Functions for modular programming

- Programs in the real world can be extremely large (e.g., compilers and operating systems can consist of thousands of lines of code, often produced by teams of programmers)
- One way to manage large programming projects is to properly divide the task into small manageable modules known as functions

jmsamaniego@uplb.edu.ph (revised 2006)

A program is a collection of functions

- main() -- all programs basically have this function, and execution starts with this main() function
- a large program typically has many other functions which are invoked/ called by the main() function or other functions
- programs that are properly designed have a main() function that looks like an outline, and the details are spelled out in various functions

Predefined functions

- Functions are not really new, we have been using many predefined functions such as
  - printf() -- for screen output
  - scanf() -- for keyboard input
  - sqrt() -- a math function for square root
  - strcpy() -- to assign a string to a string var
- Lots more can be used when we include stdio.h, math.h, string.h, assert.h, etc. -- have a look at the directory /usr/include/

jmsamaniego@uplb.edu.ph (revised 2006)

Some useful predefined functions

Math functions

- sqrt(x), sin(x), cos(x), ...
- rand()   // returns a pseudo-random int
- srand(seed)   // re-seeds the random generator

String functions

- strcpy(x, y)   // copies string y to string x
- strlen(x)   // returns number of chars in x

jmsamaniego@uplb.edu.ph (revised 2006)

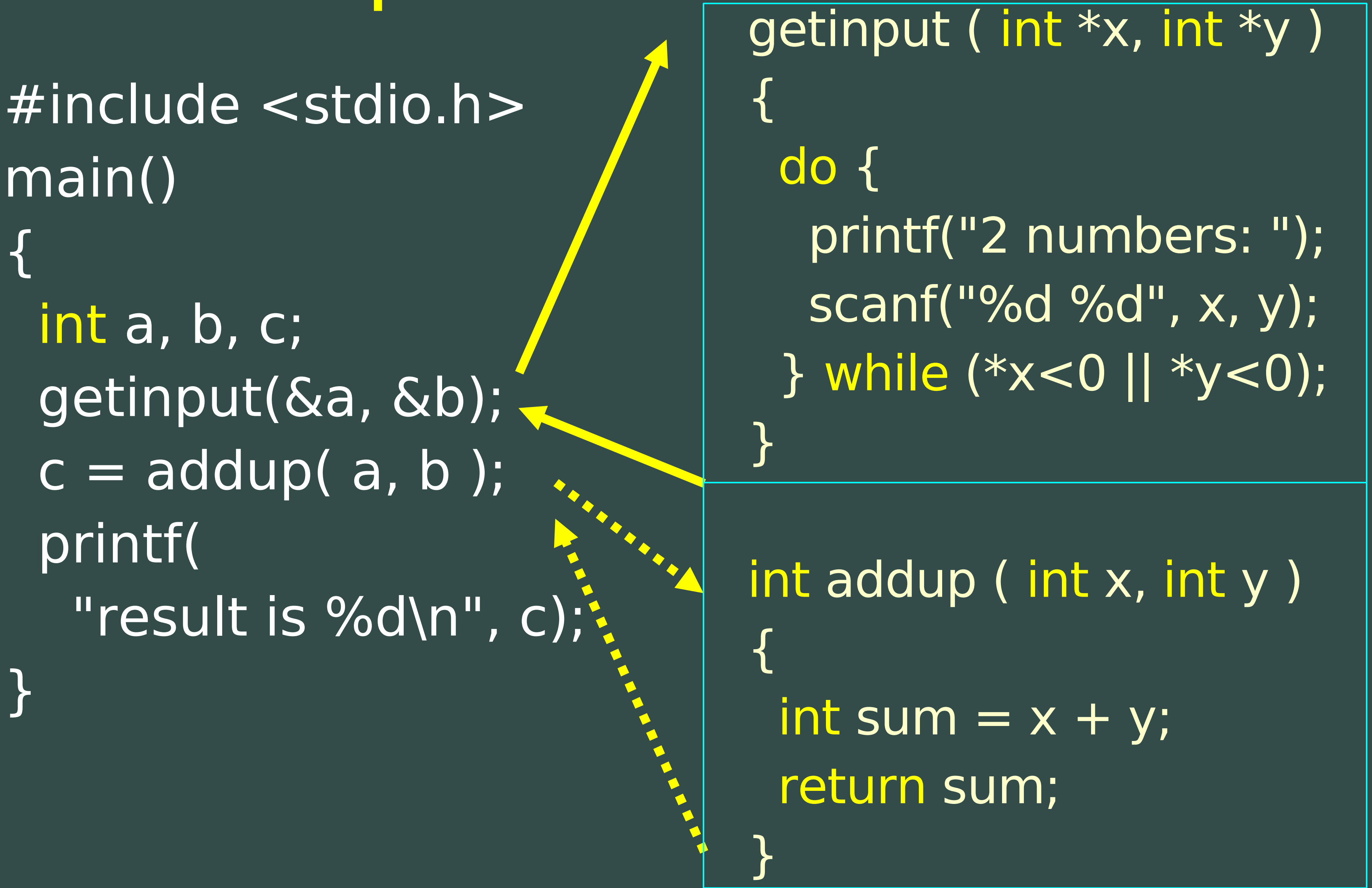
Generating random numbers

(for games of chance and simulations)

```
main()
{
    int j;
    srand(1);  /* change the seed */
               /* for a different sequence */
    for (j=0; j<200; j++) {
       printf("%d ", rand()%10 );
    } /* prints 200 random digits */
}
```

jmsamaniego@uplb.edu.ph (revised 2006)

An example



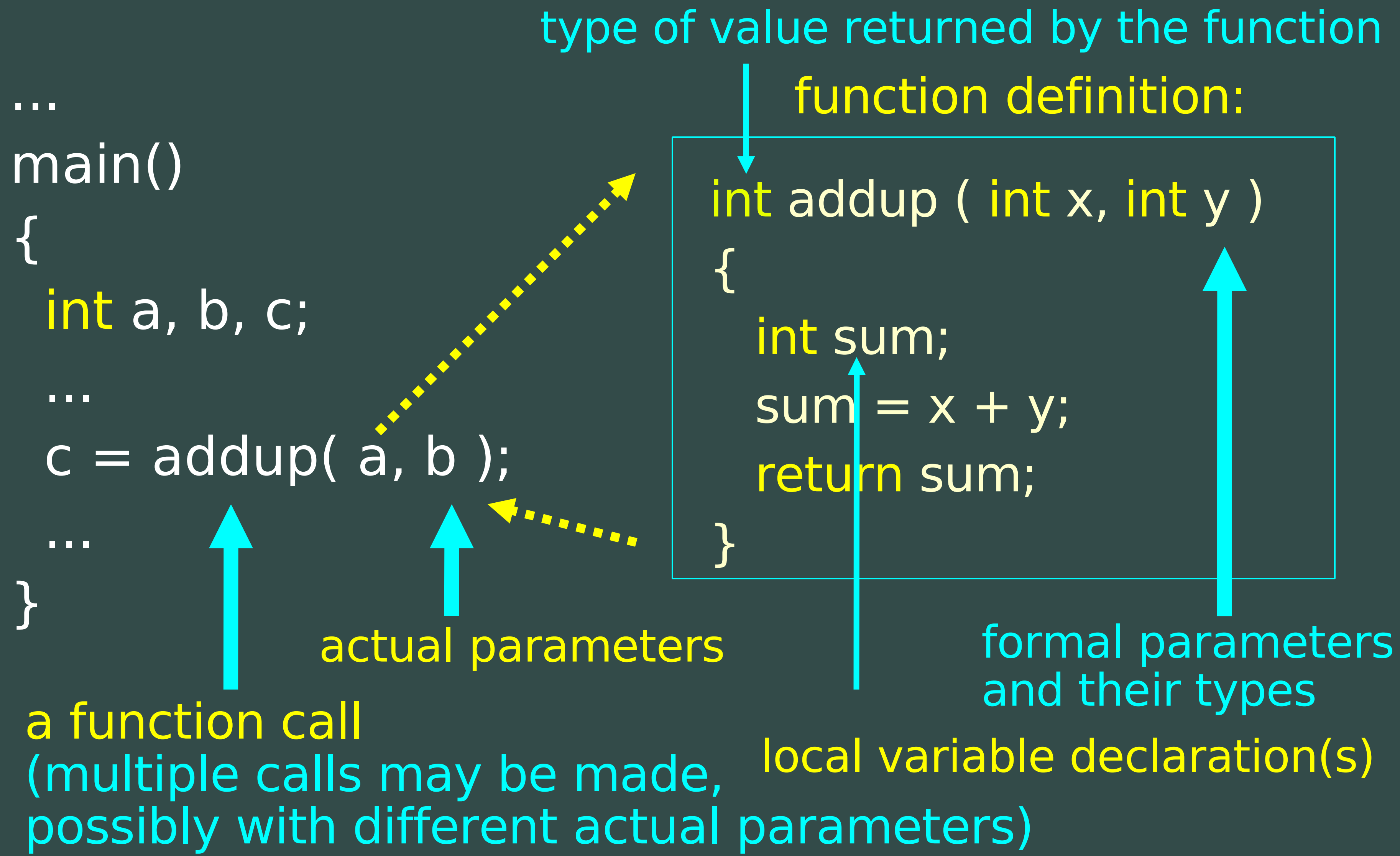
```
#include <stdio.h>
main()
{
  int a, b, c;
  getinput(&a, &b);
  c = addup( a, b );
  printf(
    "result is %d\n", c);
}
```

```
getinput ( int *x, int *y )
{
  do {
    printf("2 numbers: ");
    scanf("%d %d", x, y);
  } while (*x<0 || *y<0);
}
```

```
int addup ( int x, int y )
{
  int sum = x + y;
  return sum;
}
```

jmsamaniego@uplb.edu.ph (revised 2006)

Anatomy of a function call and a function definition



```
...
main()
{
  int a, b, c;
  ...
  c = addup( a, b );
  ...
}
```

function definition:

```
int addup ( int x, int y )
{
   int sum;
   sum = x + y;
   return sum;
}
```

type of value returned by the function

actual parameters

formal parameters and their types

local variable declaration(s)

a function call
(multiple calls may be made,
possibly with different actual parameters)

jmsamaniego@uplb.edu.ph (revised 2006)

Advantages of functions and modular programming

- Avoid redundancy – lengthy code that is repeated at different parts of a program need to be written only once
- Encourage re-usability – frequently-used functions can be added to a library (e.g., frequently used math or string functions)
- Improve readability – implementation details are hidden in the functions
- Manage complexity – large software engineering projects are split into logical modules that can be developed and tested separately (simultaneous development is even possible with teams of programmers)

jmsamaniego@uplb.edu.ph (revised 2006)

Defining your own functions

```
float square( float x )
{
    return x*x;
}

float cube( float x )
{
    return x*x*x;
}
```

jmsamaniego@uplb.edu.ph (revised 2006)

Calling your functions

```
/* a small table of squares and cubes */
float square(float x);    // function headings
float cube(float x);
main()
{
    int j;
    for (j=1; j<6; j++) {
       printf(“%d %10f %10f \n”,
            j, square(j), cube(j) );
    }
}
```

jmsamaniego@uplb.edu.ph (revised 2006)

More examples of function definitions

```
float max( float x, float y )
{  // find and return the larger value of 2 numbers
   if (x >= y) return x;
   else return y;
}


float maxof3 ( float x, float y, float z )
{// find and return the largest among 3 numbers
   float temp = max( x, y );
   return max(temp, z);
}
```

jmsamaniego@uplb.edu.ph (revised 2006)

Functions can call previously-defined functions

```
float maxof3 ( float x, float y, float z )
{  // poor version of same function in the previous slide
   float temp;

   if (x >= y) temp = x;
   else temp = y;

   if (temp >= z) return temp;
   else return z;
}
```

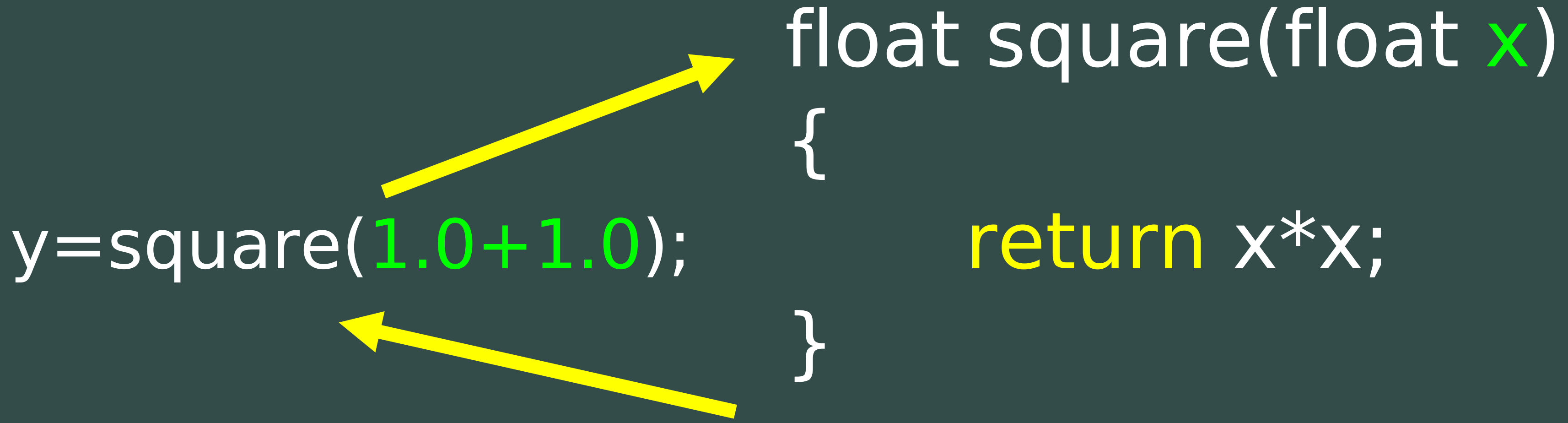
jmsamaniego@uplb.edu.ph (revised 2006)

Good programming style with functions

- Use predefined functions when available (don't reinvent the wheel – except when you want to know how wheels are made)
- If you have to write your own function, consider using appropriate parameters to make it more useful
- Using local variables make functions as self-contained as possible
- One can use the function simply by knowing the right parameters and the return value; one does not need to know how the function works in detail

Parameters

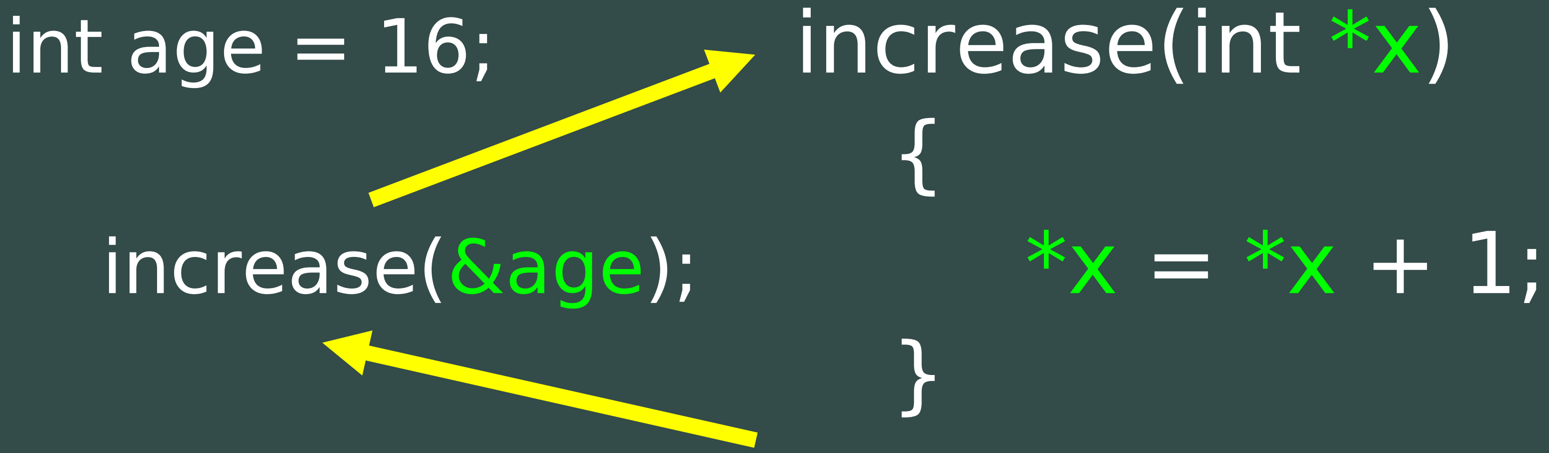
- Parameters and the return value are used to communicate data between the caller and the function
- Most parameters serve as “input” to the function, the “output” is returned using the return statement



```
                          float square(float x)
                          {
y=square(1.0+1.0);              return x*x;
                          }
```

Output parameters

- A parameter can be changed by passing the address of a variable (instead of its value)



```
int age = 16;                increase(int *x)
                                 {
   increase(&age);                   *x = *x + 1;
                                 }
```

More on parameter-passing
(call-by-value)

```
main()
{
    int n = 5;
    printf("%d", n);
    foo(n);
    printf("%d", n);
}
```

```
foo ( int x )
{
    x = x + 1;
}
```

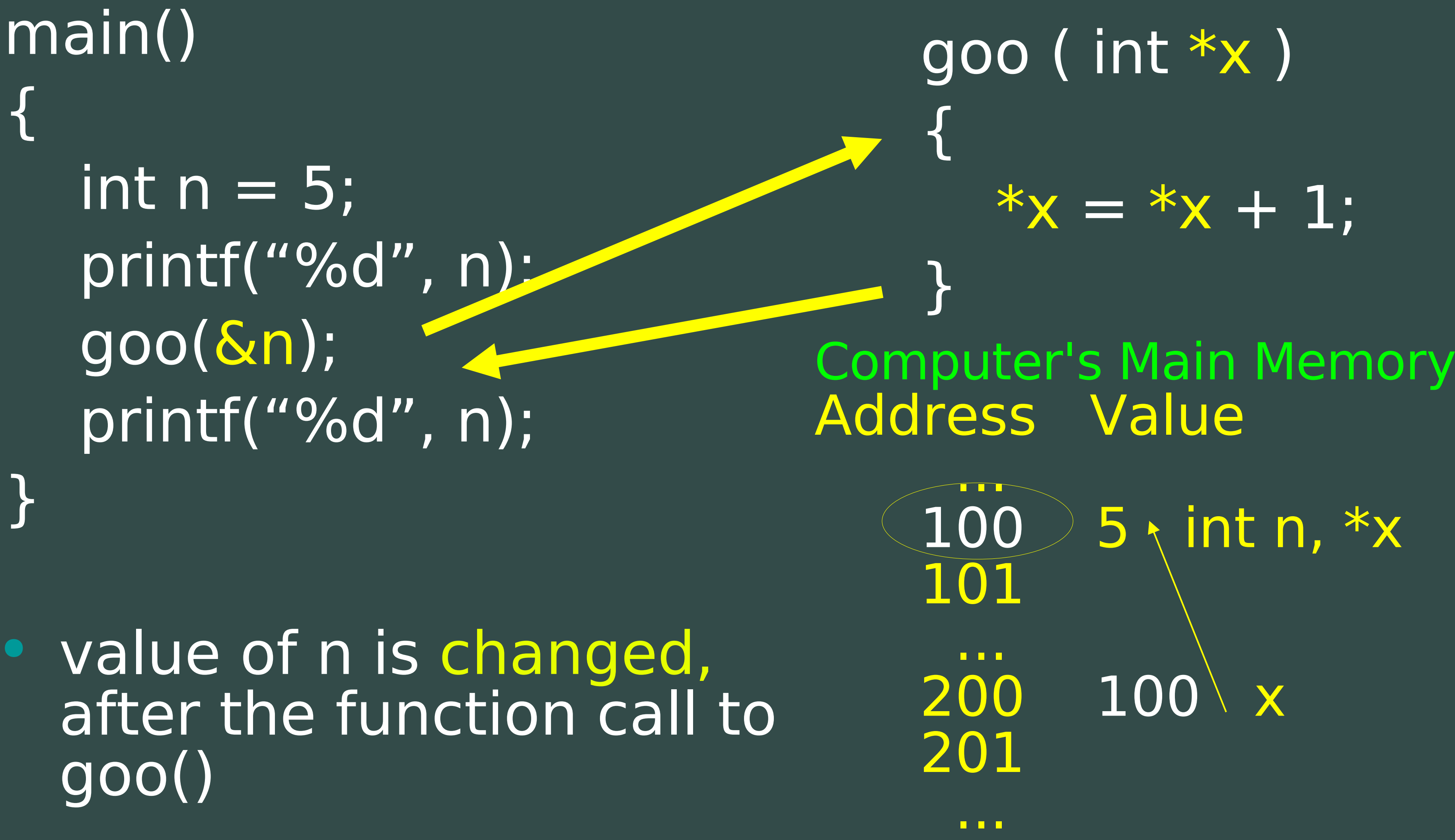
- value of n is unchanged, even after the function call to foo()

Computer's Main Memory

Address	Value	
...		
100	5	(int n)
101		
...		
200	5	(int x)
201		
...		

jmsamaniego@uplb.edu.ph (revised 2006)

Passing an address instead of a value



```
main()
{
    int n = 5;
    printf(“%d”, n);
    goo(&n);
    printf(“%d”, n);
}
```

```
goo ( int *x )
{
    *x = *x + 1;
}
```

Computer's Main Memory

Address	Value	
...		
100	5	int n, *x
101		
...		
200	100	x
201		
...		

- value of n is changed, after the function call to goo()

Example: swapping values

```
main()
{
    int a=1, b=2;
    printboth( a, b );
    swap( &a, &b );
    printboth( a, b );
}
```

```
printboth( int x, int y )
{
  printf(“%d and %d \n”,
    x, y);
}

swap( int *x, int *y )
{
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

jmsamaniego@uplb.edu.ph (revised 2006)

Sorting any three numbers in ascending order

```
main()
{
    int a, b, c;
    printf(“input any 3 numbers: ”);
    scanf(“%d %d %d”, &a, &b, &c);
    if (a > b) swap(&a, &b);
    if (b > c) swap(&b, &c);  // largest now in c
    if (a > b) swap(&a, &b);  // a, b, c now sorted
    printf(“sorted: %d %d %d\n”, a, b, c);
}
```