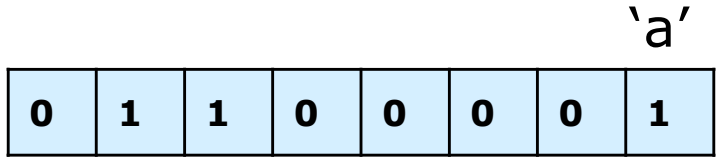
- Representation – Manipulation
  - char, unsigned/signed
  - short,
  - int, unsigned/signed
  - long
  - float
  - double
  - long double
  - …

char < short < int < long < float < double < long double

Built-in Data Types in C
char

- Type name: char
- Memory allocation: sizeof(char) = 1 byte
  - The smallest addressable unit in memory
  - Enough to store a single ASCII character
- Signed range: -128..127
  - 1 bit for sign (0 = +; 1 = -)
- Unsigned range: 0..255
  - 8 bits for value



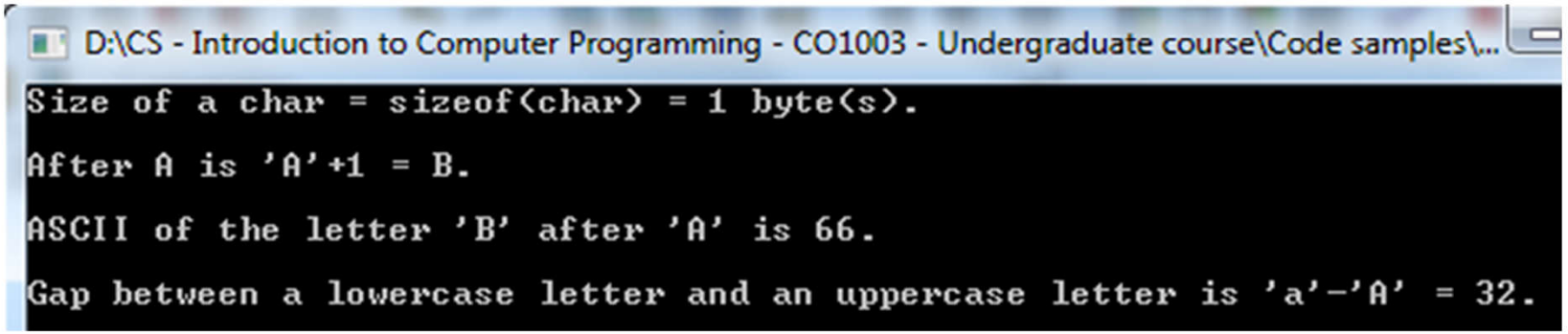


ASCII = American Standard Code for Information Interchange

Built-in Data Types in C
char

- Values can be used as numbers instead of characters.
  - Addition, Subtraction, Comparison

```c
#include <stdio.h>

void main() {

    //char type

    printf("Size of a char = sizeof(char) = %d byte(s).\n\n", sizeof(char));

    printf("After A is \'A\'+1 = %c.\n\n", 'A' + 1);

    printf("ASCII of the letter \'%c\' after \'A\' is %d.\n\n", 'A' + 1, 'A' + 1);

    printf("Gap between a lowercase letter and an uppercase letter is \'a\'-\'A\' = %d.\n\n", 'a'-'A');

}
```



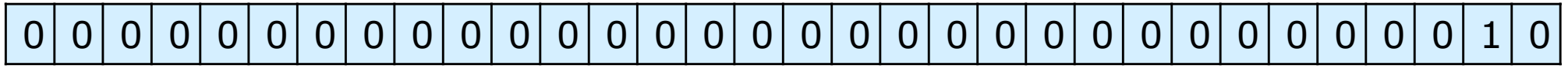
Built-in Data Types in C

int

- Type name: int
- Memory allocation: sizeof(int) = 4 bytes
  - Depend on the implementation: 2 bytes
- Range: $-2^{31}..(2^{31}-1)$

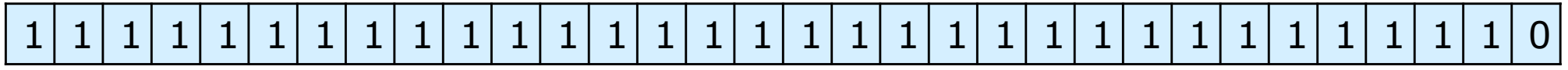
  $= -2{,}147{,}483{,}648..2{,}147{,}483{,}648$
  - 1 bit for sign (0 = +; 1 = -)
  - (4x8 – 1) = 31 bits for value
    - Non-negative values: standard bit representation

2



    - Negative values: 2's complement = $2^{32} - x$

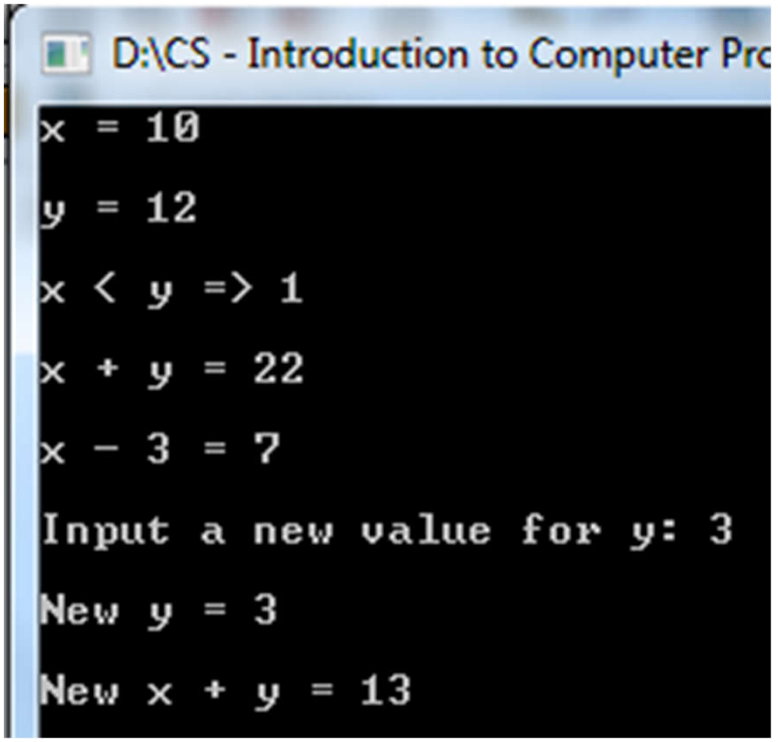
-2



Built-in Data Types in C
int

```
void main() {

    int x;
    int y;

    x = 10;
    y = 12;

    printf("x = %d\n\n", x);
    printf("y = %d\n\n", y);

    printf("x < y => %d\n\n", x < y);

    printf("x + y = %d\n\n", x + y);
    printf("x - 3 = %d\n\n", x - 3);

    printf("Input a new value for y: ");
    scanf("%d", &y);

    printf("\nNew y = %d\n\n", y);
    printf("New x + y = %d\n\n", x + y);
}
```

- Values can be used with the operators:
  - Arithmetic (+, -, *, /, …)
  - Relational (comparison)
  - …



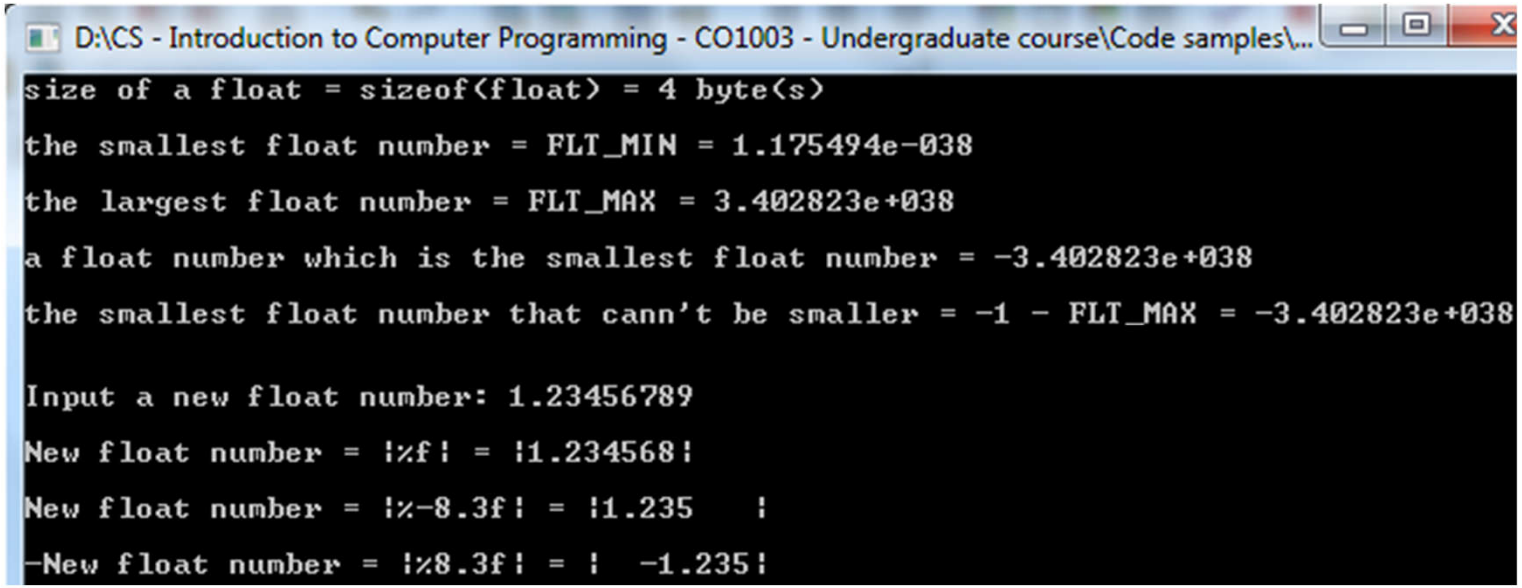
Built-in Data Types in C
float

- Type name: float
- Memory allocation: sizeof(float) = 4 bytes
- Range: -3.4e+38 to 3.4e+38
  - s bit (1 bit) for sign (0 = +; 1 = -)
  - m bits for exponent
  - f bits for fraction

Floating-point number = $(-1)^s * f * 2^m$

- Real numbers are approximately represented.
- Values can be used with the operators:
  - Arithmetic (+, -, *, /, …), Relational (comparison),…
  - Be careful with equality comparison!

Built-in Data Types in C
float

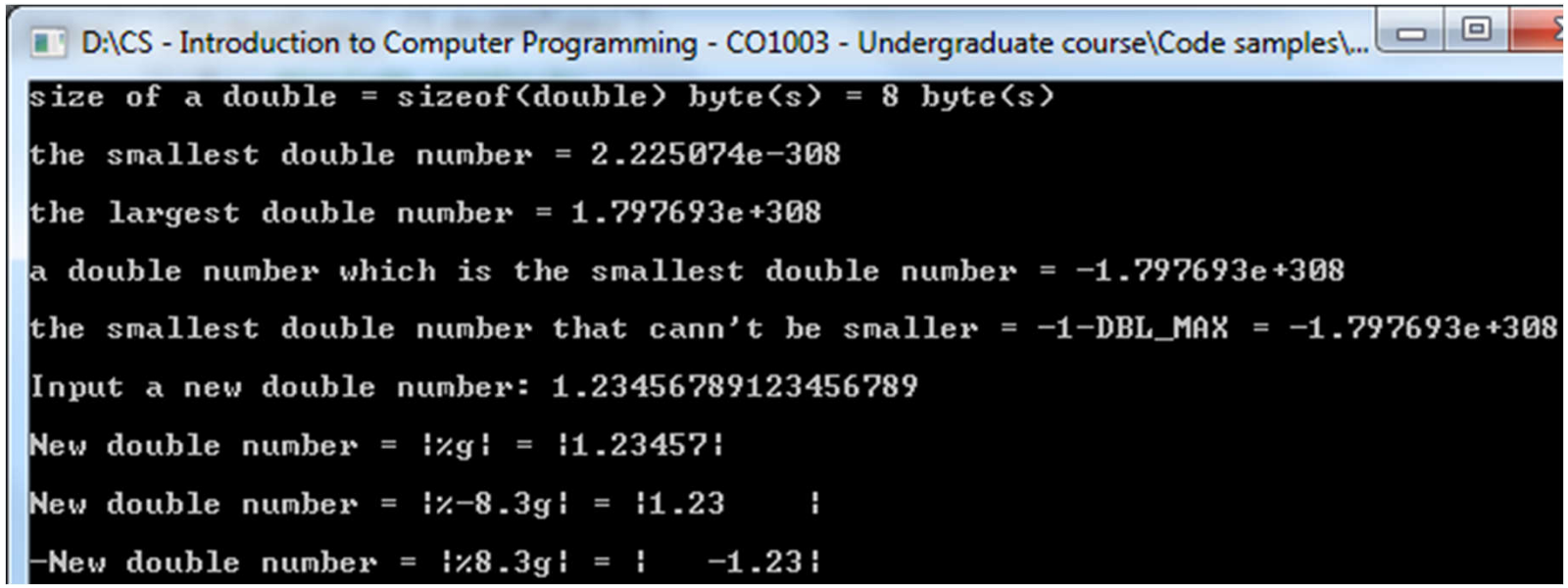



```c
#include <stdio.h>
#include <float.h>

void main() {

    float aFloat = -FLT_MAX;

    //float type
    printf("size of a float = sizeof(float) = %d byte(s)\n\n", sizeof(float));

    printf("the smallest float number = FLT_MIN = %e\n\n", FLT_MIN);
    printf("the largest float number = FLT_MAX = %e\n\n", FLT_MAX);

    printf("a float number which is the smallest float number = %e\n\n", aFloat);
    printf("the smallest float number that cann't be smaller = -1 - FLT_MAX = %e\n\n", -1 - FLT_MAX);

    printf("Input a new float number: ");
    scanf("%f", &aFloat);

    printf("\nNew float number = |%%f| = |%f|\n\n", aFloat);
    printf("New float number = |%%-8.3f| = |%-8.3f|\n\n", aFloat);
    printf("-New float number = |%%8.3f| = |%8.3f|\n\n", -aFloat);
}
```

Built-in Data Types in C
double

- Type name: double
- Memory allocation: sizeof(double) = 8 bytes
- Range: -1.7e+308 to 1.7e+308
  - s bit (1 bit) for sign (0 = +; 1 = -)
  - m bits for exponent
  - f bits for fraction
- Real numbers are approximately represented.
- Values can be used with the operators:
  - Arithmetic (+, -, *, /, …), Relational (comparison),…
  - Be careful with equality comparison!

Built-in Data Types in C
double




```c
#include <stdio.h>
#include <float.h>

void main() {

    double aDouble = -DBL_MAX;

    //double type
    printf("size of a double = sizeof(double) byte(s) = %d byte(s)\n\n", sizeof(double));

    printf("the smallest double number = %e\n\n", DBL_MIN);

    printf("the largest double number = %e\n\n", DBL_MAX);

    printf("a double number which is the smallest double number = %e\n\n", aDouble);
    printf("the smallest double number that cann't be smaller = -1-DBL_MAX = %e\n\n", -1 - DBL_MAX);

    printf("Input a new double number: ");
    scanf("%lf", &aDouble);

    printf("\nNew double number = |%%g| = |%g|\n\n", aDouble);
    printf("New double number = |%%-8.3g| = |%-8.3g|\n\n", aDouble);
    printf("-New double number = |%%8.3g| = |%8.3g|\n\n", -aDouble);
}
```

enum Data Type

- Enumerations, introduced by the keyword **enum**, are unique types with values ranging over a set of named constants called enumerators.
- The identifiers in an enumerator list are declared as constants of type **int**, and may appear wherever constants are required.
- Enumerator names in the same scope must all be distinct from each other and from ordinary variable names, but the values need not be distinct.

enum Data Type

enum-specifier:
  enum *identifier*$_{opt}$ { *enumerator-list* }
  enum *identifier*

enumerator-list:
  enumerator
  enumerator-list , *enumerator*

enumerator:
  identifier
  identifier = *constant-expression*

[2], pp. 215

Type definition:

enum bool {NO, YES};

enum {NO, YES};

enum {NO = 'N', YES = 'Y'}

enum bool {FALSE = 0, TRUE = 1}

Variable definition:

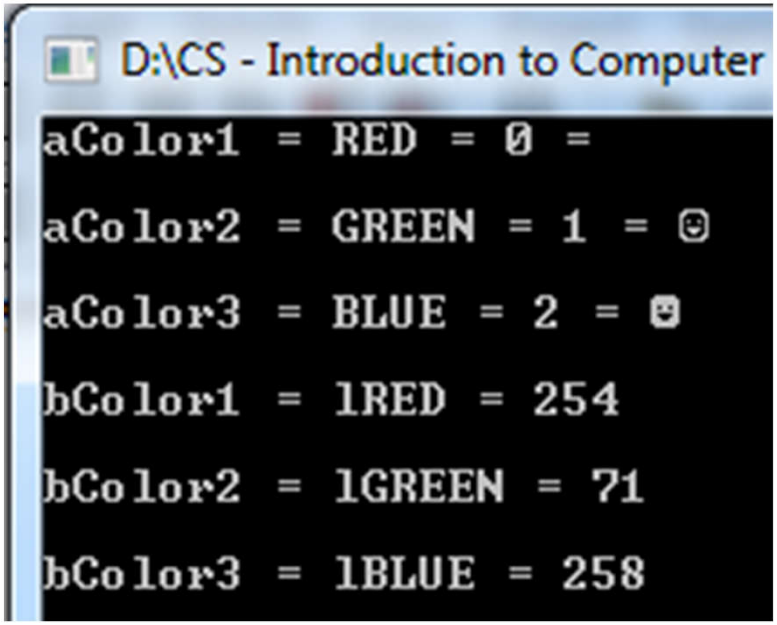
enum bool isHappy = YES;

int isTrue = TRUE;

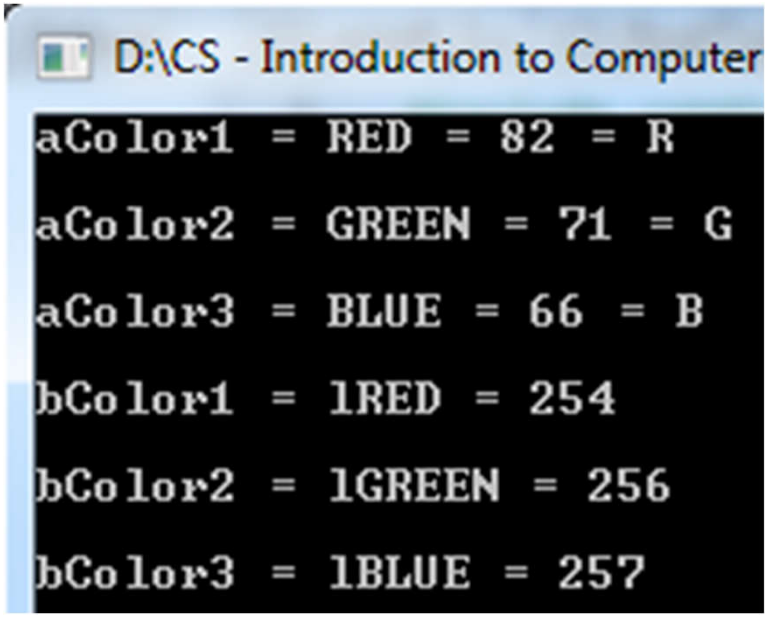
enum Data Type

- If no enumerations with = appear, then the values of the corresponding constants begin at 0 and increase by 1 as the declaration is read from left to right.
- An enumerator with = gives the associated identifier the value specified; subsequent identifiers continue the progression from the assigned value.

```c
#include <stdio.h>

enum {RED = 'R', GREEN = 'G', BLUE = 'B'};

void main() {

    enum color {lRED = 254, lGREEN = 256, lBLUE = 258};
    enum color bColor1 = lRED, bColor2 = GREEN, bColor3 = lBLUE;

    enum {RED, GREEN, BLUE};

    int aColor1 = RED, aColor2 = GREEN, aColor3 = BLUE;

    printf("aColor1 = RED = %d = %c\n\n", aColor1, aColor1);
    printf("aColor2 = GREEN = %d = %c\n\n", aColor2, aColor2);
    printf("aColor3 = BLUE = %d = %c\n\n", aColor3, aColor3);

    printf("bColor1 = lRED = %d\n\n", bColor1);
    printf("bColor2 = lGREEN = %d\n\n", bColor2);
    printf("bColor3 = lBLUE = %d\n\n", bColor3);
}
```



```c
#include <stdio.h>

enum {RED = 'R', GREEN = 'G', BLUE = 'B'};

void main() {

    enum color {lRED = 254, lGREEN = 256, lBLUE};
    enum color bColor1 = lRED, bColor2 = lGREEN, bColor3 = lBLUE;

    int aColor1 = RED, aColor2 = GREEN, aColor3 = BLUE;

    printf("aColor1 = RED = %d = %c\n\n", aColor1, aColor1);
    printf("aColor2 = GREEN = %d = %c\n\n", aColor2, aColor2);
    printf("aColor3 = BLUE = %d = %c\n\n", aColor3, aColor3);

    printf("bColor1 = lRED = %d\n\n", bColor1);
    printf("bColor2 = lGREEN = %d\n\n", bColor2);
    printf("bColor3 = lBLUE = %d\n\n", bColor3);
}
```



struct Data Type

- A structure is a collection of one or more variables grouped together under a single name for convenient handling.
- The keyword **struct** introduces a structure declaration, which is a list of declarations enclosed in braces, to define a derived type.
- An optional name called a *structure tag* may follow the word struct.
- The variables named in a structure are called *members*.
- Structures can be nested.

struct Data Type

- A structure declaration that is not followed by a list of variables reserves no storage; merely describes a template or shape of a structure.
- Size of a **struct** data type is a total sum of all the sizes of the types of its members.
- An automatic structure may also be initialized by assignment or by calling a function that returns a structure of the right type.
- A member of a particular structure is referred to in an expression by a member operator:

structure-name.member

struct Data Type

```
struct {
        int x;
        int y;
};
```

```
struct point {
        int x;
        int y;
};
```

```
struct student {
        char        IdStudent[10];
        char        Name[50];
        int         IdMajor;
        int         EntranceYear;
        struct point  Location;
};
```

Structure declaration

```
struct {
        int x;
        int y;
} aPoint1, aPoint2;
```

```
struct point {
        int x;
        int y;
} aPoint4, aPoint5;
```

```
struct student aStudent;
```

```
struct {
        int x;
        int y;
} aPoint3 = {0, 0};
```

```
struct point aPoint6 = {0, 0};
```

Variable declaration

```
aPoint3.x
```

```
aPoint6.y
```

```
aStudent.EntranceYear
aStudent.Location.x
```

Member access

struct Data Type

```
struct {                    struct point {          struct student {
        int x;                      int x;                  char        IdStudent[10];
        int y;                      int y;                  char        Name[50];
};                          };                              int         IdMajor;
                                                            int         EntranceYear;
                                                            struct point Location;
                                                    };
```

tag

member

Structure declaration

```
struct {                    struct point {          struct student aStudent;
        int x;                      int x;
        int y;                      int y;
} aPoint1, aPoint2;         } aPoint4, aPoint5;

struct {                    struct point aPoint6 = {0, 0};
        int x;
        int y;
} aPoint3 = {0, 0};
```

initialization

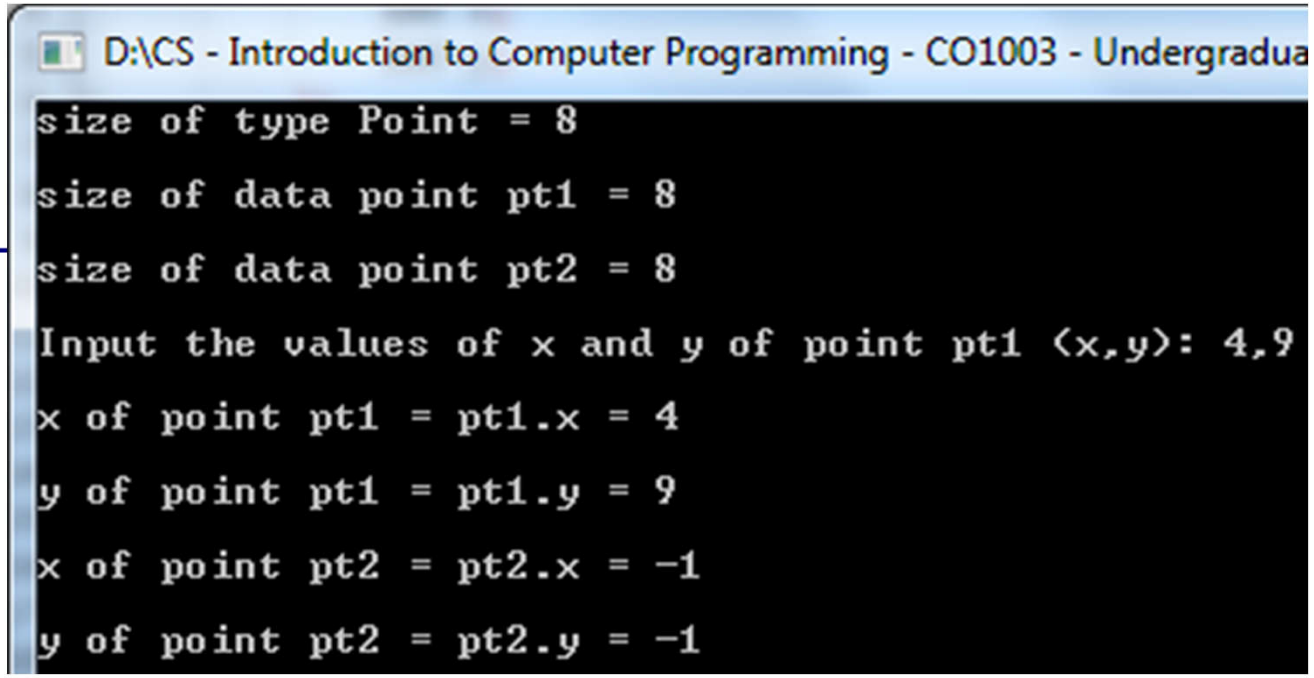
Variable declaration

```
aPoint3.x                   aPoint6.y               aStudent.EntranceYear
                                                    aStudent.Location.x
```

Member access

struct Data Type

```
#include <stdio.h>

struct Point {
        int x;
        int y;
    };

void main() {

    struct Point pt1, pt2 = {-1, -1};

    printf("size of type Point = %d\n\n", sizeof(struct Point));

    printf("size of data point pt1 = %d\n\n", sizeof(pt1));

    printf("size of data point pt2 = %d\n\n", sizeof(pt2));

    printf("Input the values of x and y of point pt1 (x,y): ");

    scanf("%d,%d", &pt1.x, &pt1.y);

    printf("\nx of point pt1 = pt1.x = %d\n\n", pt1.x);
    printf("y of point pt1 = pt1.y = %d\n\n", pt1.y);

    printf("x of point pt2 = pt2.x = %d\n\n", pt2.x);
    printf("y of point pt2 = pt2.y = %d\n\n", pt2.y);
}
```



New Type Name Definition

- A mechanism for creating synonyms (or aliases) for previously defined data types

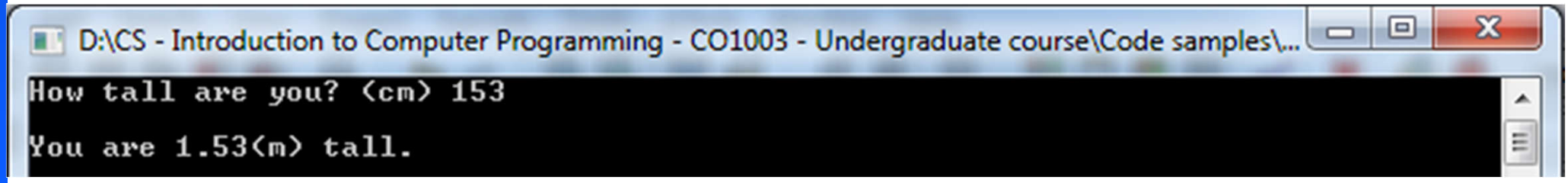
  typedef *old_type_name new_type_name*;

- Useful for the following cases:
  - Shorten the existing type names (e.g. struct)
  - Meaningful specific-purpose type names for the existing type names
  - Self-documenting, more portable
- No new data type is created with typedef.

New Type Definition

- Unit converter: receive a value in centimeter and print it in meter.

```c
#include <stdio.h>

typedef unsigned char cm;

void main() {
    cm height;

    printf("How tall are you? (cm) ");
    scanf("%d", &height);

    printf("\nYou are %2.2f(m) tall.\n\n", height/100.0);
}
```



Variables and Variable Declaration

- A variable is a location in memory where a value can be stored for use by a program.
- All variables must be defined with a name and a data type before they are used in a program.
- Each variable has a value processed in a program within a scope to be referred.
- Variable classification
  - Global variables
  - Local variables

Variables and Variable Declaration

- Variable names actually correspond to locations in the computer's memory.
- A variable name in C is any valid identifier.
  - a series of characters consisting of letters, digits and underscores (_) that does not begin with a digit
    - •: _minNumber, global_counter, i1, i2
    - X: min#, 123Iteration, ThisVar., @g_Variable
  - of any length, but only the first 31 characters are required to be recognized by C standard compilers
  - not a keyword in C
- C is case sensitive.
  - Global_counter *is different from* global_counter.

Variables and Variable Declaration

- A data type of a variable is specified in its declaration.

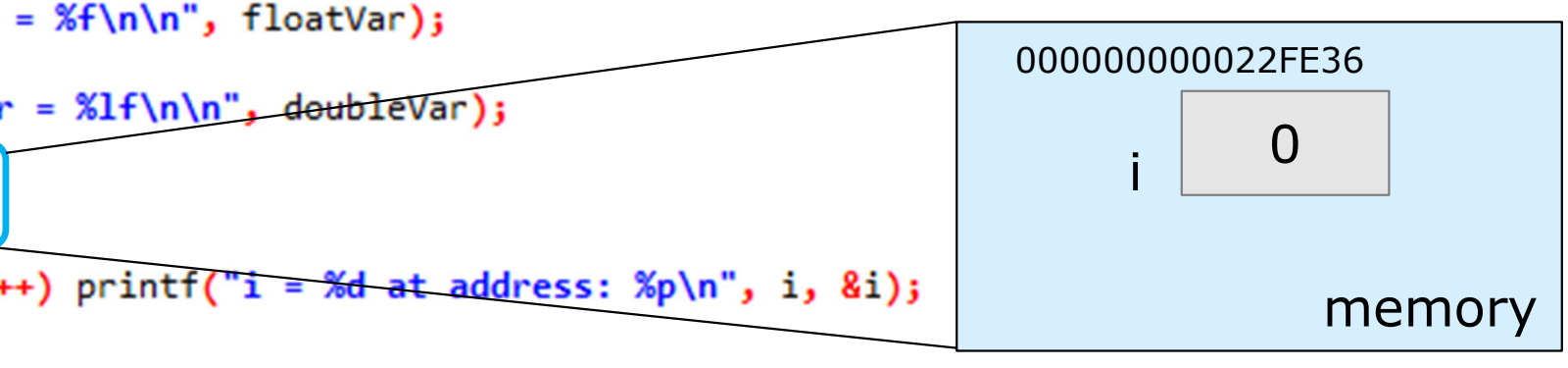
  type_name *variable_name*_1 [= *initial_value*_1]
  [, *variable_name*_2 [= *initial_value*_2]]
  … [, *variable_name*_n [= *initial_value*_n]];

- A compiler allocates memory for declared variables up to the data type and its storage class at run time.
- A compiler associates *variable_name* to the allocated memory.
- A compiler sets *initial_value* to the content of the allocated memory if *initial_value* exists.

Variable Declaration

```c
#include <stdio.h>

void main() {

    printf("This is a test for variable declarations.\n\n");

    char charVar;

    int intVar1 = 97, intVar2;

    float floatVar;

    double doubleVar;

    charVar = 65;

    intVar2 = charVar + 32;

    floatVar = 1.0/3;

    doubleVar = floatVar*charVar;

    printf("charVar = %c\n\n", charVar);

    printf("intVar1 = %d for a character = %c\n\n", intVar1, intVar2);

    printf("floatVar = %f\n\n", floatVar);

    printf("doubleVar = %lf\n\n", doubleVar);

    short i=0;

    for (i=0; i<3; i++) printf("i = %d at address: %p\n", i, &i);
}
```

```
D:\CS - Introduction to Computer Programming - CO1003
This is a test for variable declarations.

charVar = A

intVar1 = 97 for a character = a

floatVar = 0.333333

doubleVar = 21.666668

i = 0 at address: 000000000022FE36
i = 1 at address: 000000000022FE36
i = 2 at address: 000000000022FE36
```



Variables and Variable Declaration

- Global variables
  - Declared outside of all the functions
  - Globally accessed inside of any functions
  - Hold values throughout the lifetime of a program
  - Initialized by the system once defined
- Local variables
  - Declared and locally accessed inside a function (main, others) or block between the brackets
  - Should be defined immediately after the left bracket that begins the body of a function/block
  - Exist only as long as the function or block where the variables are declared is still executing
  - Not initialized by the system once defined

```c
#include <stdio.h>

void main() {
    char aChar;
    int anInt1, anInt2 = 9;
    float aFloat1, aFloat2, aFloat3;
    double aDouble;
    char aString[10];
    long double *pLDouble;

    printf("aChar = %c\n\n", aChar);
    printf("aString = %s\n\n", aString);
    printf("anInt1 = %d\n\n", anInt1);
    printf("anInt2 = %d\n\n", anInt2);
    printf("aFloat1 = %f\n\n", aFloat1);
    printf("aFloat2 = %f\n\n", aFloat2);
    printf("aFloat3 = %f\n\n", aFloat3);
    printf("aDouble = %lf\n\n", aDouble);
    printf("pLDouble = %p\n\n", pLDouble);

    printf("A new value for anInt2 : ");
    scanf("%d", &anInt2);

    printf("\nA new value for aFloat1 : ");
    scanf("%f", &aFloat1);

    printf("\nA new value for aDouble : ");
    scanf("%lf", &aDouble);

    pLDouble = &aDouble;

    printf("\nAgain, aChar = |%c|\n\n", aChar);
    printf("Again, aString = |%s|\n\n", aString);
    printf("Again, anInt1 = |%d|\n\n", anInt1);
    printf("Again, anInt2 = |%d|\n\n", anInt2);
    printf("Again, aFloat1 = |%f|\n\n", aFloat1);
    printf("Again, aFloat2 = |%f|\n\n", aFloat2);
    printf("Again, aFloat3 = |%f|\n\n", aFloat3);
    printf("Again, aDouble = |%lf| at address: |%p|\n\n", aDouble, &aDouble);
    printf("Again, pLDouble = |%p|\n\n", pLDouble);
}
```

```
D:\CS - Introduction to Computer Programming
aChar =
aString =          w
anInt1 = 1
anInt2 = 9
aFloat1 = 0.000000
aFloat2 = 0.000000
aFloat3 = 0.000000
aDouble = 0.000000
pLDouble = 0000000000000077
A new value for anInt2 : _
```

```
D:\CS - Introduction to Computer Programming - CO1003 - Undergraduate course\
A new value for anInt2 : 12
A new value for aFloat1 : 3.45678912
A new value for aDouble : 4.56789123456789123456789
Again, aChar = ! !
Again, aString = !         w!
Again, anInt1 = !1!
Again, anInt2 = !12!
Again, aFloat1 = !3.456789!
Again, aFloat2 = !0.000000!
Again, aFloat3 = !0.000000!
Again, aDouble = !4.567891! at address: !000000000022FE28!
Again, pLDouble = !000000000022FE28!
```

ariable
ration.
bles
ues in
me.

```c
#include <stdio.h>

char gChar1 = 'a', gChar2;
int *gPointer;
int gInt, anInt = 10;
float gFloat;
double gDouble;

void main() {

    char lChar;
    int lInt = -1, anInt;
    float lFloat;
    double lDouble;

    printf("This is a test for global and local variable declarations.\n\n");

    printf("gChar1 = |%c|\n\n", gChar1);

    printf("gChar2 = |%c|\n\n", gChar2);

    printf("gPointer = |%p|\n\n", gPointer);

    printf("gInt = %d\n\n", gInt);

    printf("anInt = %d\n\n", anInt);

    printf("gFloat = %f\n\n", gFloat);

    printf("gDouble = %lf\n\n", gDouble);

    printf("lChar = |%c|\n\n", lChar);

    printf("anInt = %d\n\n", anInt);

    printf("lInt = %d\n\n", lInt);

    printf("lFloat = %f\n\n", lFloat);

    printf("lDouble = %lf\n\n", lDouble);
}
```

?

Initialized values for global variables:

- char '\0' (i.e. NULL)
- int 0
- float 0
- double 0
- pointer NULL

All the bytes in memory are filled with zeros.

```
D:\CS - Introduction to Computer Programming - CO1003 - Undergraduate course\
This is a test for global and local variable declarations.

gChar1 = |a|

gChar2 = | |

gPointer = |0000000000000000|

gInt = 0

anInt = 1

gFloat = 0.000000

gDouble = 0.000000

lChar = | |

anInt = 1

lInt = -1

lFloat = 0.000000

lDouble = 0.000000
```

Variables and Variable Declaration

- The *scope* of a name is the part of the program within which the name can be used.
- A scope of a variable name is a region of the program (function() {…}, block {..}) where a variable can have its existence. Beyond that, it cannot be accessed.
- For a variable declared at the beginning of a function, the scope is the function where the name is declared.
  - Local variables of the same name in different functions are unrelated.
  - The same is true for the parameters of the function, which are in fact local variables.
- The scope of an external variable or a function lasts from the point at which it is declared to the end of the file being compiled.

Variables and Variable Declaration

```c
#include <stdio.h>

char aChar = 'a', gChar = 'A';

void main() {

    printf("1.1. aChar = %c\n\n", aChar);
    printf("1.1. gChar = %c\n\n", gChar);
    printf("1.1. gChar2 = %c\n\n", gChar2);

    char aChar = 'b', bChar = 'B';

    printf("1.2. aChar = %c\n\n", aChar);
    printf("1.2. gChar = %c\n\n", gChar);

    printf("1.2. aChar = %c\n\n", aChar);
    printf("1.2. bChar = %c\n\n", bChar);

    {
        printf("1.3.1. aChar = %c\n\n", aChar);
        printf("1.3.1. gChar = %c\n\n", gChar);

        printf("1.3.1. aChar = %c\n\n", aChar);
        printf("1.3.1. bChar = %c\n\n", bChar);

        char aChar = 'c', cChar = 'C';

        printf("1.3.2. aChar = %c\n\n", aChar);
        printf("1.3.2. gChar = %c\n\n", gChar);

        printf("1.3.2. aChar = %c\n\n", aChar);
        printf("1.3.2. bChar = %c\n\n", bChar);

        printf("1.3.2. aChar = %c\n\n", aChar);
        printf("1.3.2. cChar = %c\n\n", cChar);
    }
}

char gChar2 = 'Z';
```

gChar2 is unable to be accessed in the **main** function due to its improper declaration place!

```c
#include <stdio.h>

char aChar = 'a', gChar = 'A';

void main() {

    printf("1.1. aChar = %c\n\n", aChar);
    printf("1.1. gChar = %c\n\n", gChar);

    char aChar = 'b', bChar = 'B';

    printf("1.2. aChar = %c\n\n", aChar);
    printf("1.2. gChar = %c\n\n", gChar);

    printf("1.2. aChar = %c\n\n", aChar);
    printf("1.2. bChar = %c\n\n", bChar);

    {
        printf("1.3.1. aChar = %c\n\n", aChar);
        printf("1.3.1. gChar = %c\n\n", gChar);

        printf("1.3.1. aChar = %c\n\n", aChar);
        printf("1.3.1. bChar = %c\n\n", bChar);

        char aChar = 'c', cChar = 'C';

        printf("1.3.2. aChar = %c\n\n", aChar);
        printf("1.3.2. gChar = %c\n\n", gChar);

        printf("1.3.2. aChar = %c\n\n", aChar);
        printf("1.3.2. bChar = %c\n\n", bChar);

        printf("1.3.2. aChar = %c\n\n", aChar);
        printf("1.3.2. cChar = %c\n\n", cChar);
    }
}
```

gChar

bChar

cChar

D:\CS - Introduction to

```
1.1. aChar = a

1.1. gChar = A

1.2. aChar = b

1.2. gChar = A

1.2. aChar = b

1.2. bChar = B

1.3.1. aChar = b

1.3.1. gChar = A

1.3.1. aChar = b

1.3.1. bChar = B

1.3.2. aChar = c

1.3.2. gChar = A

1.3.2. aChar = c

1.3.2. bChar = B

1.3.2. aChar = c

1.3.2. cChar = C
```

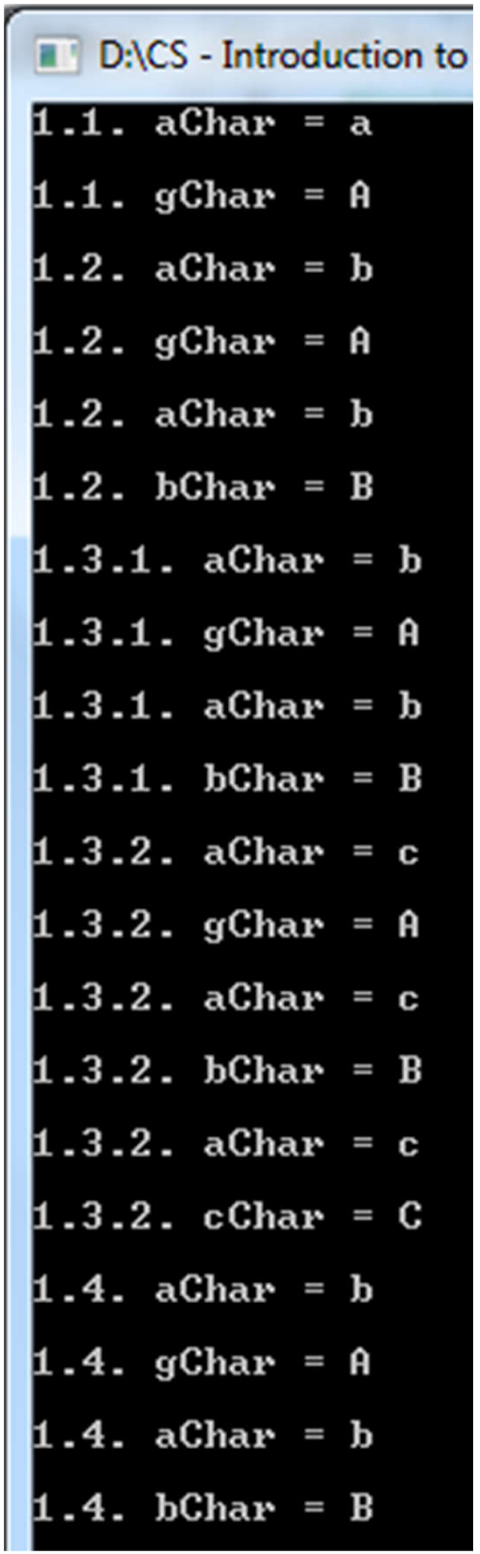
```c
#include <stdio.h>

char aChar = 'a', gChar = 'A';

void main() {
    printf("1.1. aChar = %c\n\n", aChar);
    printf("1.1. gChar = %c\n\n", gChar);

    char aChar = 'b', bChar = 'B';

    printf("1.2. aChar = %c\n\n", aChar);
    printf("1.2. gChar = %c\n\n", gChar);

    printf("1.2. aChar = %c\n\n", aChar);
    printf("1.2. bChar = %c\n\n", bChar);

    {
        printf("1.3.1. aChar = %c\n\n", aChar);
        printf("1.3.1. gChar = %c\n\n", gChar);

        printf("1.3.1. aChar = %c\n\n", aChar);
        printf("1.3.1. bChar = %c\n\n", bChar);

        char aChar = 'c', cChar = 'C';

        printf("1.3.2. aChar = %c\n\n", aChar);
        printf("1.3.2. gChar = %c\n\n", gChar);

        printf("1.3.2. aChar = %c\n\n", aChar);
        printf("1.3.2. bChar = %c\n\n", bChar);

        printf("1.3.2. aChar = %c\n\n", aChar);
        printf("1.3.2. cChar = %c\n\n", cChar);
    }

    printf("1.4. aChar = %c\n\n", aChar);
    printf("1.4. gChar = %c\n\n", gChar);

    printf("1.4. aChar = %c\n\n", aChar);
    printf("1.4. bChar = %c\n\n", bChar);
}
```

The most "local" variables will take precedence over the others.

How to refer to them?

Naming!

Which aChar is printed?



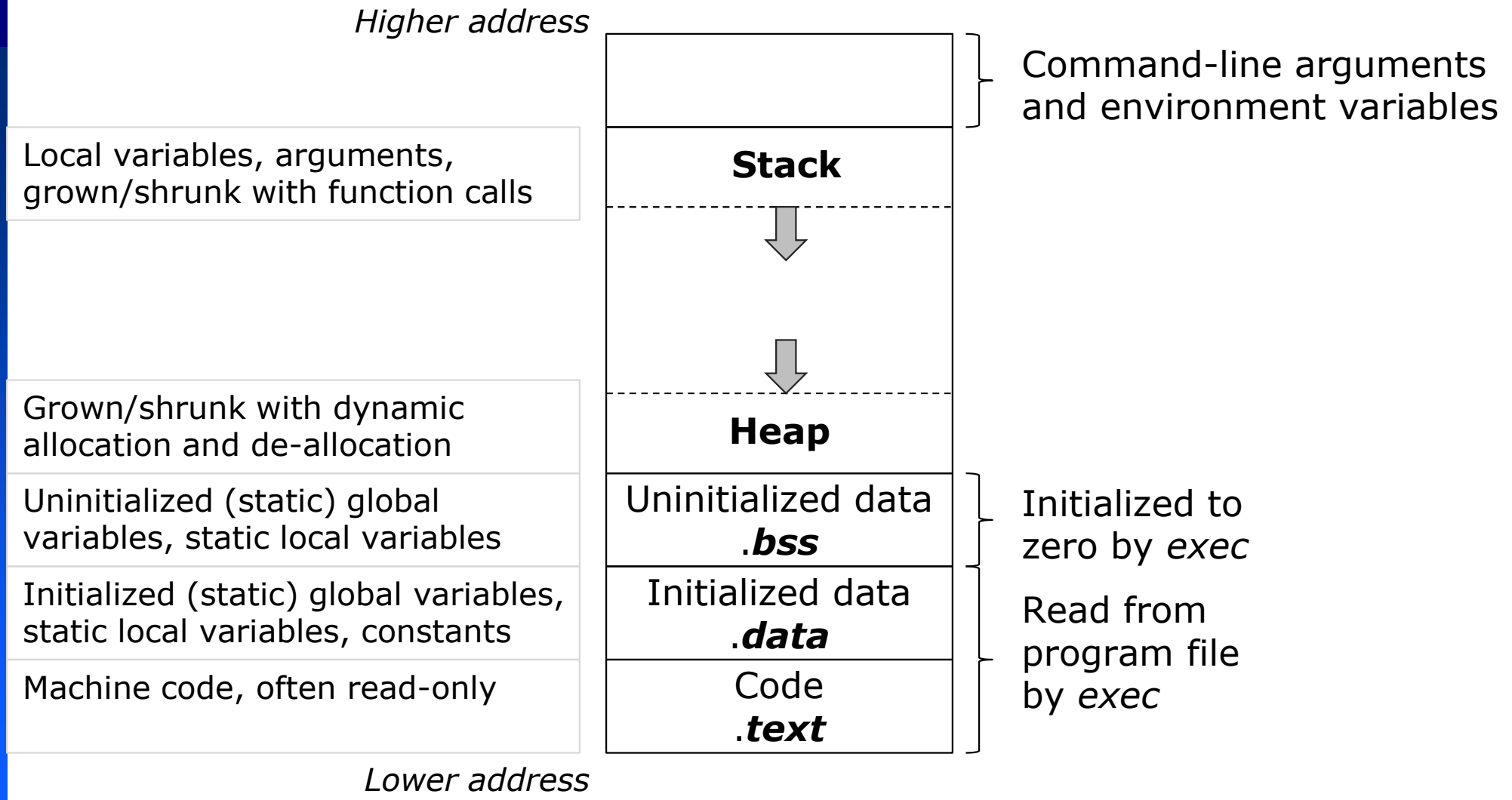
Variables and Variable Declaration

- Where are variable values stored?
  - Storage of data in variables and arrays is temporary in (registers and) RAM. That is such data is lost when a program terminates.
  - Storage classes for different distinct memory areas

Variable Type	Keyword	Storage class	Scope
Local variables	auto (redundant)	Automatic (default)	Declared function/block
Register local variables	register	Register if possible. If not, automatic	Declared function/block
Static local variables	static	Static	Declared function/block
Global variables		Static	Program
Global variables	extern	Static	Program
Static global variables	static	Static	File
Variables with dynamically allocated memory	malloc(), calloc(), free(), realloc()	Dynamic	Variable's scope: local, global

Variables and Variable Declaration

- Memory layout of a C program



bss = block started by symbol, better save space

Variables and Variable Declaration

- Memory areas in C
  - Constant data area
    - Read-only memory for string constants and other data whose values are known at compile time, existing for the lifetime of the program
  - Static area
    - Read-writable memory for extern/static variables existing for the lifetime of the program
  - Stack area
    - Read-writable last-in-first-out memory for a local variable, existing from the point where/when the variable is defined and released immediately as the variable goes out-of-scope
  - Heap area
    - Memory dynamically allocated explicitly by programmers

Constant Definition

- Constants refer to fixed values that the program may not alter during its execution.
- Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, a string literal, or enumeration constants*.
- Constants are treated just like regular variables except that their values cannot be modified after their definition.

Constant Definition

- Defined by:
  - Using #define preprocessor

    #**define** identifier value

  - Using const keyword

    const type_name variable_name = value;

  - Using enum type
    - Integer constants represented by identifiers

    enum [type_name] {identifier [= value], …};

Constant Definition

- Defined by:
  - Using #define preprocessor

```
#define MAX 50
```

  - Using const keyword

```
const short MAX = 50;
```

  - Using enum type
    - Integer constants represented by identifiers

```
enum min_max {min=0, MAX=50};
enum {min=0, MAX=50};
```

Expressions

- An expression is simply a valid combination of operators and their operands (variables, constants, …)
- Each expression has a value and a type.
- Primary expressions
  - Identifier (variable, function, symbolic constant)
  - Constant
    - involves only constants
    - evaluated at during compilation rather than run-time
    - used in any place that a constant can occur
  - String literal
  - (*expression*)
    - type and value are identical to the enclosed *expression*

Expressions

- Expression values are determined by the operations in a certain order based on precedence and associativity of each operator.
- Expressions are grouped by the operators
  - Arithmetic expressions
  - Logical expressions
  - Relational expressions
  - Bitwise expressions
  - Assignment expressions
  - Conditional expressions
  - …

Expressions

- Valid expressions
  - 12 + intVar – sqrt(23)*2
  - “This is an expression.”
  - ‘A’ + 32
  - (12/4)%2 + 8 – floatVar
  - …
- Invalid expressions
  - ‘ABC’
  - 0”Wrong”1
  - intVar*2# - 10
  - …

Operators

- Arithmetic operators
- Relational operators
- Logic operators
- Bitwise operators
- Comma operator
- Assignment operators
- Type casting operator
- Other operators

Arithmetic Operators

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.

+, -, *, /	All numeric types. Integer division yields integer results.
%	Integer types (including enum) only.

Increment and Decrement Operators

- Increment and decrement operators: ++, --

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

int x=4, y=5;

++x – y = ?, x = ?, y = ?

x = x + 1 = 5, increment
5 – 5 = 0, use • pre.
y = 5

int x=4, y=5;

x++ – y = ?, x = ?, y = ?

4 – 5 = -1, use
x = x + 1 = 5, increment • post.
y = 5

Relational Operators

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
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 2.12 | Equality and relational operators.

All numeric types.

!!! EQUALITY with ==

Logic Operators

- Logic operators: &&, ||, !

corresponding to AND, OR, NOT

- The C language has no boolean data type.
- Zero (0) is used to represent FALSE.
- Non-zero (≠0) is used to represent TRUE.

1 && 2 • 1	1 || 2 • 1	!1 • 0
1 && 1 • 1	1 || 1 • 1	!2 • 0
1 && 0 • 0	1 || 0 • 1	!-2 • 0
0 && 0 • 0	0 || 0 • 0	!0 • 1

Bitwise Operators

Operator		Description
&	bitwise AND	The bits in the result are set to 1 if the corresponding bits in the two operands are both 1.
|	bitwise inclusive OR	The bits in the result are set to 1 if at least one of the corresponding bits in the two operands is 1.
^	bitwise exclusive OR	The bits in the result are set to 1 if exactly one of the corresponding bits in the two operands is 1.
<<	left shift	Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from the right with 0 bits.
>>	right shift	Shifts the bits of the first operand right by the number of bits specified by the second operand; the method of filling from the left is machine dependent.
~	one's complement	All 0 bits are set to 1 and all 1 bits are set to 0.

Fig. 10.6 | Bitwise operators.

The binary bitwise operators are used to manipulate the bits of integral operands (char, short, int and long; both signed and unsigned).

Unsigned integers are normally used with the bitwise operators.

Bitwise data manipulations are machine dependent.

Assignment Operators

- Assignment operator: =
- Assignment shorthand operators:

+=, -=	Increment or decrement by RHS
*=, /=	Multiply or divide by RHS
%=	Mod by RHS
>>=	Bitwise right shift by RHS (divide by power of 2)
<<=	Bitwise left shift RHS (multiply by power of 2)
&=, |=, ^=	Bitwise and, or, xor by RHS

RHS = right hand side

Assignment Operators

- Assignment operator: =
  - copies the value from its right hand side to the variable on its left hand side
  - also acts as an expression which returns the newly assigned value

1. Copy:

variable = RHS;

2. return: RHS

- Data type of the variable and data type of RHS must be the same.
- Otherwise, data type of RHS will be casted to data type of the variable.

```
int x=4, y=2;
x = y + 1;
y = (x = x + 10);
x = ? y = ?
Result:
x = 13, y = 13
```

Assignment Operators

- Assignment operator: =

```
int x = 2, y = 3;

float f = 1.5;

char c;

y = f + x*2;

c = y*20 + x + 8*f;

f = (y = c - x*3);

x = f/5.0;
```

x = ? y = ? f = ? c = ?

```
int x = 2, y = 3;

float f = 1.5;

char c;

y = 1.5 + 2*2 = 5.5 = 5;

c = 5*20 + 2 + 8*1.5
  = 100 + 2 + 12.0 = 114.0 = 114;

y = 114 – 2*3 = 108;
f = (y=108) = 108.0;

x = 108.0/5.0 = 21.6 = 21;
```

x = 21? y = 108? f = 108.0? c = 114?

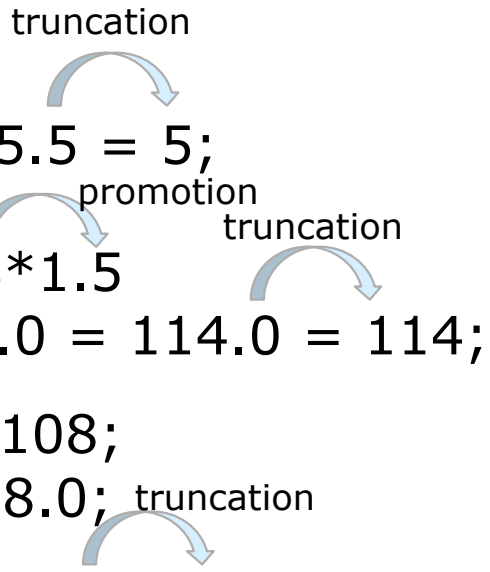
Assignment Operators

- Assignment operator: =

```
int x = 2, y = 3;

float f = 1.5;

char c;

y = f + x*2;

c = y*20 + x + 8*f;

f = (y = c - x*3);

x = f/5.0;
```

x = ? y = ? f = ? c = ?

```
int x = 2, y = 3;

float f = 1.5;

char c;

y = 1.5 + 2*2 = 5.5 = 5;

c = 5*20 + 2 + 8*1.5
  = 100 + 2 + 12.0 = 114.0 = 114;

y = 114 – 2*3 = 108;
f = (y=108) = 108.0;

x = 108.0/5.0 = 21.6 = 21;
```



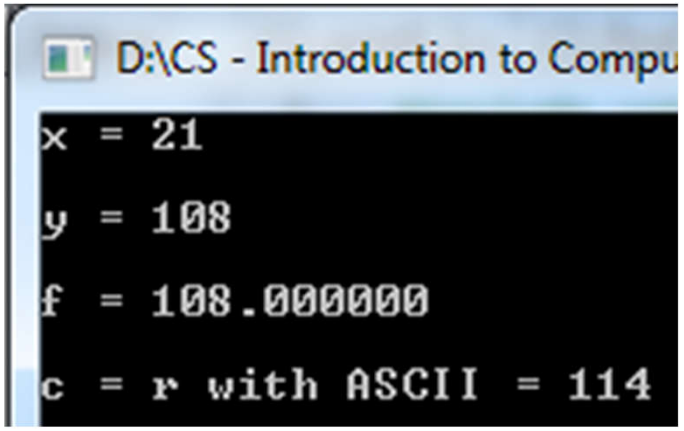
x = 21? y = 108? f = 108.0? c = 114?

Assignment Operators

```
int x = 2, y = 3;

float f = 1.5;

char c;

y = f + x*2;

c = y*20 + x + 8*f;

f = (y = c - x*3);

x = f/5.0;
```

x = ? y = ? f = ? c = ?

```
#include <stdio.h>

void main() {

    int x = 2, y = 3;

    float f = 1.5;

    char c;

    y = f + x*2;

    c = y*20 + x + 8*f;

    f = (y = c - x*3);

    x = f/5.0;

    printf ("x = %d\n\n", x);

    printf("y = %d\n\n", y);

    printf("f = %f\n\n", f);

    printf("c = %c with ASCII = %d\n\n", c, c);

}
```



Assignment Operators

- Assignment shorthand 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.

variable *operator*= RHS;

variable = variable *operator* (RHS);

RHS = right hand side

int x=4, y=5;

x *= y – 2;

x = x * (y-2);

Result:

x = 12;

NOT: x = 18!

Comma Operators

- Comma operator: ,

expression:
assignment-expression
expression , assignment-expression

  - A pair of expressions separated by a comma is evaluated left-to-right, and the value of the left expression is discarded.
  - The type and value of the result are the type and value of the right operand.
  - All side effects from the evaluation of the left-operand are completed before beginning the evaluation of the right operand.

Comma Operators

- Comma operator: ,

expression:
assignment-expression
expression , *assignment-expression*

- The comma operator most often finds use in the ***for*** statement.
- The commas that separate function arguments, variables in declarations, etc., are *not* comma operators, and do not guarantee left to right evaluation.

Comma Operators

- Comma operator: ,

expression:
assignment-expression
expression , assignment-expression

```
int intVar1, intVar2, i;
char charVar1 = 'A', charVar2 = 'B', charVar3;

for (i=1, intVar1=1; i<charVar1 && intVar1<charVar1; i++) {…}
intVar1 = (charVar3 = 'C', intVar2 = 2 + charVar3);
intVar1 = ? intVar2 = ? charVar1 = ? charVar2 = ? charVar3 = ?
```

```
charVar3 = 'C';
intVar2 = 2 + charVar3 = 2 + 'C' = 2 + 67 = 69;
intVar1 = 2 + charVar3 = 69;
charVar1 = 'A';
charVar2 = 'B';
```

Comma Operators

- Comma operator: ,

expression:
assignment-expression
expression , assignment-expression

```
int intVar1, intVar2, i;
char charVar1 = 'A', charVar2 = 'B', charVar3;

for (i=1, intVar1=1; i<charVar1 && intVar1<charVar1; i++) {…}
intVar1 = (charVar3 = 'C', intVar2 = 2 + charVar3);
```

intVar1 = ? intVar2 = ? charVar1 = ? charVar2 = ? charVar3 = ?

```
charVar3 = 'C';
intVar2 = 2 + charVar3 = 2 + 'C' = 2 + 67 = 69;
intVar1 = 2 + charVar3 = 69;
charVar1 = 'A';
charVar2 = 'B';
```

Conditional Operators

- Conditional operator

  <expression1> ? <expression2> : <expression3>

  - Evaluate <expression1>
  - If the value of <expression1> is true (non-zero), evaluate and return <expression2>.
  - Otherwise, evaluate and return <expression3>.

```
int x = 1, y = 4;
int min;

min = (x<=y) ? x : y;

min = ?
```

- Evaluate: x<=y • 1 <= 4 • 1 (≠0)
- Evaluate: x
- Return: 1
- Copy 1 to the variable min

```
min = 1;
```

Type Casting Operators

- Type casting operator: (**type**) *expression*
  - Produces the value of *expression* in the **type**
  - Provides an explicit type conversion
  - Has the highest precedence
- Type casting vs. Type promotion vs. Truncation

char < short < int < long < float < double < long double

promotion →

truncation ←

```c
#include <stdio.h>

void main() {

    char charVar;
    int intVar = 1;
    float floatVar = 2.5;

    floatVar = 199/2;

    charVar = floatVar + 2;

    intVar = floatVar + 1;

    printf("charVar = %c with ASCII = floatVar + 2 = %d\n\n", charVar, charVar);

    printf("intVar = floatVar + 1 = %d\n\n", intVar);

    printf("floatVar = 199/2 = %e\n\n", floatVar);

    floatVar = (float)199/2;

    charVar = (char)floatVar + 2;

    intVar = (int)floatVar + 1;

    printf("Again, charVar = %c with ASCII = (char)floatVar + 2 = %d\n\n", charVar, charVar);

    printf("Again, intVar = (int)floatVar + 1 = %d\n\n", intVar);

    printf("Again, floatVar = (float)199/2 = %e\n\n", floatVar);

    floatVar = 199/(float)2;

    printf("Again, floatVar = 199/(float)2 = %e\n\n", floatVar);

    floatVar = (float)(199/2);

    printf("Again, floatVar = (float)(199/2) = %e\n\n", floatVar);
}
```

```
D:\CS - Introduction to Computer Programming - CO1003 - Undergraduate cours
charVar = e with ASCII = floatVar + 2 = 101

intVar = floatVar + 1 = 100

floatVar = 199/2 = 9.900000e+001

Again, charVar = e with ASCII = (char)floatVar + 2 = 101

Again, intVar = (int)floatVar + 1 = 100

Again, floatVar = (float)199/2 = 9.950000e+001

Again, floatVar = 199/(float)2 = 9.950000e+001

Again, floatVar = (float)(199/2) = 9.900000e+001
```

Other Operators

Name	Operator	Description	Example
sizeof	**sizeof**(type), **sizeof**(variable)	Returns the size (bytes) of a type or a variable	**sizeof**(**char**) **int** anInt = 0; **sizeof**(anInt);
address	**&**Variable	Returns the address of the memory named Variable	**char** aChar; **char*** ptrChar; ptrChar = **&**aChar;
Dereferencing	***Pointer**	Returns the value of the memory Pointer points to	aChar = ***ptrChar + 1;

Index	Variable**[..]**	Returns the element at the index	**int** intArray[3];
			
			intArray**[0]** = 0;
			intArray**[1]** = 1;
			intArray**[2]** = 2;
			anInt = intArray**[1]**;

Structure member	Structure_ name**.**member	Refers to a member of a particular structure	**struct** point pt; pt**.**x = 10;

Higher precedence

Operators	Associativity
`()  []  ->  .`	left to right
`! ~ ++ -- + - * & (type) sizeof`	right to left
`*  /  %`	left to right
`+  -`	left to right
`<<  >>`	left to right
`<  <=  >  >=`	left to right
`==  !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`?:`	right to left
`= += -= *= /= %= &= ^= |= <<= >>=`	right to left
`,`	left to right

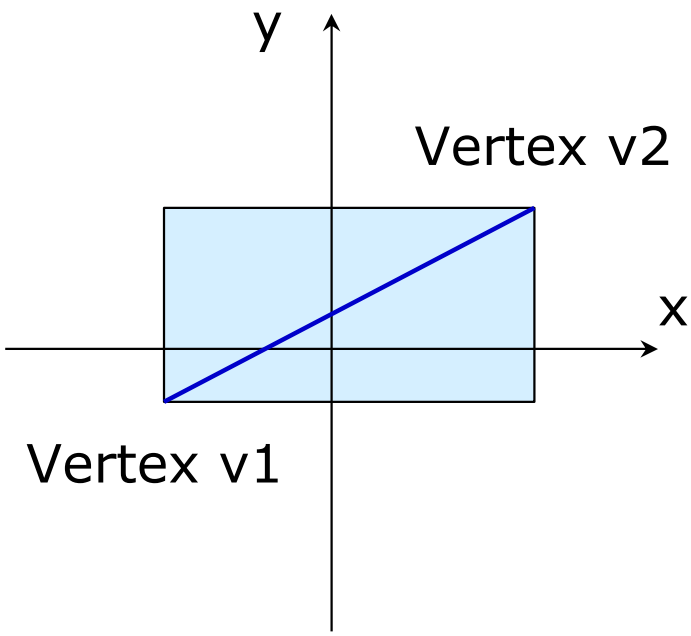
Lower precedence

Unary +, −, and * have higher precedence than the binary forms.

[2], pp. 53

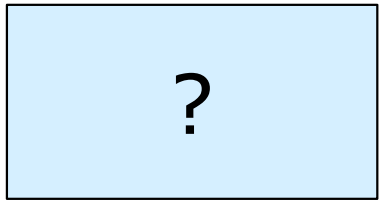
Put them altogether

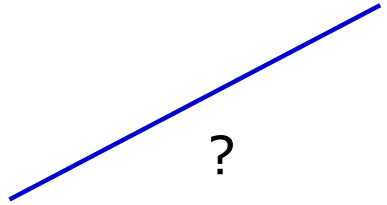
- Write a program to describe a rectangle by two opposite vertices with the following requirements:
  - Each vertex has integer coordinates.
  - Input the description of a given rectangle via its two opposite vertices
  - Calculate and print the area of a given aforementioned rectangle. If there is no area, print -1 instead.
  - Calculate and print the length of the diagonal line of a given aforementioned rectangle



v1.x, v1.y ?

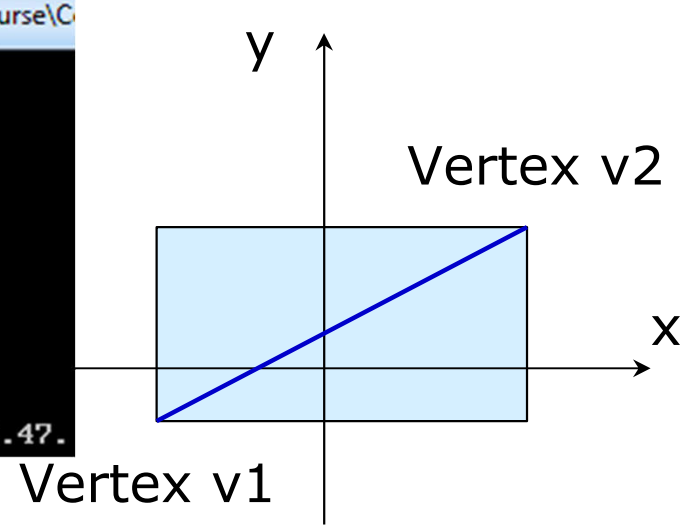
v2.x, v2.y ?





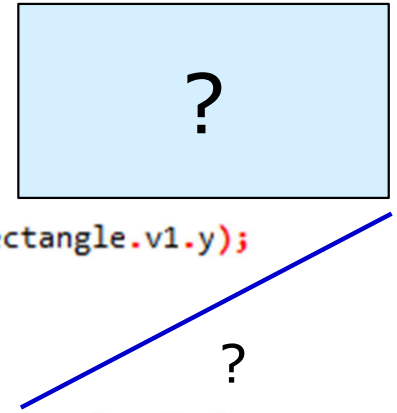
```c
#include <stdio.h>
#include <math.h>

typedef struct {
    int x;
    int y;
} vertex;

typedef struct {
    vertex v1;
    vertex v2;
} rectangle;

void main() {

    rectangle aRectangle;

    float area, diagonal;

    printf("Input the vertices v1 and v2:\n\n");

    printf("(x,y) for vertex v1 = ");
    scanf("%d,%d", &(aRectangle.v1.x), &(aRectangle.v1.y));

    printf("\naRectangle.v1.x = %d - aRectangle.v1.y = %d\n\n", aRectangle.v1.x, aRectangle.v1.y);

    printf("\n(x,y) for vertex v2 = ");
    scanf("%d,%d", &(aRectangle.v2.x), &(aRectangle.v2.y));

    printf("\naRectangle.v2.x = %d - aRectangle.v2.y = %d\n\n", aRectangle.v2.x, aRectangle.v2.y);

    area = abs((float)aRectangle.v1.x-aRectangle.v2.x)*abs(aRectangle.v1.y-aRectangle.v2.y);

    diagonal = sqrt(((float)aRectangle.v1.x-aRectangle.v2.x)*(aRectangle.v1.x-aRectangle.v2.x) \
                + (aRectangle.v1.y-aRectangle.v2.y)*(aRectangle.v1.y-aRectangle.v2.y));

    printf("\nThe area of the rectangle is %6.2f.\n\n", area>0?area:-1);

    printf("The length of the diagonal line of the rectangle is %6.2f.\n", diagonal);
}
```

```
D:\CS - Introduction to Computer Programming - CO1003 - Undergraduate course\C
Input the vertices v1 and v2:

(x,y) for vertex v1 = 3,6

aRectangle.v1.x = 3 - aRectangle.v1.y = 6

(x,y) for vertex v2 = 1,2

aRectangle.v2.x = 1 - aRectangle.v2.y = 2

The area of the rectangle is   8.00.

The length of the diagonal line of the rectangle is   4.47.
```



v1.x, v1.y ?

v2.x, v2.y ?



Summary

- How to handle data in a C program
  - Data types
    - Built-in: char, int, float, double, …, void
    - Derived: array, pointer, structure, union, enum
  - → enum and structure for abstract data
  - → More advanced types (array, pointer) come later.
  - Variables: declaration vs. definition
    - Naming
    - Storage
    - Type
    - Value
    - Scope

Summary

- Constants
  - No change during the execution of the program
  - Known at the compile time
  - Defined with: #define, enum, const
- Expressions: value, type
- Operators
  - Assignment
  - Arithmetic
  - Bitwise
  - Logic
  - Relational and others
- Type casting: explicit vs. implicit conversion

Summary

- Data
  - "information, especially facts or numbers, collected for examination and consideration and used to help decision-making, or information in an electronic form that can be stored and processed by a computer" – Cambridge Dictionary
- How to handle data in a C program
  - Under control!

Chapter 3: Variables and Basic Data Types