Introduction to Programming

C – More Input/Output, Structures, Enumeration, Preprocessor Directives

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

January, 2020

The getchar()

Taking a single character input from the user:

```
char c; // int c also works (use carefully)
c = getchar();
```

The getchar()

Taking a single character input from the user:

```
char c; // int c also works (use carefully)
c = getchar();
```

Taking a series of character inputs from the user:

```
char c; // int c also works (use carefully)
while ((c = getchar()) != EOF) {
    ...
}
```

More on getchar()

Reading a number from input using getchar():

```
int x;
char c;
while(isdigit(c = getchar()))
    x = x * 10 + c - '0';
```

Note: `isdigit()` is defined in the header file `ctype.h`.

Structures

Definition

A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling.

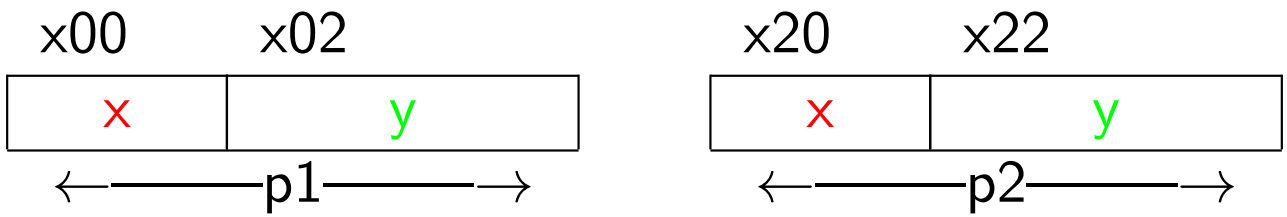
Example:

```
struct point{
    int x; // An integer field
    float y; // A floating point field
}p1, p2;

struct triangle{
    struct point a, b, c;
}t;
```

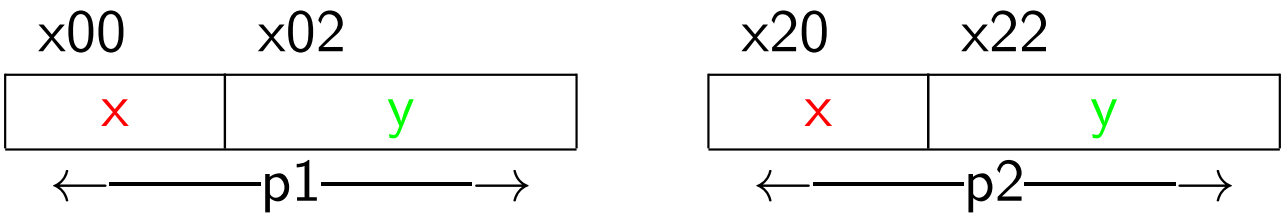
Structures

```
struct point{
    int x; // An integer field
    float y; // A floating point field
}p1, p2;
```

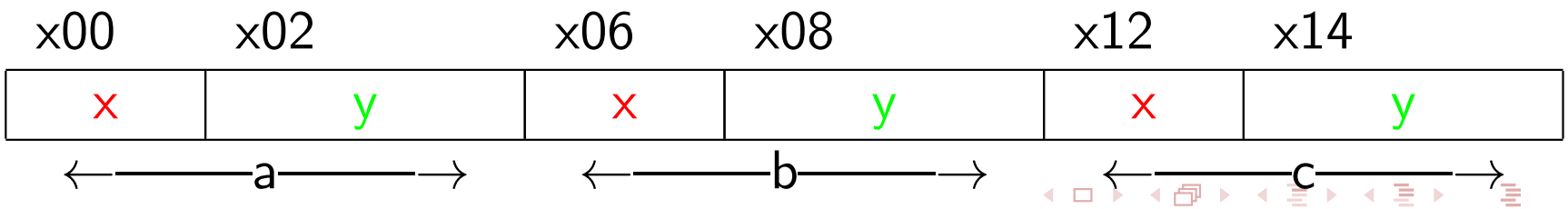


Structures

```
struct point{
    int x; // An integer field
    float y; // A floating point field
}p1, p2;
```




```
struct triangle{
    struct point a, b, c;
}t;
```



Operations on structures

- Assignment to members / fields
  `p1.x = 1; p1.y = 2.0; t.a.x = 0; t.a.y = 0.5;`
- Assignment / copying of structure variables
  `struct triangle t1, t2; ... ; t2 = t1;`
- Structures may be passed to functions, and returned by functions.

Note: Comparison operators (==, !=) do not work on structures.

Structure initialization

- Initialization

```
struct point{
    int x; // An integer field
    float y; // A floating point field
}p1, p2;

struct triangle{
    struct point a, b, c;
}t = {{1, 1.0},
      {-1, 1.0},
       {1, -1.0}};
```

Typedefs

```
typedef unsigned int UI;
UI i, j;                  // A pair of unsigned integers
UI myArray[10];           // An array of unsigned integers

typedef char *STR;
STR p;                    // A single string
STR myStrings[128];       // An array of strings

p = (STR) malloc(100);
UI strcmp(STR, STR); // Prototype declaration
```

Typedefs

```
typedef struct {
    int x;
    float y;
}POINT;
POINT p1, p2;

typedef struct {
    struct point a, b, c;
}TRIANGLE;
TRIANGLE t;
```

Pointers to structures

```
struct point *pp; // Old scheme (before typedef)
POINT *pp;        // New scheme (after typedef)

(*pp).x = 10; (*pp).y = 1.414; // Syntax 1
pp->x =10; pp->y = 1.414;      // Syntax 2
```

Memory allocation

```
TRIANGLE *tp;

tp = (TRIANGLE *)malloc(num_triangles*sizeof(TRIANGLE));
```

Enumeration (enum)

We can use enum to declare new enumeration types and define their values.

```
enum MONTH{January, February, March, April, May, June,
  July, August, September, October, November, December};
int main(){
    enum MONTH mi;
    mi = August; // Index of August is assigned to mi
    printf("%d", mi); // Prints 7
    return 0;
}
```

Basics

Definition (What is a preprocessor?)

Before a C program is compiled by a compiler, the source code is processed by a program called preprocessor. This process is called preprocessing.

Definition (What are preprocessor directives?)

The commands used in preprocessor are defined as preprocessor directives and they begin with '#' symbol.

Predefined macros

```
#include<stdio.h>
int main(){
    printf("Current date: %s\n", __DATE__);
    printf("Current time: %s\n", __TIME__);
    printf("\# Lines in the code: %d\n", __LINE__);
    printf("File name: %s\n", __FILE__ );
    printf("C Standard: %d\n", __STDC_VERSION__);
    printf("Compilation was successful: %d\n", __STDC__);
}
```

Predefined macros

```
#include<stdio.h>
int main(){
    printf("Current date: %s\n", __DATE__);
    printf("Current time: %s\n", __TIME__);
    printf("\# Lines in the code: %d\n", __LINE__);
    printf("File name: %s\n", __FILE__ );
    printf("C Standard: %d\n", __STDC_VERSION__);
    printf("Compilation was successful: %d\n", __STDC__);
}
```

Note: The `__STDC_VERSION__` value 199409L signifies the C89 standard, 199901L signifies the C99 standard, and 201112L signifies the C11 standard.

#include

This preprocessor directive is used to include system-defined and user-defined files into the current file. There are two different ways of doing this as shown below.

```
#include <filename>
```

This is used to include system header files. It first searches for a file in a list of directories specified by you, then in a standard list of system directories.

```
#include "filename"
```

This is used include user-defined contents. It first searches for a file in the current directory, then in a standard list of system directories.

#define

There are two different types of macros as listed below.

1. **Object-like macros:**

```
#define e 2.71828
```

2. **Function-like macros:**

```
#define COMPARE(x, y) (((x) > (y)) - ((x) < (y)))
```

#define

What is the output of the following C program?

```
#include<stdio.h>
#define RECIPROCAL(x) 1/x
int main(){
    int x = 1, y = 2, z;
    z = (x+y) * RECIPROCAL(x+y);
    printf("%d", z);
    return 0;
}
```

#define (also known as macros)

Multi-line macros can be accommodated by inserting line separators at the end of each line.

```
#define ASCII(character) {\
    printf("The ASCII value of ");\
    printf("%c", character);\
    printf(" is ");\
    printf("%d", character);\
}
```

#undef

The definition of a macro can be dropped as follows.

```
#undef COMPARE(x, y)
```

#ifdef

We can check whether a macro is already defined as follows.

```
#ifdef COMPARE(x, y)
    // Code segment
#endif
```

Note: If the definition exists, it will execute the code segment.

#ifndef

We can check whether a macro is not yet defined as follows.

```
#ifndef COMPARE(x, y)
    // code segment
#endif
```

Note: If the definition does not exist, it will execute the code segment.

#if and #endif

```
#if <expression>
    // code segment
#endif
```

#else

```
#if <expression>
    // if code segment
#else
    // else code segment
#endif
```

#elif

```
#if expression
    // if code segment
#elif expression
    // elif code segment
#else
    // else code segment
#endif
```

#error

```
#include<stdio.h>
#ifndef __MATH_H
#error Remember to include math.h!!!
#else
void main(){
    float f = 0.7;
    printf("%0.2f", pow(f, 2));
}
#endif
```