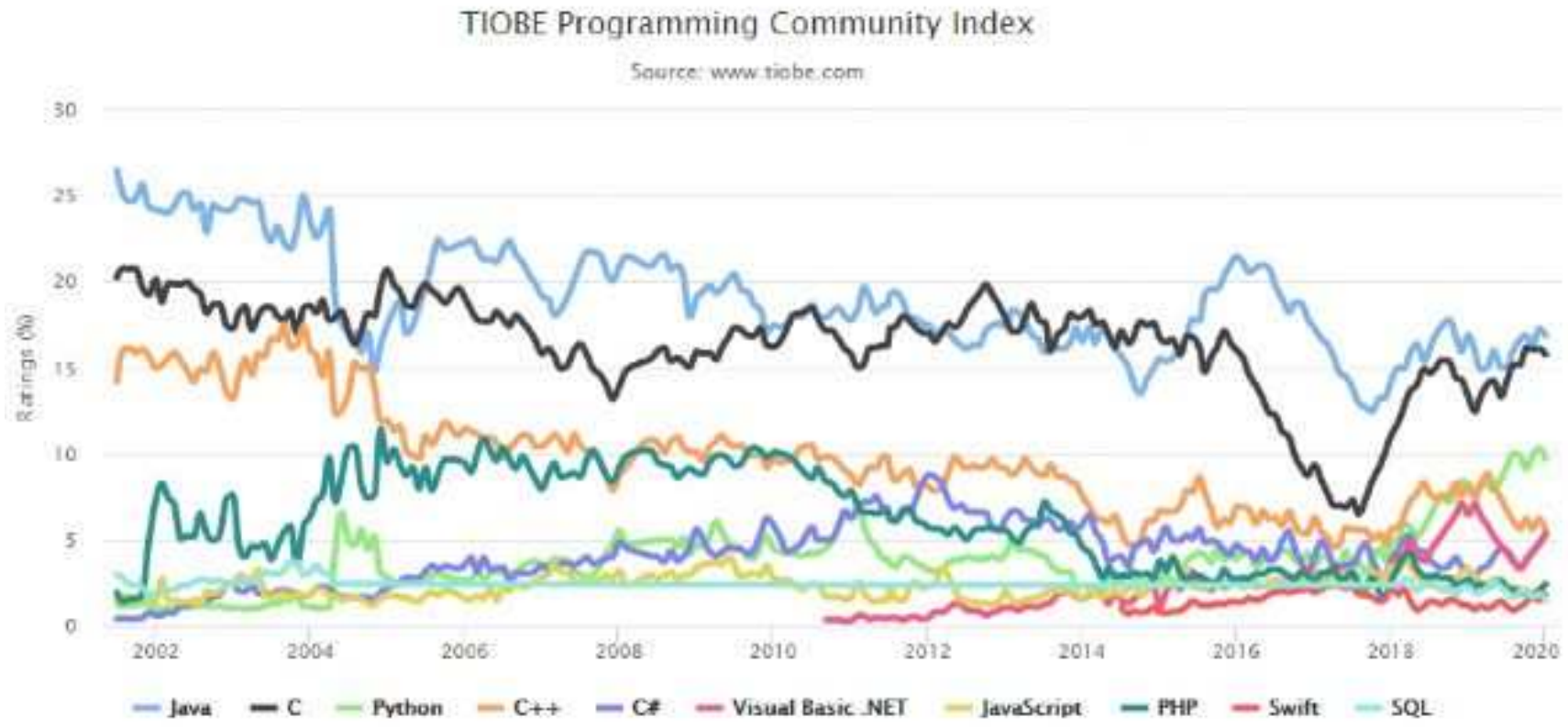
Quick Introduction to C

경희대학교 컴퓨터공학과

조 진 성

C Programming Language

1972, Kenneth Thompson & Dennis Ritchie



The First C Program in Your Life

```
#include <stdio.h>

main()
{
  printf("Hello, World!\n");
}
```

1978, Brian Kernighan & Dennis Ritchie

Standard I/O: Output Function

```
#include <stdio.h>

main()
{
   char c = 'a', s[] = "hello";
   int i = 100; long l = 99999;
   float f = 3.14; double d = 99.999;
   int *p = &i;

   printf("Output: %c %s %d %#X %ld %.4f %.2lf %p\n",
           c, s, i, i, l, f, d, p);
   putchar(c);
   puts(s);
}
```


Standard I/O: Input Function

```
#include <stdio.h>

main()
{
   char c, s[80];
   int i; long l;
   float f; double d;

   scanf("%c %s %d %ld %f %lf", &c, s, &i, &l, &f, &d);
   printf("Output: %c %s %d %ld %.4f %.2lf %p\n",
            c, s, i, l, f, d, p);
   c = getchar();
   putchar(c);
   gets(s);
   puts(s);
}
```


Condition

Not zero → True

Zero → False

```
enum { FALSE, TRUE };
```

a == b

a != b

a > b a >= b

a < b a <= b

a && b

a || b

! a

if - else

```
#include <stdio.h>
main()
{
   int num = -1;

   if (num > 0)
    printf("positive");
   else if (num < 0)
    printf("negative");
   else
    printf("zero");

   if (num)
    printf("true");
   else
    printf("false");
}
```


switch - case

```
#include <stdio.h>

main()
{
   char ch = 'z';

   switch (ch)  {
   case 'A':
   case 'B': printf("Upper-case\n"); break;
   case 'a':
   case 'b': printf("Lower-case\n"); break;
   default: puts("unknown"); break;
   }
}
```


for loop

```
#include <stdio.h>

main()
{
   int i, sum = 0;
   for (i = 0 ; i <= 100 ; i++)   {
     sum += i;
   }
}
```


while loop

```
#include <stdio.h>

main()
{
   int i = 0, sum = 0;
   while (i <= 100)   {
     sum += i++;
   }
}
```


do-while loop

```
#include <stdio.h>

main()
{
   int i = 0, sum = 0;
   do  {
     sum += ++i;
   }
   while (i <= 100);
}
```


break & continue

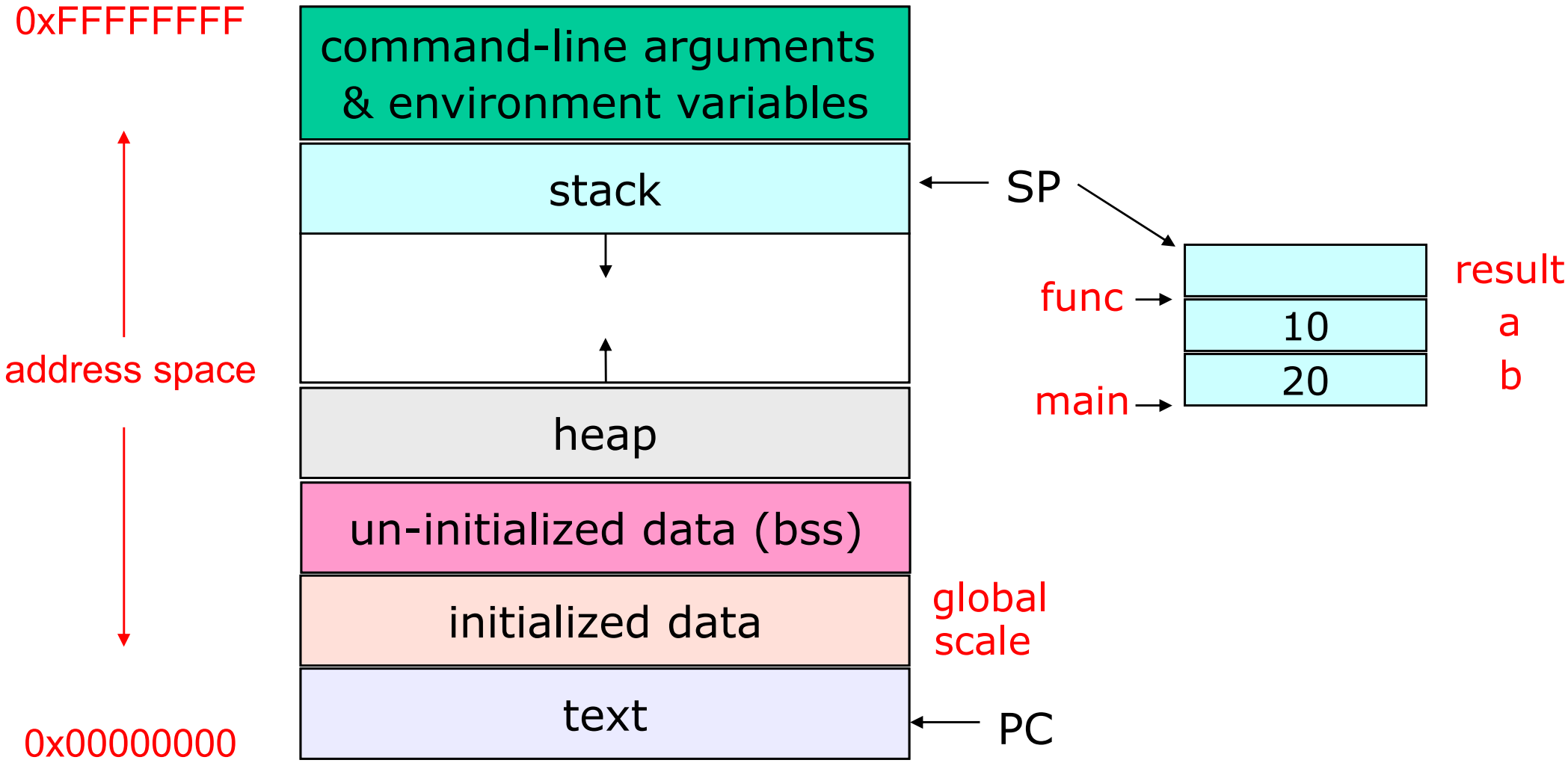
```
#include <stdio.h>

main()
{
   int i;
   for (i = 0; i < 100 ; i++)   {
    if (i % 2)
        continue;
    printf("here\n");
   }
   i = 0;
   while (i < 100)   {
    if (DoSomething(i++) < 0)
        break;
   }
}
```


Variables & Arguments(Parameters)

Local vs. Global & Automatic vs. Static

```
#include <stdio.h>
int global = 2;
func(int a, int b)
{
  static int scale = 1;
  int result = (a + b) * global * scale;
  scale *= 10;
  return result;
}
main()
{
  printf("result=%d\n", func(10, 20));
  printf("result=%d\n", func(10, 20));
}
```


Memory Layout of a Process



Command-line Arguments

```
$ ./mycp mycp.c yourcp.c
```

```
argv[0] = "./mycp"
argv[1] = "mycp.c"
argv[2] = "yourcp.c"
argv[3] = ""
```

```
#include <stdio.h>
main(int argc, char *argv[])
{
  if (argc !=3)  {
    printf("Usage: %s src dst\n", argv[0]);
    exit(1);
  }
  /* copy argv[1] to argv[2] */
}
```

```
$ ./a.out *
```

```
#include <stdio.h>
main(int argc, char *argv[])
{
  int i;
  for (i = 0 ; i < argc ; i++)
    printf("argv[%d]=%s\n", i, argv[i]);
}
```


Dynamic Memory Management

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

main()
{
  int *p = (int *)malloc(100*sizeof(int));
  if (! p)  {
   printf("Not enough memory!\n");
   exit(1);
  }
  p[0] = 1; p[1] = 2; /* do something */
  free(p);
}
```


Pointers

```c
#include <stdio.h>

main()
{
  int a = 10;
  int *p;
  p = &a;
  printf("a=%d/%d\n", a, *p);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
main()
{
  int *p = (int *)
      malloc(sizeof(int));
  *p = 10;
  printf("%d\n", *p);

  /* 1) dangling pointer */
  free(p);
  printf("%d\n", *p);
```

or

```c
  /* 2) garbage */
  p = (int *)
      malloc(sizeof(int));
}
```


Call-by-Value vs. Call-by-Reference

```
#include <stdio.h>

square(int a)
{
   a *= a;
}

main()
{
   int a = 10;
   square(a);
   printf("a=%d\n", a);
}
```

```
#include <stdio.h>

square(int *a)
{
   (*a) *= (*a);
}

main()
{
   int a = 10;
   square(&a);
   printf("a=%d\n", a);
}
```


Pointer & Array

```
#include <stdio.h>

square(int a[], int num)
{
  int i;

  for (i = 0 ; i < num ; i++)
    a[i] *= a[i];
}

main()
{
  int a[5] =
       { 1, 2, 3, 4, 5 };
  square(a, 5);
}
```

```
#include <stdio.h>
square(int *a, int num)
{
  int i;
  for (i = 0 ; i < num ; i++)
    a[i] *= a[i];
```

or

```
  for (i = 0 ; i < num ; i++)
  {
    (*(a+i)) *= (*(a+i));
  }
```

or

```
  for (i = 0 ; i < num ; i++)
  {
    (*a) *= (*a);
    a++;
  }
}
```


Common Errors with Pointers

```
#include <stdio.h>
square(double data[], int num)
{
   int i;
   for (i = 0 ; i < num ; i++)
     data[i] *= data[i];
}
main()
{
   int *p;
   char *str;
   double *data;

   *p = 10;
   strcpy(str, "hello");
   square(data, 5);
}
```

Run-time error !!!

String Manipulation

String in C (& C++)

- ✓ Null-terminated one-dimensional character array

```
#include <string.h>
char char_array[] = {'C','o','m','p','u','t','e','r','\0'};
char string_var[] = "Kyung Hee University";
char string_var2[80] = "Computer Engineering";
char *string_ptr = "Temporarily stored at heap";
```


String Manipulation (Cont'd)

Related C library

- ✓ `int strlen(char *s);`
  - return: length of s
- ✓ `char *strcpy(char *dest, char *src);`
  - return: dest
- ✓ `char *strcat(char *dest, char *src);`
  - return: dest
- ✓ `int strcmp(char *s1, char *s2);`
  - return: 0 if s1 == s2, positive value if s1 > s2, negative value otherwise

Common errors with string

```
char *str; strcpy(str, "Hello");
```

Run-time error !!!

```
char str[10]; strcpy(str, "12345678901234567890");
char str[] = "Hello"; strcat(str, ", World!");
```


String Manipulation (Cont'd)

String convert to integer

- ✓ `int atoi(char *s);`
  - return: an integer represented by s

```
char str[] = "10";
int i = 10;
int j = atoi(str);
```

String convert from integer

```
char str[80];
int i = 10;
sprintf(str, "%d", i);
sprintf(str, "%#X", 0x9f);
sprintf(str, "%f", 3.141592);
```


String Manipulation (Cont'd)

Character classification & convert

- ✓ `#include <ctype.h>`
- ✓ `int isalpha(int c);`
- ✓ `int isupper(int c); int islower(int c);`
- ✓ `int isdigit(int c); int isxdigit(int c);`
- ✓ `int isalnum(int c); int isspace(int c);`
- ✓ `int isascii(int c); int iscntrl(int c);`
- ✓ `int ispunct(int c); int isprint(int c);`
  - all return: nonzero for true, zero for false
- ✓ `int toupper(int c); int tolower(int c);`
  - each returns: upper/lower character of c if c is a character, c otherwise

Types

New type definition

```
typedef unsigned long size_t;
typedef unsigned char byte;
typedef struct {
  int year; int month; int day;
} date_t;
```

Type-cast

```
char *p = (char *)malloc(1000);
int  a = 2, b = 9;
double c = (double)a / b;
```


Bitwise Operator

AND → &

0xE8 & 0x7F

OR → |

0xE8 | 0x01

Exclusive OR → ^

(0xE8 ^ 0xE8) ^ 0xE8

Negation → ~

~(0xE8)

Left-shift → <<

0xE8 << 2

Right-shift → >>

0xE8 >> 1

Preprocessor

Include

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

Define

```
#define    TRUE          1
#define    FALSE         0
#define    MAX_BUF       100
#define    PI            3.141592654
#define    MIN(a,b)      (((a) < (b)) ? (a) : (b))
#define    SQUARE(x)     ((x) * (x))
```


Preprocessor (Cont'd)

Conditional compilation

```
#include <stdio.h>
#define DEBUG

main()
{
   .....
   #ifdef     DEBUG
    printf("sum=%d\n", sum);
   #else
    printf("Total %d won\n",
            sum);
   #endif
   .....
}
```

or `gcc -DDEBUG`

```
#include <stdio.h>
#define SOLARIS 0
#define LINUX 1
#define MACHINE LINUX

main()
{
   .....
   #if MACHINE == LINUX
    clone();
   #else
    fork()
   #endif
   .....
}
```

or `gcc -DMACHINE=LINUX`

Exercise

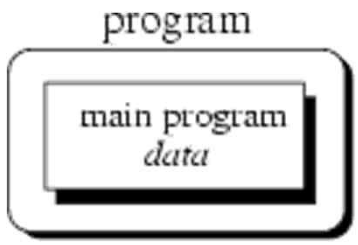
String manipulation examples

```
$ gcc -o string string.c (or make string)
$ ./string
```


Programming Techniques

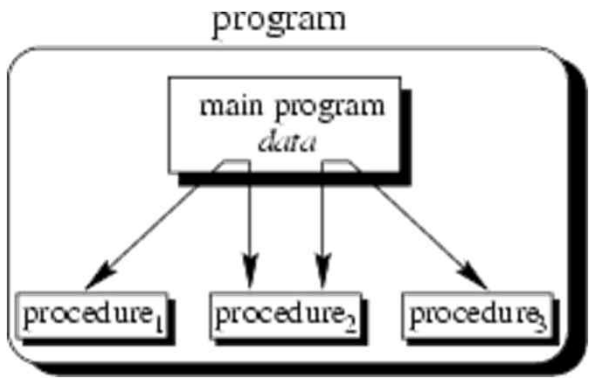
Unstructured Programming

- ✓ Writing small and simple programs consisting only of one main program



Procedural (Structured) Programming

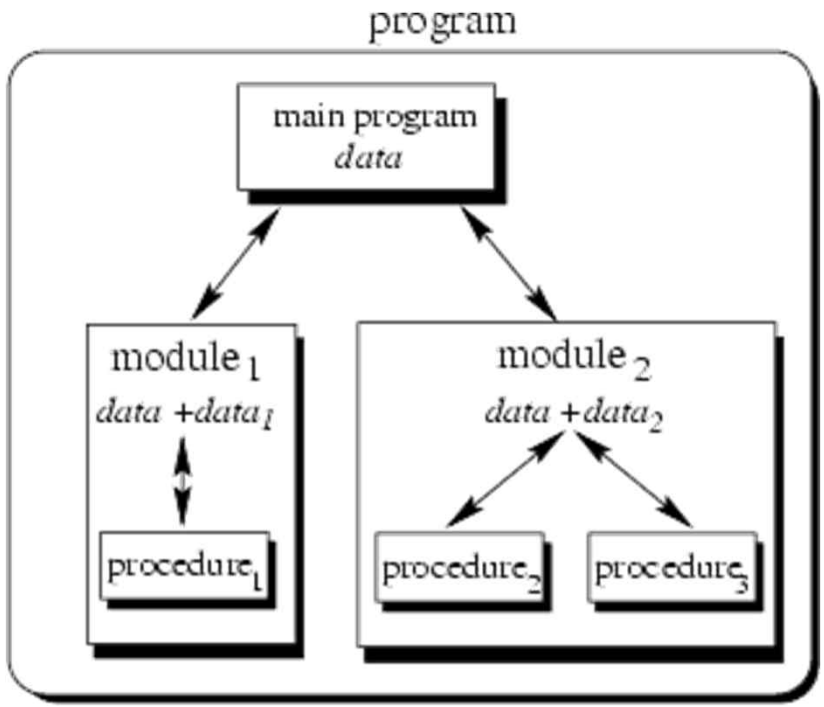
- ✓ Writing a sequence of procedures (or functions)



Programming Techniques (Cont'd)

Modular Programming

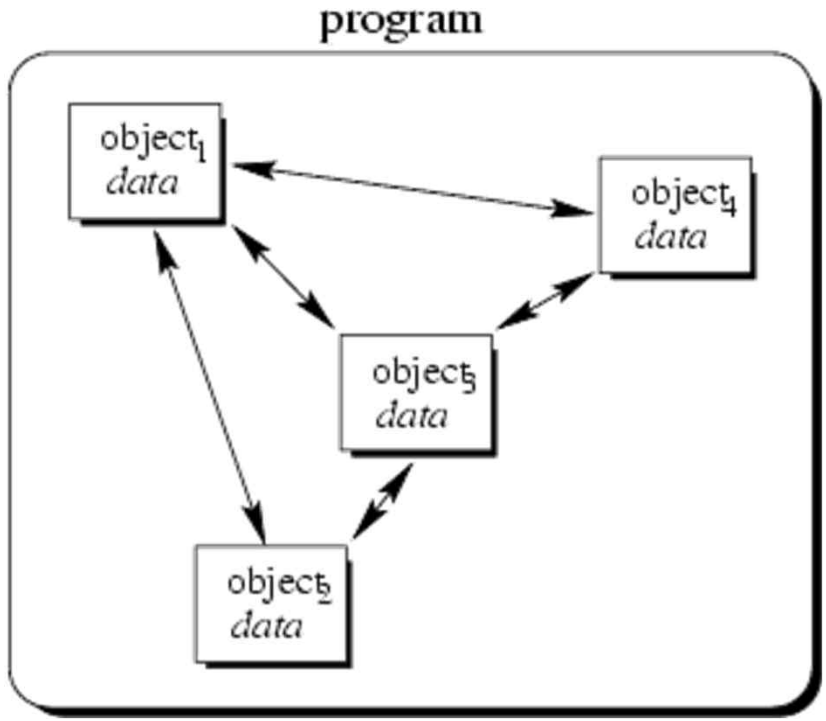
- ✓ Grouping procedures into modules
- ✓ Each module may have its own state and data



Programming Techniques (Cont'd)

Object-Oriented Programming

- ✓ Writing objects which interact with each other
- ✓ *So what? Isn't this just a more fancier modular programming technique?*



Programming Techniques (Cont'd)

Why Object-Oriented Programming?

- ✓ Code reusability
- ✓ Code readability
- ✓ Code reliability
- ✓ From Abstraction, Encapsulation, Inheritance, Polymorphism, etc.

Keep in mind these things, even though you are programming in C

Programming Techniques (Cont'd)

OO-like C program

[stack.h]

```
#define   SIZE   256

typedef struct {
  int stack[SIZE];
  int top;
} Stack;

int InitStack(Stack *);
int PushStack(Stack *, int);
int PopStack(Stack *);
int DestroyStack(Stack *);
```

[main.c]

```
#include "stack.h"
main()
{
  Stack stack;
  int data;

  InitStack(&stack);
  .....
  PushStack(&stack, 10);
  PushStack(&stack, 99);
  .....
  data = PopStack(&stack);
  .....
  DestroyStack(&stack);
}
```


Summary

Quick introduction to C

- ✓ Subset of C++

Keep in mind when programming in C

- ✓ OO-like C program