The C Programming Language – Part 3

(with material from Dr. Bin Ren, William & Mary Computer Science, and www.cpp.com)

Overview

- Functions
- Parameter Passing
- Arrays

Functions

```
#include <stdio.h>

int addition (int a, int b)
{
    int r;

    r = a + b;
    return r;
}

int main ()
{
    int z;

    z = addition (5, 3);
    printf ("The result is %d\n", z);
}
```

```
The result is 8
```

Functions

```
#include <stdio.h>

int subtraction (int a, int b)
{
    int r;
    r = a - b;
    return r;
}

int main ()
{
    int x = 5, y = 3, z;

    z = subtraction (7, 2);
    printf ("The first result is %d\n", z);
    printf ("The second result is %d\n", subtraction (7, 2));
    printf ("The third result is %d\n", subtraction (x, y));

    z = 4 + subtraction (x, y);
    printf ("fourth result is %d\n", z);
}
```

```
The first result is 5
The second result is 5
The third result is 2
The fourth result is 6
```

Functions

```
// void function example
#include <stdio.h>

void print_message ()
{
    printf ("I'm a function!\n");
}

int main ()
{
   print_message ();
}
```

```
I'm a function!
```

Function Prototypes

```
Please, enter number (0 to exit): 9
It is odd.
Please, enter number (0 to exit): 6
It is even.
Please, enter number (0 to exit): 1030
It is even.
Please, enter number (0 to exit): 0
It is even.
```

```
#include <stdio.h>

void odd (int x);
void even (int x);

int main ()
{
    int i;
    do {
       printf ("Please enter number (0 to exit): ");
       scanf ("%d", &i);
       odd (i);
    } while (i != 0);
}

void odd (int x)
{
    if ((x % 2) != 0) printf ("It is odd.\n");
    else even (x);
}

void even (int x)
{
    if ((x % 2) == 0) printf ("It is even.\n");
    else odd (x);
}
```

Recursion

```
#include <stdio.h>

long factorial (long a)
{
    if (a > 1)
        return (a * factorial (a – 1));
    else
        return 1;
}

int main ()
{
    long num = 9;

    printf ("%d! = %d\n", num, factorial (num));
}
```

```
9! = 362880
```

Parameter Passing

- **parameters can be passed to functions**
  - by value
    - copy of value passed
    - no special syntax needed
    - changes only valid within function
  - by reference
    - address of variable passed
    - must use pointers for arguments and formal parameters
    - changes made with function reflected after retrun

Pass by Value

```
#include <stdio.h>

void duplicate (int a, int b, int c)
{
    a *= 2;
    b *= 2;
    c *= 2;

    printf ("a = %d, b = %d, c = %d\n", a, b, c);
}

int main (void)
{
    int x = 1, y = 3, z = 7;

    duplicate (x, y, z);
    printf ("x = %d, y = %d, z = %d\n", x, y, z);
}
```

```
a = 2, b = 6, c = 14
x = 1, y = 3, z = 7
```

Pass by Reference

```
#include <stdio.h>

void duplicate (int *a, int *b, int *c)
{
    *a *= 2;
    *b *= 2;
    *c *= 2;

    printf ("a = %d, b = %d, c = %d\n", *a, *b, *c);
}

int main (void)
{
    int x = 1, y = 3, z = 7;

    duplicate (&x, &y, &z);
    printf ("x = %d, y = %d, z = %d\n", x, y, z);
}
```

```
a = 2, b = 6, c = 14
x = 2, y = 6, z = 14
```

Scope

- **global vs. local variables**
  - avoid global variables
  - always use local variables and pass as parameter when needed

```
int foo;      // global variable

int some_function ()
{
    int bar;
    bar = 0;
}

int other_function ()
{
    foo = 1;   // OK: foo is a global variable
    bar = 2;   // not OK: bar is not visible here
}
```

Scope

- name can only represent one entity

```
int some_function ()
{
    int x;
    x = 0;

    double x;   // wrong: name already used in this scope
    x = 0.0;
}
```

Scope

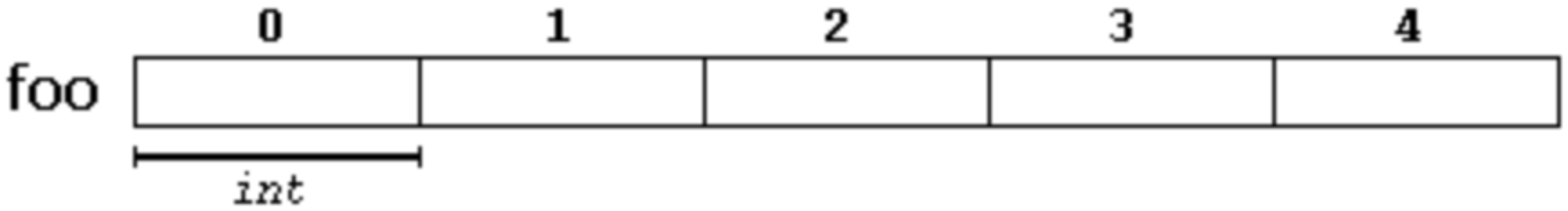
- block scope

```
inner block:
x: 50
y: 50
outer block:
x: 10
y: 50
```

```
#include <stdio.h>

int main ()
{
    int x = 10;
    int y = 20;

    {
        x = 50;
        y = 50;
        printf ("inner block:\n");
        printf ("x: %d\n", x);
        printf ("y: %d\n", y);
    }
    printf ("outer block:\n");
    printf ("x: %d\n", x);
    printf ("y: %d\n", y);
}
```

Arrays

- contiguous memory locations

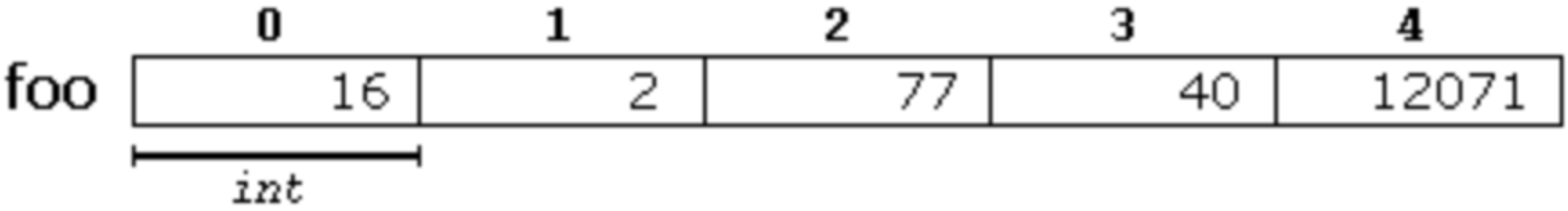



```
int foo [5];
```

Arrays

- elements not automatically initialized, but can be explicitly initialized

```
int foo [5] = { 16, 2, 77, 40, 12071 };
```



Arrays

- initialized arrays without size are automatically sized to accommodate values

```
int foo [] = { 16, 2, 77, 40, 12071 };
```

- no error if range exceeded
- example uses of arrays

```
1 foo[0] = a;
2 foo[a] = 75;
3 b = foo [a+2];
4 foo[foo[a]] = foo[2] + 5;
```

Arrays

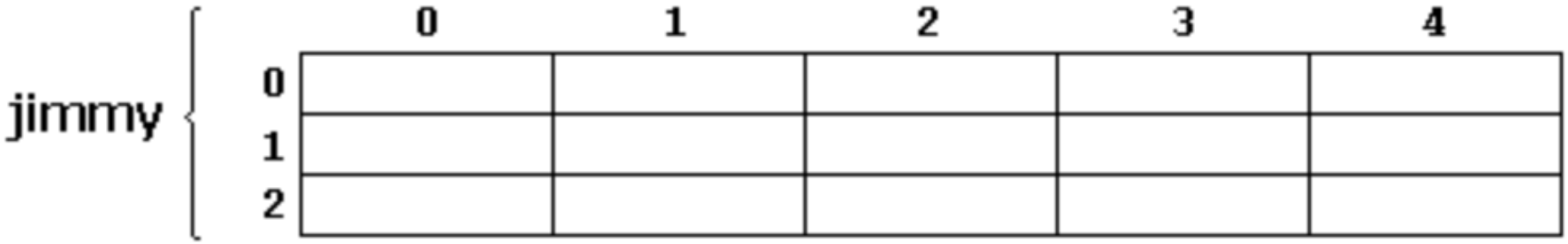
```
#include <stdio.h>

int main (void)
{
    int foo [] = {16, 2, 77, 40, 12071};
    int n, result = 0;

    for (n = 0; n < 5; n++)
    {
        result += foo [n];
    }

    printf ("%d\n", result);
}
```

```
12206
```

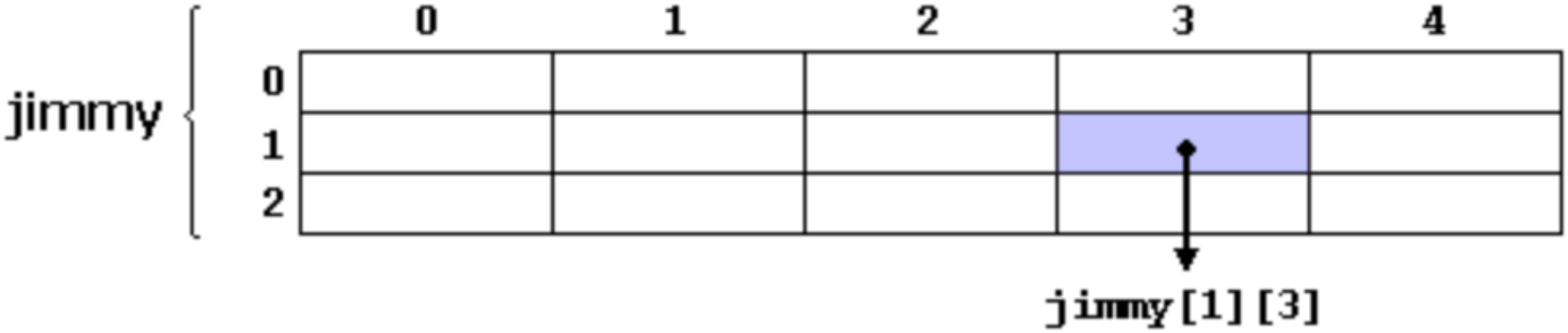
Multidimensional Arrays

- arrays of arrays

```
int jimmy [3][5];
```




```
jimmy[1][3]
```



Multidimensional Arrays

- **can be any dimension, but space increases exponentially**

```
char century [100][365][24][60][60];
```

  - allocates a char for each second in the last century
  - consumes 3GB of memory

- **could implement as a single-dimension array**

```
1 int jimmy [3][5];    // is equivalent to
2 int jimmy [15];      // (3 * 5 = 15)
```

Multidimensional Arrays

multidimensional array	pseudo-multidimensional array

`#define WIDTH 5`
`#define HEIGHT 3`
 	
`int jimmy [HEIGHT][WIDTH];`
`int n,m;`
 	
`int main ()`
`{`
`  for (n=0; n<HEIGHT; n++)`
`  for (m=0; m<WIDTH; m++)`
`  {`
`  jimmy[n][m]=(n+1)*(m+1);`
`  }`
`}`
	`#define WIDTH 5`
	`#define HEIGHT 3`
	
	`int jimmy [HEIGHT * WIDTH];`
	`int n,m;`
	
	`int main ()`
	`{`
	`  for (n=0; n<HEIGHT; n++)`
	`  for (m=0; m<WIDTH; m++)`
	`  {`
	`  jimmy[n*WIDTH+m]=(n+1)*(m+1);`
	`  }`
	`}`

jimmy

	0	1	2	3	4
0	1	2	3	4	5
1	2	4	6	8	10
2	3	6	9	12	15

Arrays as Parameters

```
#include <stdio.h>

void print_array (int array [], int length)
{
    int n;

    for (n = 0; n < length; n++)
        printf ("%d ", array [n]);
    printf ("\n");
}

int main (void)
{
    int first [] = {5, 10, 15};
    int second [] = {2, 4, 6, 8, 10};

    print_array (first, 3);
    print_array (second, 5);
}
```

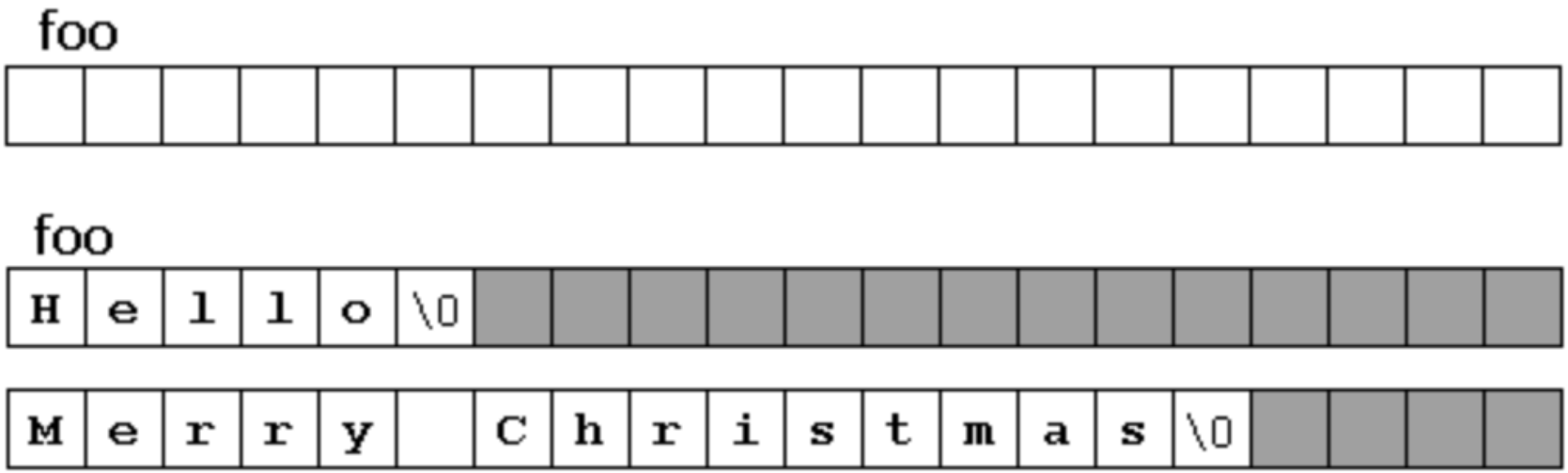
```
5 10 15
2 4 6 8 10
```

```
void procedure (int myarray[][3][4])
```

Character Arrays

- space set aside when declared, but not initialized
- must end with '\0' character

```
char foo [20];
```



Character Arrays

- can initialize with individual elements or string literals

```
1 char myword[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
2 char myword[] = "Hello";
```

- not valid

```
1 myword = "Bye";
2 myword[] = "Bye";
```

```
myword = { 'B', 'y', 'e', '\0' };
```

- OK

```
1 myword[0] = 'B';
2 myword[1] = 'y';
3 myword[2] = 'e';
4 myword[3] = '\0';
```

Preview of Strings

```
#include <stdio.h>
#include <string.h>

void main ()
{
   char input [10];
   char yn;
   char valid_input [3] = "YN";

   printf ("Please enter Y or N: ");
   scanf ("%s", input);
   yn = toupper (input [0]);

   if (strlen (input) > 1 || (yn != 'Y' && yn != 'N')) {
      printf ("bad input: %s or option: %c\n", input, yn);
   }

   // another way to check input
   if (strlen (input) > 1 || !strchr (valid_input, yn)) {
      printf ("bad input: %s or option: %c\n", input, yn);
   }
}
```