Fundamentals of C

First C program – print on screen

```
#include <stdio.h>
void main()
{
    printf ("Hello, World! \n") ;
}
```

More print

```
#include <stdio.h>
void main()
{
    printf ("Hello, World! ") ;
    printf ("Hello \n World! \n") ;
}
```

Some more print

```
#include <stdio.h>
void main()
{
    printf ("Hello, World! \n") ;
    printf ("Hello \n World! \n") ;
    printf ("Hell\no \t World! \n") ;
}
```

Reading values from keyboard

```
#include <stdio.h>
void main()
{
        int num ;
        scanf ("%d", &num) ;
        printf (“No. of students is %d\n”, num) ;
}
```

Centigrade to Fahrenheit

```
#include <stdio.h>
void main()
{
    float cent, fahr;
    scanf(“%f”,&cent);
    fahr = cent*(9.0/5.0) + 32;
    printf( “%f C equals %f F\n”, cent, fahr);
}
```

Largest of two numbers

```
#include <stdio.h>
void main()
{
    int x, y;
    scanf(“%d%d”,&x,&y);
    if (x>y) printf(“Largest is %d\n”,x);
    else printf(“Largest is %d\n”,y);
}
```

What does this do?

```
#include <stdio.h>
void main()
{
    int x, y;
    scanf(“%d%d”,&x,&y);
    if (x>y) printf(“Largest is %d\n”,x);
    printf(“Largest is %d\n”,y);
}
```

largest-2.c

The C Character Set

- The C language alphabet
  - Uppercase letters ‘A’ to ‘Z’
  - Lowercase letters ‘a’ to ‘z’
  - Digits ‘0’ to ‘9’
  - Certain special characters:

!	#	%	^	&	*	(	)
-	_	+	=	~	[	]	\
|	;	:	‘	“	{	}	,
.	<	>	/	?	blank		

A C program should not contain anything else

Structure of a C program

- A collection of functions (we will see what they are later)
- Exactly one special function named main must be present. Program always starts from there
- Each function has statements (instructions) for declaration, assignment, condition check, looping etc.
- Statements are executed one by one

Variables

- Very important concept for programming
- An entity that has a value and is known to the program by a name
- Can store any temporary result while executing a program
- Can have only one value assigned to it at any given time during the execution of the program
- The value of a variable can be changed during the execution of the program

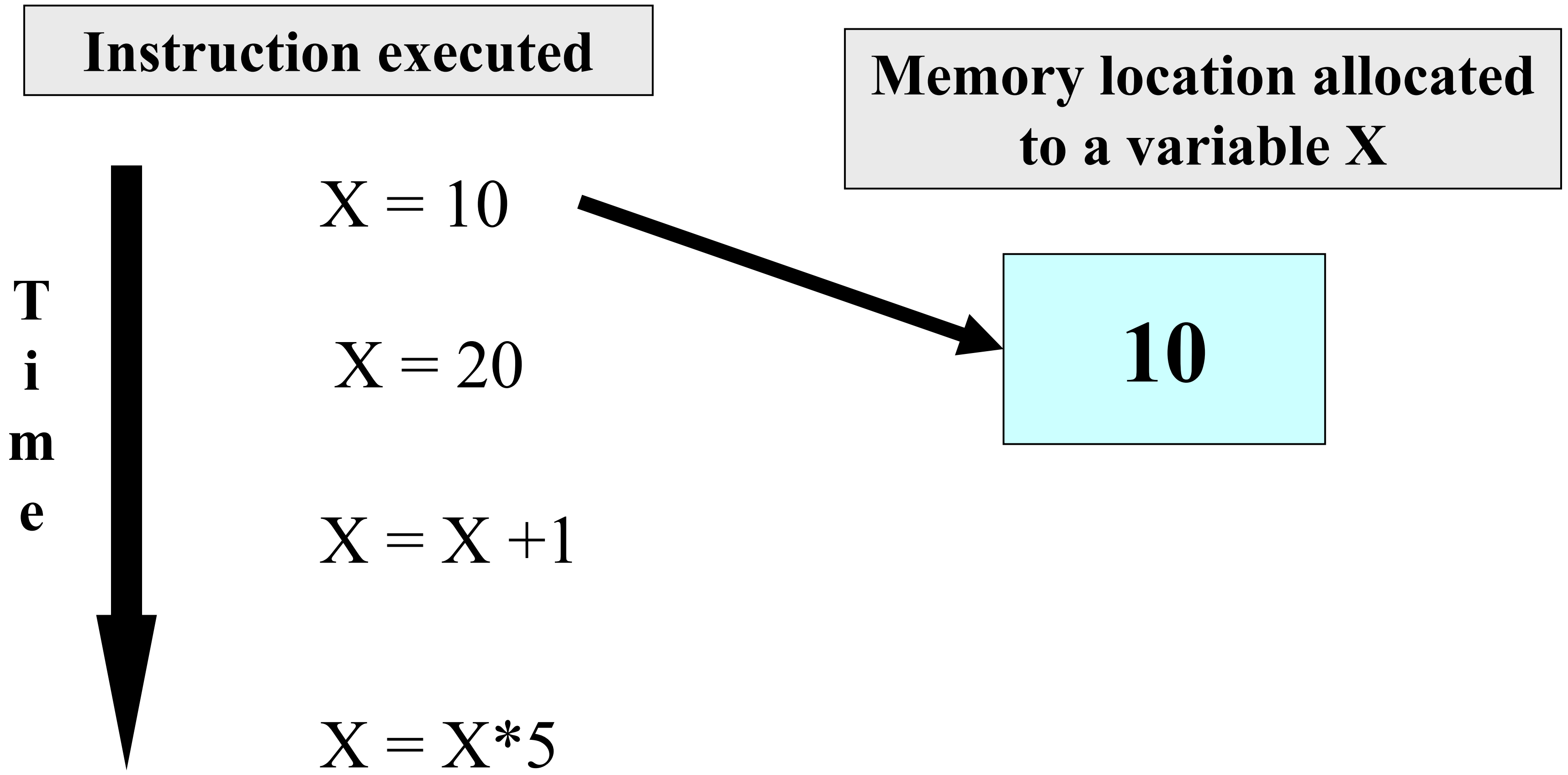
Contd.

- Variables stored in memory
- Remember that memory is a list of storage locations, each having a unique address
- A variable is like a bin
  - The contents of the bin is the value of the variable
  - The variable name is used to refer to the value of the variable
  - A variable is mapped to a location of the memory, called its address

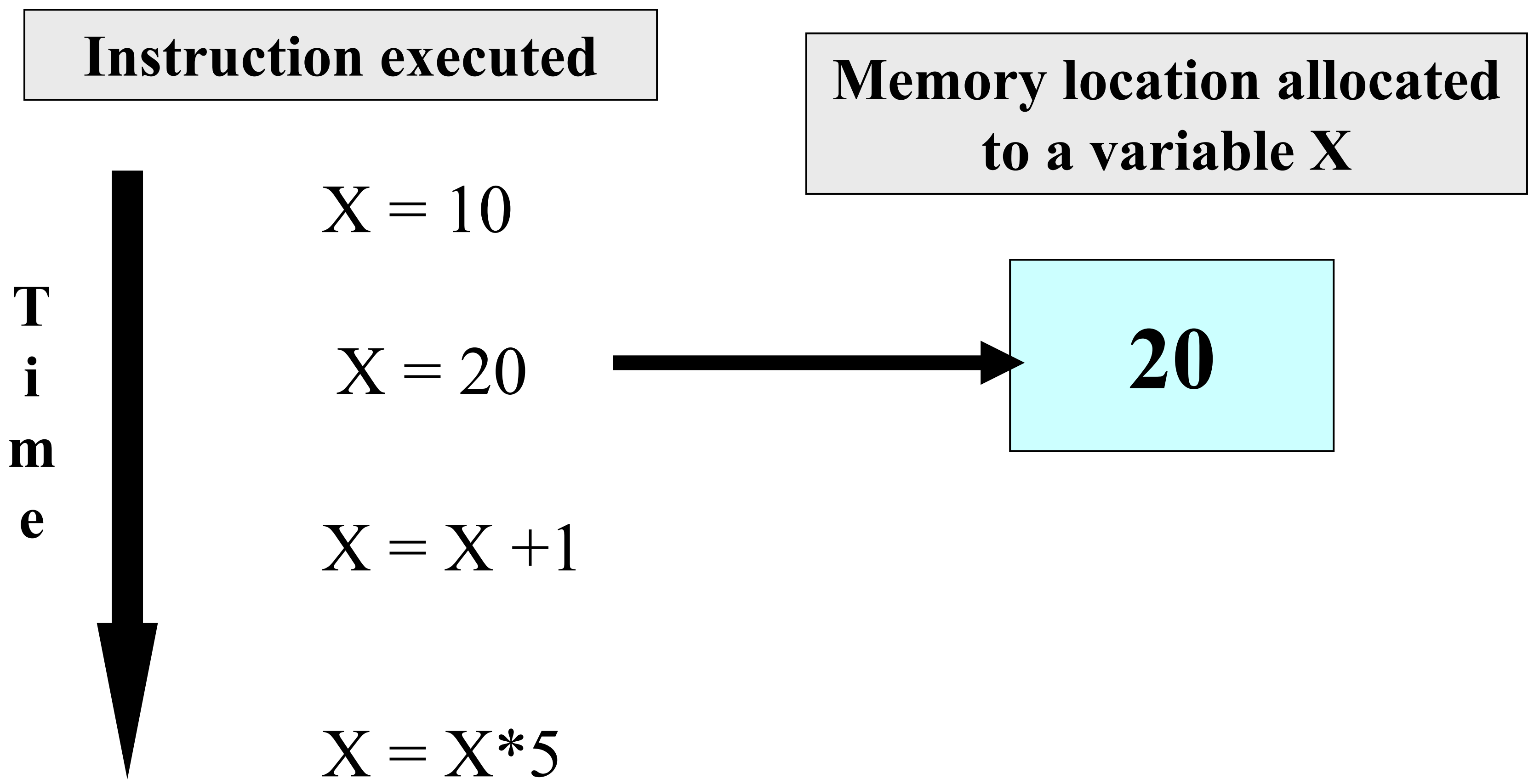
Example

```
#include <stdio.h>
void main( )
{
      int x;
      int y;
      x=1;
      y=3;
      printf("x = %d, y= %d\n", x, y);
}
```

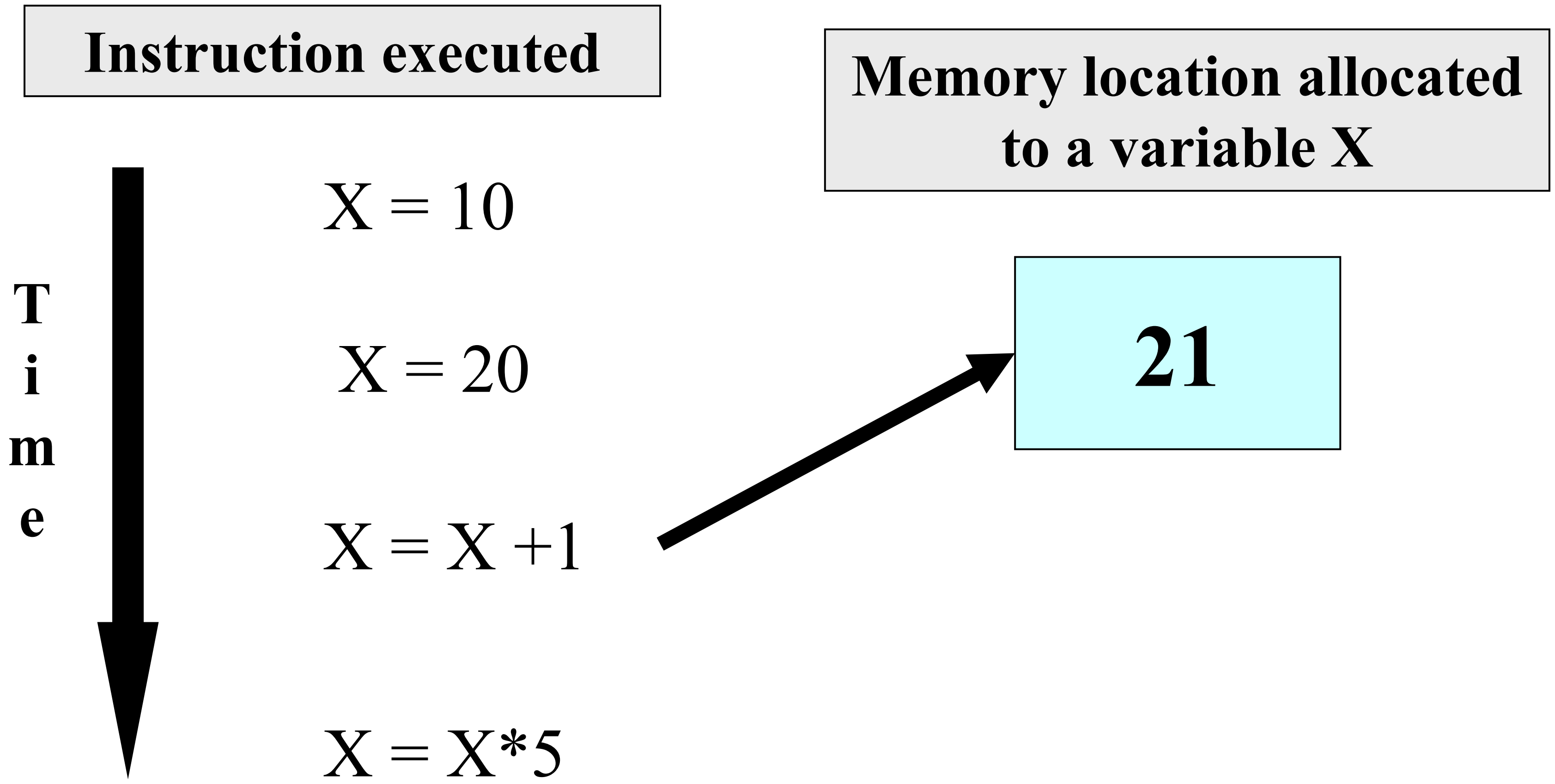
Variables in Memory



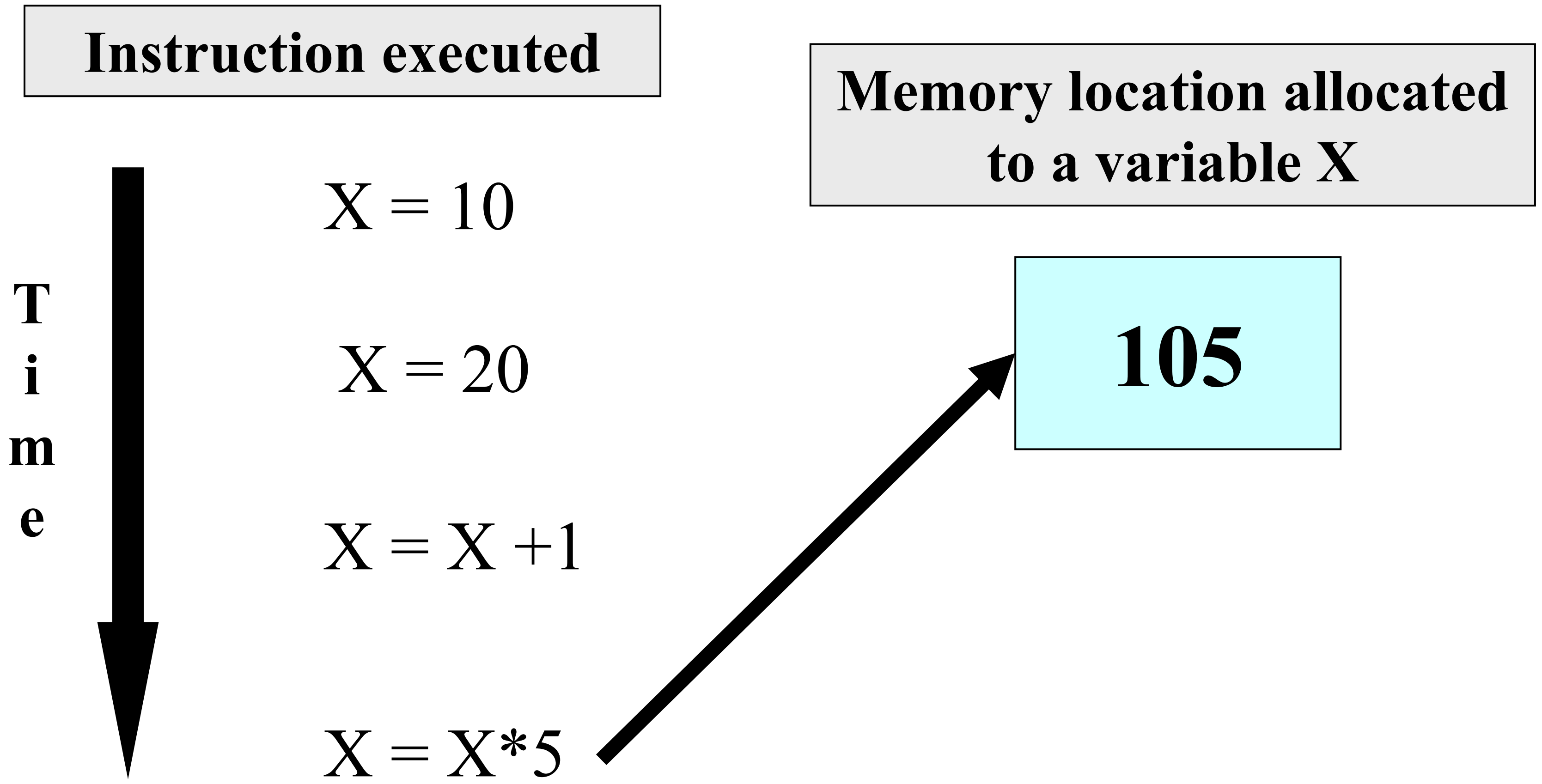
Variables in Memory



Variables in Memory



Variables in Memory



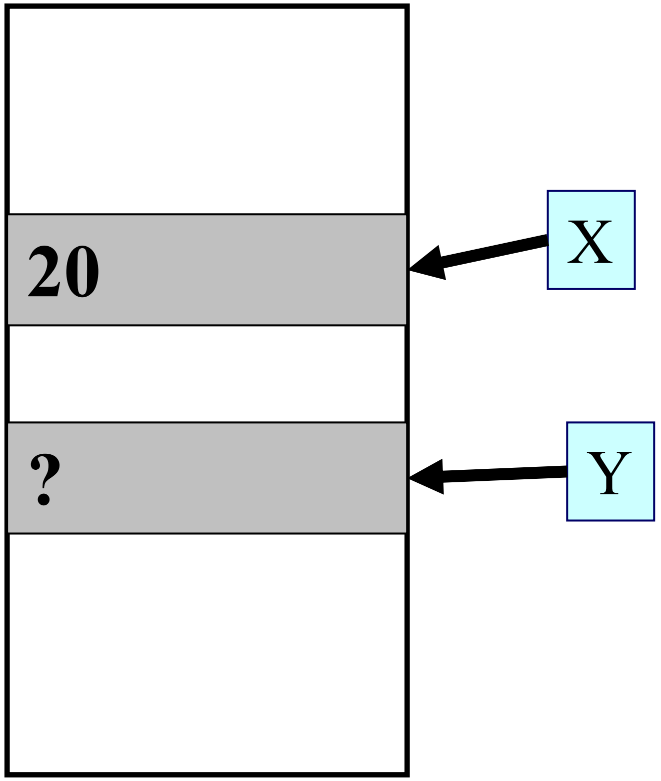
Variables (contd.)

```
X = 20

Y=15

X = Y+3

Y=X/6
```



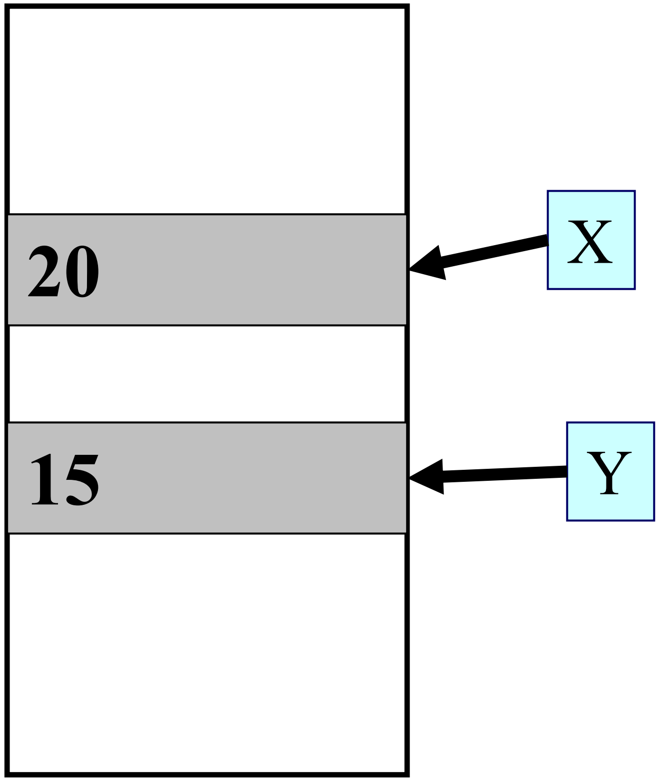
Variables (contd.)

```
X = 20

Y=15

X = Y+3

Y=X/6
```



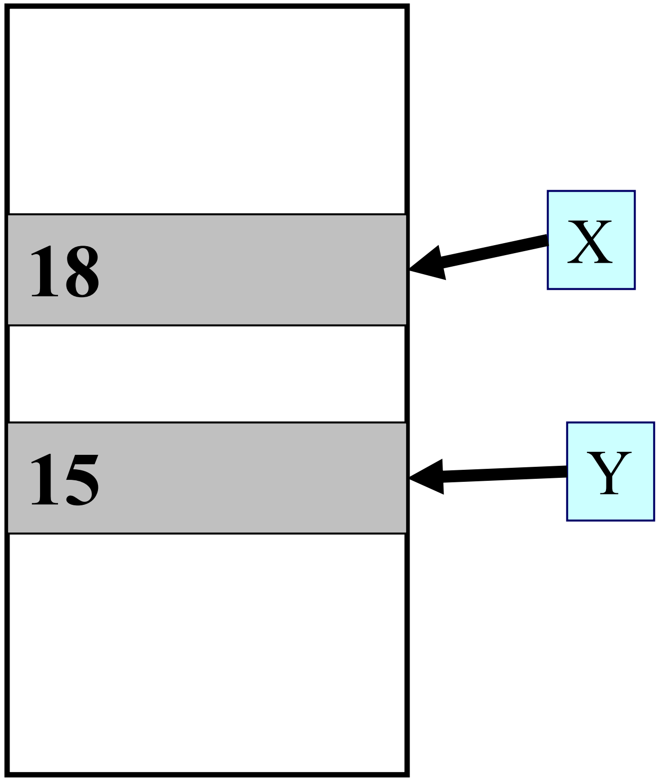
Variables (contd.)

X = 20

Y=15

X = Y+3

Y=X/6



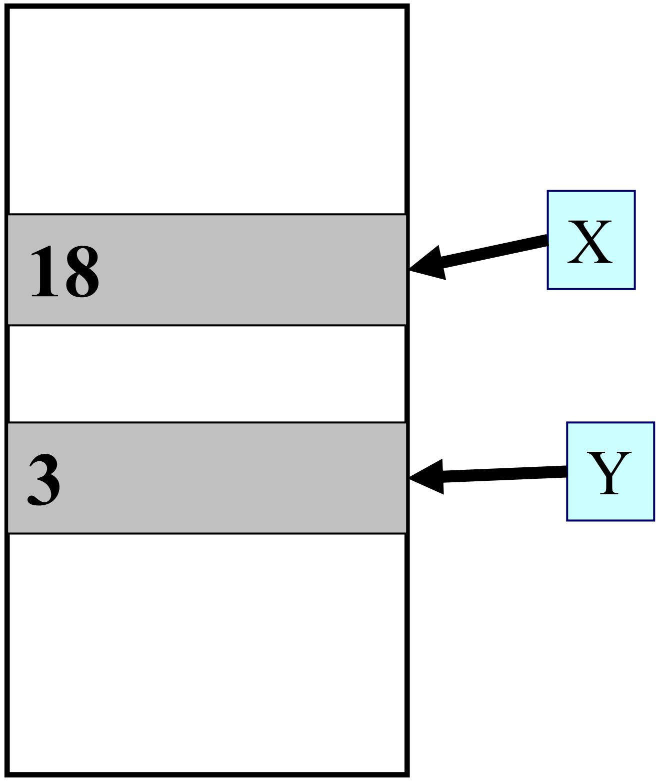
Variables (contd.)

X = 20

Y=15

X = Y+3

Y=X/6



Data Types

- Each variable has a type, indicates what type of values the variable can hold
- Four common data types in C
  - int - can store integers (usually 4 bytes)
  - float - can store single-precision floating point numbers (usually 4 bytes)
  - double - can store double-precision floating point numbers (usually 8 bytes)
  - char - can store a character (1 byte)

Contd.

- Must declare a variable (specify its type and name) before using it anywhere in your program
- All variable declarations should be at the beginning of the main() or other functions
- A value can also be assigned to a variable at the time the variable is declared.

```
    int   speed = 30;
    char  flag = ‘y’;
```

Variable Names

- Sequence of letters and digits
- First character must be a letter or '_'
- No special characters other than '_'
- No blank in between
- Names are case-sensitive (max and Max are two different names)
- Examples of valid names:
  - i rank1 MAX max Min class_rank
- Examples of invalid names:
  - a's fact rec 2sqroot class,rank

More Valid and Invalid Identifiers

- Valid identifiers

  X

  abc

  simple_interest

  a123

  LIST

  stud_name

  Empl_1

  Empl_2

  avg_empl_salary

- Invalid identifiers

  10abc

  my-name

  “hello”

  simple interest

  (area)

  %rate

C Keywords

- Used by the C language, cannot be used as variable names
- Examples:
  - int, float, char, double, main, if else, for, while. do, struct, union, typedef, enum, void, return, signed, unsigned, case, break, sizeof,….
  - There are others, see textbook…

Example 1

```
#include <stdio.h>
void main()
{
    int x, y, sum;
    scanf(“%d%d”,&x,&y);
    sum = x + y;
    printf( “%d plus %d is %d\n”, x, y, sum );
}
```

Three int type variables declared

Values assigned

Example - 2

```
#include <stdio.h>
void main()
{
      float x, y;
      int d1, d2 = 10;
      scanf("%f%f%d",&x, &y, &d1);
      printf( "%f plus %f is %f\n", x, y, x+y);
      printf( "%d minus %d is %d\n", d1, d2, d1-d2);
}
```

Assigns an initial value to d2, can be changed later

Read-only variables

- Variables whose values can be initialized during declaration, but cannot be changed after that
- Declared by putting the const keyword in front of the declaration
- Storage allocated just like any variable
- Used for variables whose values need not be changed
  - Prevents accidental change of the value

Correct

```
void main() {
   const int LIMIT = 10;
   int n;
   scanf(“%d”, &n);
   if (n > LIMIT)
      printf(“Out of limit”);
}
```

Incorrect: Limit changed

```
void main() {
   const int Limit = 10;
   int n;
   scanf(“%d”, &n);
   Limit = Limit + n;
   printf(“New  limit is %d”, Limit);
}
```

Constants

- Integer constants
  - Consists of a sequence of digits, with possibly a plus or a minus sign before it
  - Embedded spaces, commas and non-digit characters are not permitted between digits
- Floating point constants
- Two different notations:
  - Decimal notation: 25.0, 0.0034, .84, -2.234
  - Exponential (scientific) notation

    3.45e23, 0.123e-12, 123e2

e means "10 to the power of"

Contd.

- Character constants
  - Contains a single character enclosed within a pair of single quote marks.
  - Examples :: ‘2’, ‘+’, ‘Z’
- Some special backslash characters

‘\n’	new line
‘\t’	horizontal tab
‘\’’	single quote
‘\”’	double quote
‘\\’	backslash
‘\0’	null

Input: scanf function

- Performs input from keyboard
- It requires a format string and a list of variables into which the value received from the keyboard will be stored
- format string = individual groups of characters (usually ‘%’ sign, followed by a conversion character), with one character group for each variable in the list

```
int a, b;
float c;
scanf(“%d %d %f”, &a, &b, &c);
```

$$\texttt{scanf(“}\underbrace{\texttt{\%d \%d \%f}}_{\text{Format string}}\texttt{”, }\overbrace{\texttt{\&a, \&b, \&c}}^{\text{Variable list (note the \& before a variable name)}}\texttt{);}$$

- Commonly used conversion characters

c	for char type variable
d	for int type variable
f	for float type variable
lf	for double type variable

- Examples

```
scanf ("%d", &size) ;
scanf ("%c", &nextchar) ;
scanf ("%f", &length) ;
scanf (“%d%d”, &a, &b);
```

Reading a single character

- A single character can be read using scanf with %c
- It can also be read using the getchar() function

```
char c;
c = getchar();
```

- Program waits at the getchar() line until a character is typed, and then reads it and stores it in c

Output: printf function

- Performs output to the standard output device (typically defined to be the screen)
- It requires a format string in which we can specify:
  - The text to be printed out
  - Specifications on how to print the values

```
printf ("The number is %d\n", num);
```

  - The format specification %d causes the value listed after the format string to be embedded in the output as a decimal number in place of %d
  - Output will appear as: The number is 125

Contd.

- General syntax:

  ```
  printf (format string, arg1, arg2, …, argn);
  ```

  - format string refers to a string containing formatting information and data types of the arguments to be output
  - the arguments arg1, arg2, … represent list of variables/expressions whose values are to be printed
- The conversion characters are the same as in scanf

- Examples:

```
printf  (“Average of %d and %d is %f”, a, b, avg);
printf  (“Hello \nGood \nMorning \n”);
printf (“%3d %3d %5d”, a, b, a*b+2);
printf (“%7.2f  %5.1f”, x, y);
```

- Many more options are available for both printf and scanf
  - Read from the book
  - Practice them in the lab