CS101 Introduction to computing

Pointer and Function

A. Sahu and P. Mitra

Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Outline

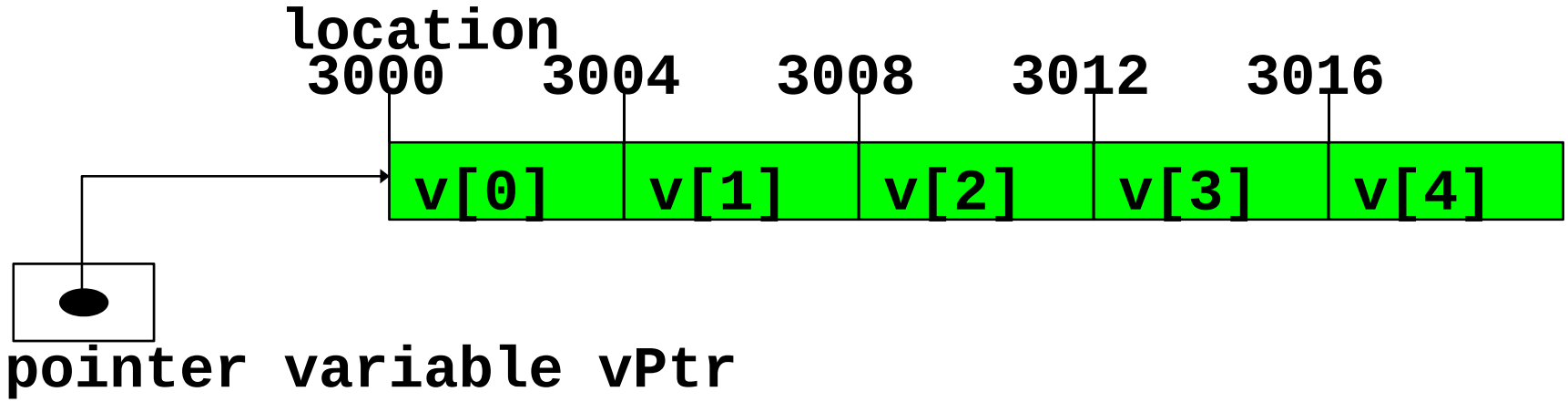
- Pointer
  - Memory access
  - Access using pointer
- Basic Pointer Arithmetic
- Function

Pointer and Array

- 5 element **int** array with 4 byte **int**s
- **vPtr** points to first element **v[0]**
  - at location **3000** (**vPtr = 3000**)

```
int v[5];
int *vPtr=v;
```

- **vPtr += 2;** sets **vPtr** to **3008**
  - **vPtr** points to **v[2]** (incremented by 2), but the machine has 4 byte **int**s, so it points to address **3008**



Demo-Ptr-vs Array

```
main() {
   int V[5]={2,1,4,6,3};
   int *vPtr=V;  /* V=vPtr;//notLegal*/
   printf("V=%p, &V[0]=%p\n",
                  V, &V[0]);
   printf("V[0]=%d   *vPtr=%d\n",
                 V[0], *vPtr);
   printf("V[2]=%d   *(vPtr+2)=%d\n",
                 V[2], *(vPtr+2));
   prntf(“%d %d %d %d”,
         V[2], 2[V], vPtr[2], 2[VPtr]);
}
```

V[2],2[V],vPtr[2] and 2[VPtr] are same

*(V+2), *(2+V), *(vPtr+2), *(2+vPtr)

Pointer Arithmetic

- Subtracting pointers
  - Returns number of elements from one to the other. If

```
vPtr2 = &v[2];
vPtr = &v[0];
```

    `vPtr2 - vPtr` would produce 2
- Pointer comparison (**`<`**, **`==`** , **`>`**)
  - See which pointer points to the higher numbered array element
  - Also, see if a pointer points to **`0`**

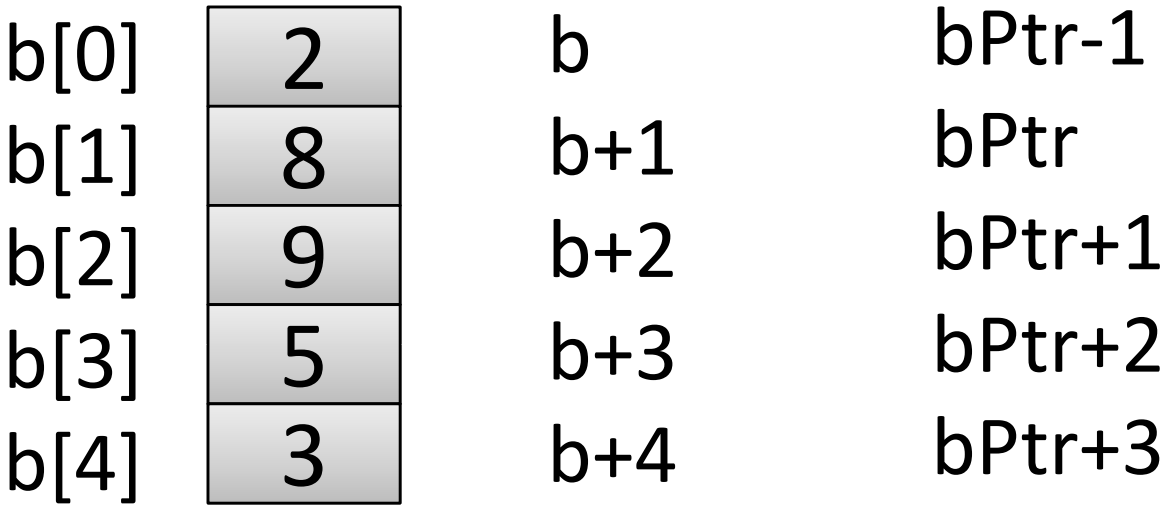
Pointer Arithmetic

- Pointers of the same type can be assigned to each other
  - If not the same type, a cast operator must be used
  - Exception: pointer to **`void`** (type **`void *`**)
- Generic pointer, represents any type
  - No casting needed to convert a pointer to **`void`** pointer
  - **`void`** pointers cannot be dereferenced

Array and Pointer

- Arrays and pointers closely related
  - **Array name like a constant pointer**
  - Pointers can do array subscripting operations

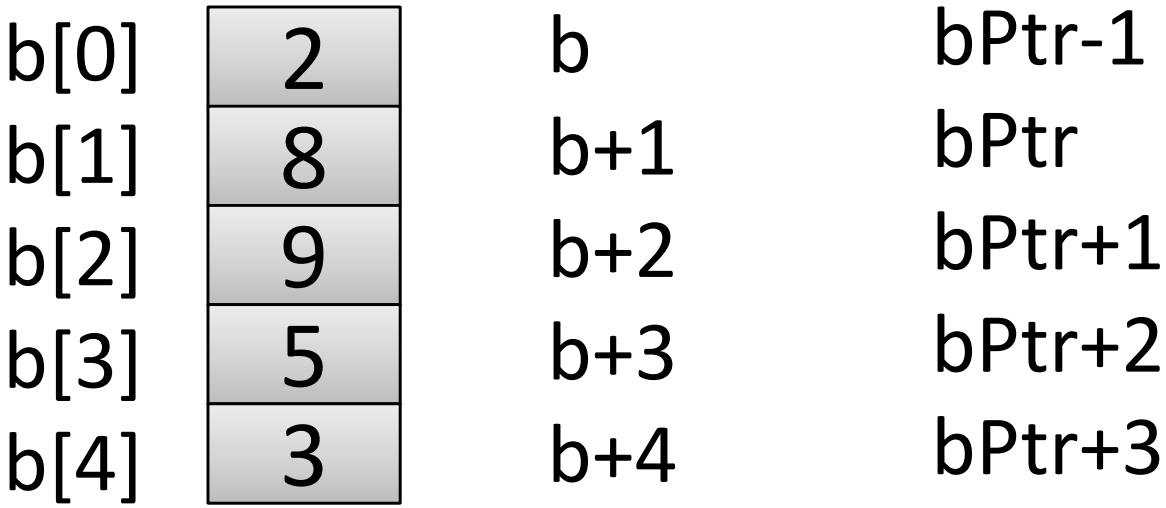
```
int  b[5]={2,8,9,5,3};
int *bPtr;
     bPtr=&b[1];
```



Array and Pointer

- Arrays and pointers closely related
  - Array name like a constant pointer
  - Pointers can do array subscripting operations

```
double  b[5]={2,8,9,5,3};
double  *bPtr;
        bPtr=&b[1];
```



Array and Pointer

```
int  b[5];
int *bPtr;
```

- To set them equal to one another use:

  `bPtr = b;`

  – The array name (**`b`**) is actually the address of first element of the array **`b[ 5 ]`**

  `bPtr = &b[ 0 ]`

  – Explicitly assigns **`bPtr`** to address of first element of **`b`**

Array and Pointer

```
int  b[5];
int *bPtr;
```

- Element **b[3]:**
  - Can be accessed by ***(bPtr + 3)**
  - Where **n** is the offset. Called pointer/offset notation

Array and Pointer

```
int  b[5];
int *bPtr;
```

- Element **b[3]**
  - Can be accessed by **bptr[3]**
  - Called pointer/subscript notation
  - **bPtr[3]** same as **b[3]**
- Element **b[3]**
  - Can be accessed by performing pointer arithmetic on the array itself ***(b + 3)**

Array and Pointer

- `int A[10];`
  `int *p;`
  - *Type* of **A** is `int *`
  - `p = A;` //legal assignment
  - `A = p;` // not legal assignment
  - `*p` refers to `A[0]`
    `*(p + n)` refers to `A[n]`
  - `p = &A[5];` is the same as `p = A+5;`

Array Name is pointer but const Ptr: == >

```
int * const A;
```

Array and Pointer

```
int  A[5],i,S=0;
int *APtr;

for(i=0;i<5;i++){
   S=S+A[i];
}
```

```
int  A[5],i,S=0;
int *APtr;

for(i=0;i<5;i++){
   S=S+*(A+i);
}
```

Array and Pointer

```
 int  A[5],i,S=0;
 int *APtr;

for(i=0;i<5;i++){
   S=S+A[i];
 }
```

```
 int  A[5],i,S=0;
 int *APtr;
Aptr=A;
for(i=0;i<5;i++){
   S=S+*(APtr);
   APtr++;
 }
```

Increment address (value of Aptr) by 4 each time

Array and Pointer

```
int  i;
char A[5],S=0;
char *APtr;

for(i=0;i<5;i++){
   S=S+A[i];
}
```

```
int  i;
char A[5],S=0;
char *APtr;
Aptr=A;
for(i=0;i<5;i++){
   S=S+*(APtr);
   APtr++;
}
```

Increment address (value of Aptr) by 1 each time

Array and Pointer

```
int  i;
long A[5],S=0;
long *APtr;

for(i=0;i<5;i++){
   S=S+A[i];
}
```

```
int  i;
long A[5],S=0;
long *APtr;
Aptr=A;
for(i=0;i<5;i++){
   S=S+*(APtr);
   APtr++;
}
```

Increment address (value of Aptr) by 8 each time

Pointer Arithmetic

```
int *p, *q;
q = p + 1;
```

- Construct a pointer to the next *integer* after ***p** and assign it to **q**

```
double *p, *r;
int n;
r = p + n;
```

- Construct a pointer to a *double* that is **n** *doubles* beyond ***p**, and assign it to **r**
- **n** may be negative

Pointer Arithmetic (continued)

```
long int *p, *q;
p++; q--;
```

- Increment **p** to point to the next **long int**; decrement **q** to point to the previous **long int**

```
float *p, *q;
int n;
n = p - q;
```

- **n** is the number of floats between ***p** and ***q**; i.e., what would be added to **q** to get **p**

Pointer Expressions and Pointer Arithmetic

- Arithmetic operations can be performed on pointers
  - Increment/decrement pointer (**++** or **- -**)
  - Add an integer to a pointer(**+** or **+=** , **-** or **-=**)
  - Pointers may be subtracted from each other
  - Operations meaningless unless performed on an array

Arrays and Pointers

- **`double A[10];`** *vs.* **`double *A;`**
- *Only* difference:–
  - **`double A[10]`** sets aside *ten* units of memory, each large enough to hold a **`double`**
  - **`double *A`** sets aside *one* pointer-sized unit of memory
    - You are expected to come up with the memory elsewhere!
  - Note:– all pointer variables are the same size in any given machine architecture
    - Regardless of what types they point to

Array-Array Assignment

- *C* does *not* assign arrays to each other
- *E.g,*
  - `double A[10], B[10];`

```
A=B;//Not a valid Statement
```

    - assigns the pointer value **B** to the pointer value **A**
    - Contents of array **A** are untouched

Function

Introduction

- Divide and conquer
  - Construct a program from smaller pieces or components
  - Each piece more manageable than the original program
- Functions
  - Modules in C
  - Programs written by combining user-defined functions with library functions
    - C standard library has a wide variety of functions
    - Makes programmer's job easier - avoid reinventing the wheel

Function calls

- Invoking functions
  - Provide function name and arguments (data)
  - Function performs operations or manipulations
  - Function returns results
- Boss asks worker to complete task
  - Worker gets information, does task, returns result
  - Information hiding: boss does not know details

Math Library Functions

- Math library functions
  - perform common mathematical calculations
  - **`#include <math.h>`**
- Format for calling functions

  `FunctionName (argument);`
    - If multiple arguments, use comma-separated list
  - **`printf( "%.2f", sqrt( 900.0 ) );`**
    - Calls function **`sqrt`**, which returns the square root of its argument
    - All math functions return data type **`double`**
  - Arguments may be constants, variables, or expressions

Math Library Functions

- double sin(double x) double cos(double x),
- double pow(double x, double n)
- long powl(long x, long n)
- float powf(float x, float n)
- double ceil(double x), double exp(double x)
- double acos(double x), double asin(double x)
- ……list continues…….

It is not good idea to remember all math functions .

Use man page

$man math.h

$man pow

$man sin

Std Library Functions and others

- double rand()
- void exit()
- int atoi()
- Long atol()

- ……list continues…….

It is not good idea to remember all stdlib functions .

Use man page

$man stdlib.h

$man rand

$man random

Functions

- Modularize a program
- All variables declared inside functions are local variables : Known only in function defined
- Parameters: Communicate info. between functions
- Function Benefits
  - Divide and conquer : Manageable program development
  - Software reusability : Use existing functions as building blocks for new programs and
  - Abstraction : hide internal details (library functions)
  - Avoids code repetition

Function Definitions

- Function definition format

```
return-value-type function-name
    (parameter-list ) {
     declarations and
statements
}
```

- Function-name: any valid identifier
- Return-value-type: data type of the result (default **int**)
  - —**void** - function returns nothing
- Parameter-list: comma separated list, declares parameters (default **int**)

Function Definitions

- Function definition format

```
return-value-type function-name
    ( parameter-list ) {
     declarations and statements
}
```

- Declarations and statements: function body (block)
  - Variables can be declared inside blocks (can be nested)
  - In C++, Function can not be defined inside another function : **But in c it is allowed**
- Returning control
  - If nothing returned : **return;**
    - or, until reaches right brace
  - If something returned : return *expression;*

Review of Structured Programming

- Structured programming is a problem solving strategy and a programming methodology that includes the following guidelines
- The program uses only the sequence, selection, and repetition control structures.
- The flow of control in the program should be as simple as possible.
- The **construction of a program embodies top-down design**.

Review of Top-Down Design

- Involves repeatedly **decomposing** a problem into smaller problems
- Eventually leads to a collection of
  - Small problems or tasks each of which can be easily coded
- The **function** construct in C is used
  - To write code for these small, simple problems.

Functions

- A C program is made up of one or more functions, one of which is main( ).
- Execution always begins with main( )
  - No matter where it is placed in the program.
- main( ) is located before all other functions.
- When program control encounters a function name, the function is **called** (**invoked**).
  1. Program control passes to the function.
  2. The function is executed.
  3. Control is passed back to the calling function.

Sample Function Call

```
#include <stdio.h>
 int main ( ) {

  printf("Hello World!\n");
  return 0 ;
}
```

printf is the name of a **predefined function** in the stdio library

this statement is is known as a **function call**

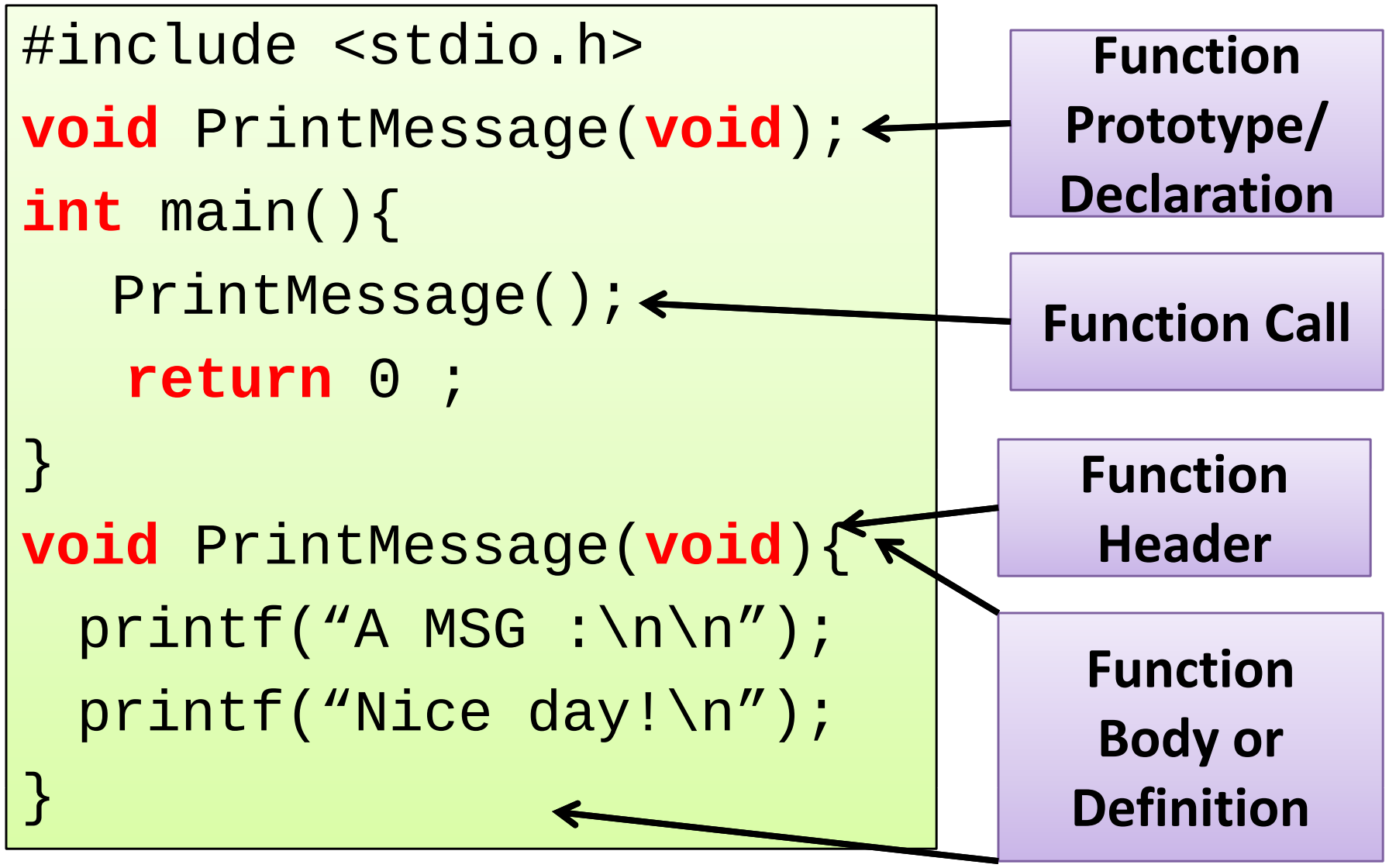
this is a string we are **passing** as an **argument** (**parameter**) to the printf function

Functions (con't)

- We have used three predefined functions so far:
  - –printf, scanf, pow, sqrt, abs, sin, cos
- Programmers can write their own functions.
- Typically, each module in a program's design hierarchy chart is implemented as a function.

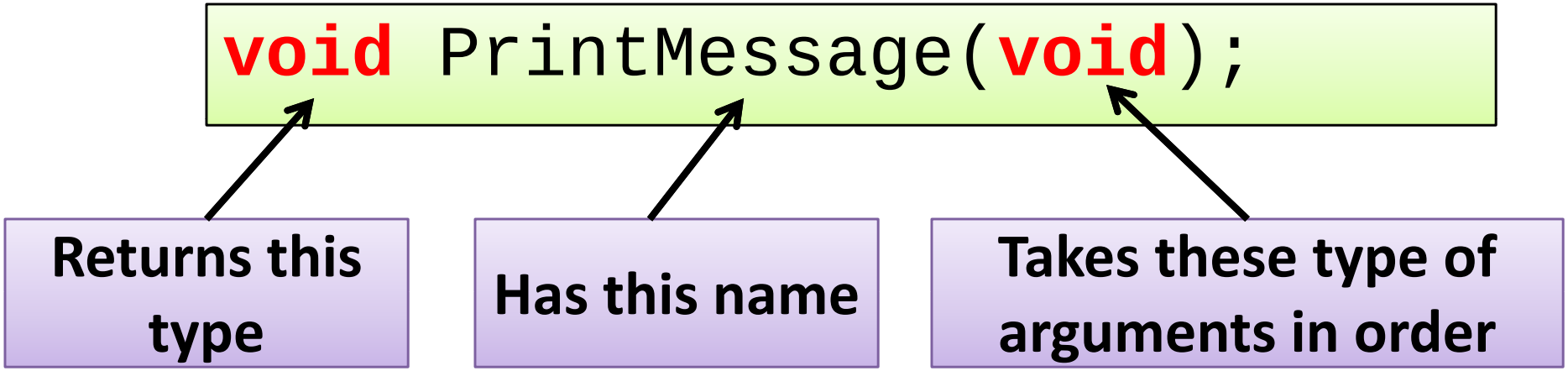
Sample -Defined Function

```
#include <stdio.h>
void PrintMessage(void);
int main(){
   PrintMessage();
    return 0 ;
}
void PrintMessage(void){
  printf("A MSG :\n\n");
  printf("Nice day!\n");
}
```



The Function Prototype

- Informs the compiler that there will be a function defined later that:

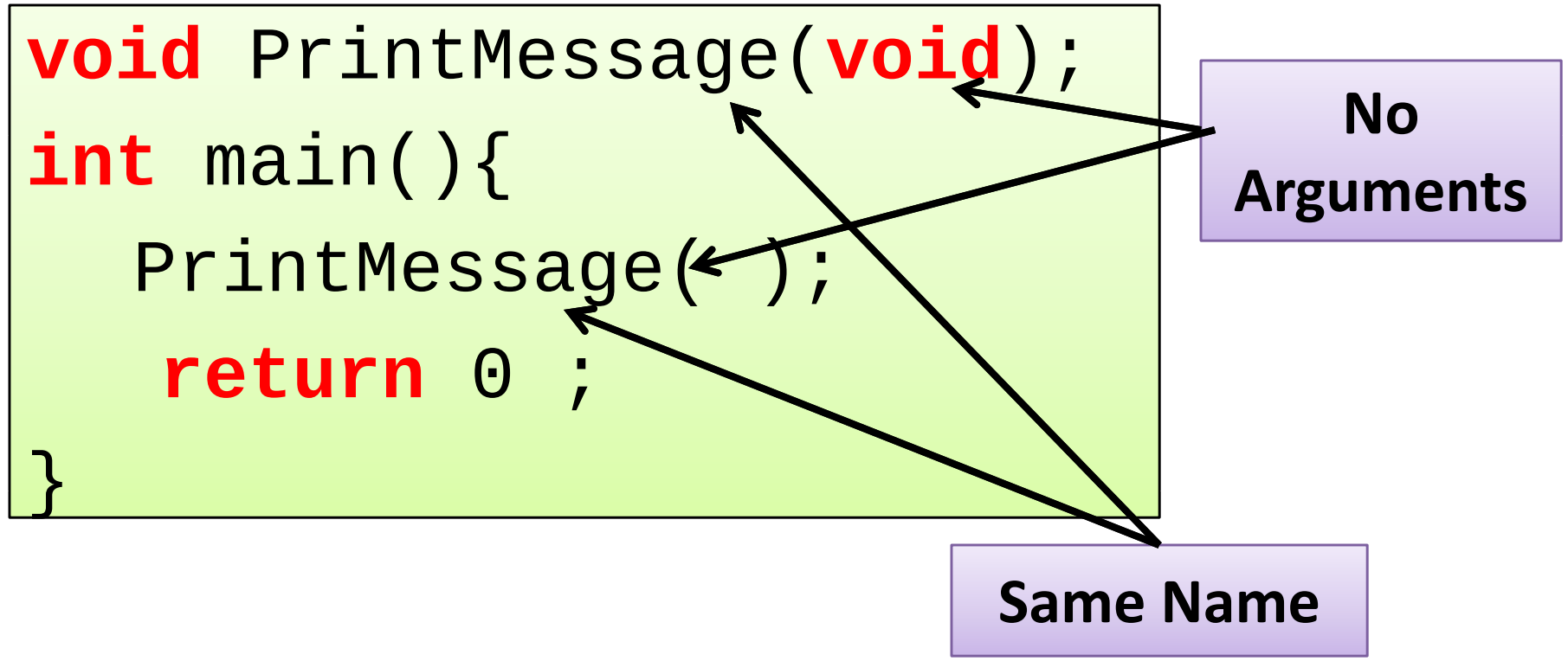


- Needed because the function call is made before the definition -- the compiler uses it to see if the call is made properly

The Function Call

- Passes program control to the function
- Must match the prototype in name, number of arguments, and types of arguments

```
void PrintMessage(void);
int main(){
   PrintMessage( );
    return 0 ;
}
```



Thanks