

The background features several large, overlapping, semi-transparent swirls in shades of light green, light blue, and light purple. Scattered throughout are numerous small, yellow, triangular shapes, some pointing upwards and some downwards, resembling confetti or starbursts.

C Programming Introduction

**Week 4: Variables,
constant, Standard input**



Topic of this week

- Variables

- Class Lecture Review

- Variables
 - Basic data types
 - Constants
 - Standard input.

- Programming Exercises

Identifiers

- Names of things (variables, functions, etc.)

```
int nMyPresentIncome = 0;  
int DownloadOrBuyCD();
```



Identifier naming rules

- Letters, digits, underscores
 - i
 - CSE_5a
 - a_very_long_name_that_isnt_very_useful
 - fahrenheit
- First character cannot be a digit
 - 5a_CSE is not valid!
- Case sensitive
 - CSE_5a is different from cse_5a



What are variables?

- A named area in the computer memory, intended to contain values of a certain kind (integers, real numbers, etc.)
- They contain the data your program works with
- They can be used to store data to be used elsewhere in the program
- In short – they are the only way to manipulate data



Variables

- Named region of storage

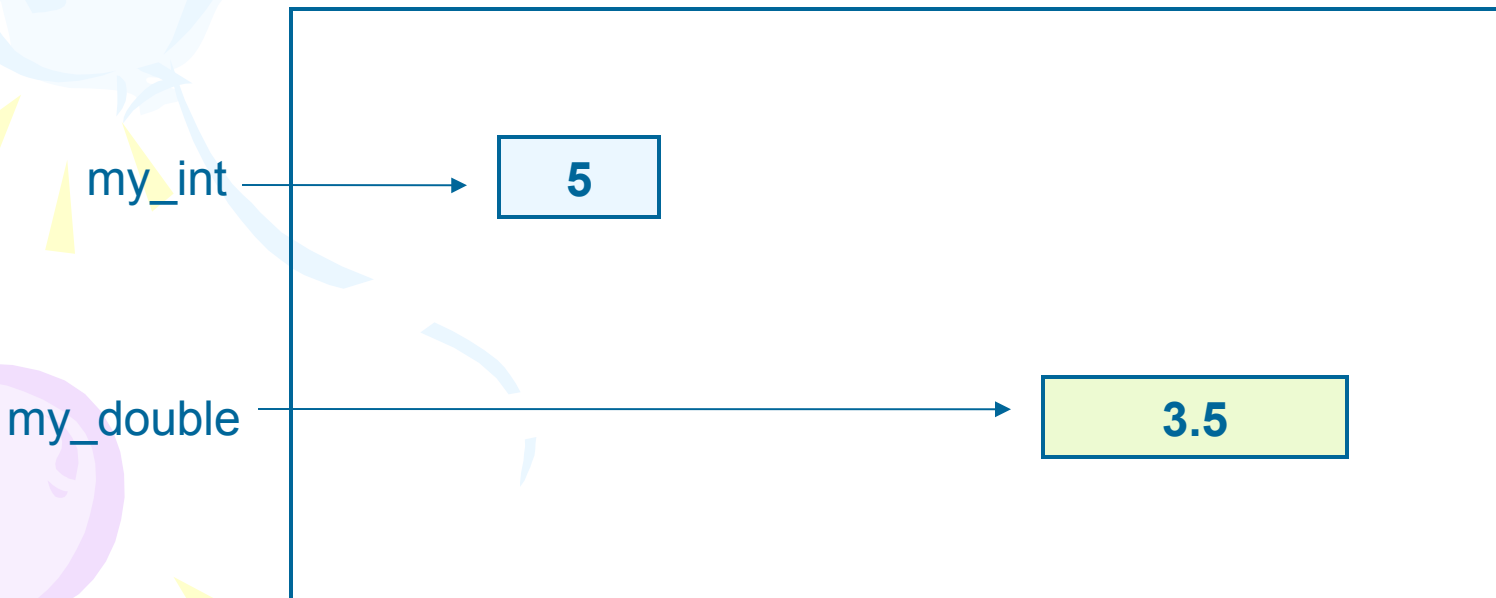
```
int nRow = 0;
```

- Type (size and meaning of the storage)
- Scope
 - Block
 - Function args
 - Global
 - Be careful not to “hide” a variable
- Lifetime (storage class)
 - Automatic/temporary (block's lifetime)
 - Globals (program's lifetime)
 - Local *static* (program's lifetime)

Variables in memory

```
int my_int = 5;
```

```
double my_double = 3.5;
```



Declarations, definitions, initialization

- Declarations that reserve storage are called definitions

```
int j;
```

- Definitions may optionally assign a value (initialization)

```
int j = 0;
```

- Declarations specify meaning but may not reserve storage (e.g. *extern*)

```
extern int j;
```

- Release builds typically don't initialize variables by default!
- Usage variables:

▶ e.g: `printf("%d + %d = %d\n", a, b, c);`



Example: variable declarations

- `int i;`
- `char c;`
- `float f1, f2;`
- `float f1=7.0, f2 = 5.2;`
- `unsigned int ui = 0;`

Example

```
1. #include <stdio.h>
2.
3. int main()
4. {
5.     int a, b, c;
6.     printf("The first number: ");
7.     scanf("%d", &a);
8.     printf("The second number: ");
9.     scanf("%d", &b);
10.    c = a + b;
11.    printf("%d + %d = %d\n", a, b, c);
12.    return 0;
13.}
```

12	c
7	b
5	a

/ngonnguC/bin/tong

The first number: 5

The second number:7

5 + 7 = 12

/ngonnguC/bin/

Variables and Constants

- Variables:

- Name for a memory object.
- Used to store values and we can change these values.
- Declaration: Tells compiler about variables and their type

`<typename> varname;`

e.g:

```
int i;
```

```
float x, y, z;
```

```
char c;
```

- Assignment: `<varname> = <value>;`

vd:

```
i = 4;
```

```
x = 5.4;
```

```
y = z = 1.2;
```

Variables and Constants (2)

- Constant: the value is invariable during the program.
 - Declaration constant:
`#define <constantname> <value>`
example:
`#define TRUE 1`
`#define FALSE 0`

Constants (1)

- Integer constants

```
31 /* decimal */  
037 /* octal */  
0x1F /* hexadecimal */  
31L /* long */  
31LU /* unsigned long */
```

- Float constants

```
123.4 /* double */  
123.4F /* float */  
123. /* double */  
123.F /* float */  
123.4L /* long double */  
1e-2 /* double */  
123.4e-3 /* double */
```

Constants (2)

- Character constants

```
'K'          /* normal ASCII - 'K' */
'\113'      /* octal ASCII - 'K' */
'\x48'      /* hexadecimal ASCII - 'K' */
'\n'        /* normal ASCII - newline */
'\t'        /* normal ASCII - tab */
'\'\'       /* normal ASCII - backslash */
'\\"'       /* normal ASCII - double quote */
'\0'        /* normal ASCII - null (marks end of string)*/
```

- String literals

"You have fifteen thousand new messages."

"I said, \"Crack, we're under attack!\"."

"hello," "world" **becomes** → "hello, world"

Basic data types (1)

- Sizes and limits (may vary for machine; CUNIX is shown here):

<i>type</i>	<i>size in bits (on CUNIX)</i>	<i>range</i>
char	8	-128...127
short	16	-32,768...32,767
int	32	-2,147,483,648...2,147,483,647
long	32	-2,147,483,648...2,147,483,647
float	32	$10^{-38} \dots 3 \times 10^{38}$
double	64	$2 \times 10^{-308} \dots 10^{308}$

- float has 6 bits of precision (on CUNIX)
- double has 15 bits of precision (on CUNIX)
- range differs from one machine to another
 - int is "native" size

Basic data types (2)

- You can also have unsigned values:

<i>type</i>	<i>size in bits (on CUNIX)</i>	<i>range</i>
unsigned char	8	0...255
unsigned short	16	0...65,535
unsigned int	32	0... 4,294,967,295
unsigned long	32	0... 4,294,967,295

- Look at `/usr/include/limits.h`



Formatting Input with Scanf

- **scanf**
 - Input formatting
 - Capabilities
 - Input all types of data
 - Input specific characters
 - Skip specific characters
- **Format**
 - `scanf(format-control-string, other-arguments);`
 - format-control-string - describes formats of inputs
 - other-arguments - pointers to variables where input will be stored
 - can include field widths to read a specific number of characters from the stream

Formatting Input with scanf (II)

Conversion specifier	Description
<i>Integers</i>	
d	Read an optionally signed decimal integer. The corresponding argument is a pointer to integer.
i	Read an optionally signed decimal, octal, or hexadecimal integer. The corresponding argument is a pointer to integer.
o	Read an octal integer. The corresponding argument is a pointer to unsigned integer.
u	Read an unsigned decimal integer. The corresponding argument is a pointer to unsigned integer.
x or X	Read a hexadecimal integer. The corresponding argument is a pointer to unsigned integer.
h or l	Place before any of the integer conversion specifiers to indicate that a short or long integer is to be input.
<i>Floating-point numbers</i>	
e , E , f , g or G	Read a floating-point value. The corresponding argument is a pointer to a floating-point variable.
l or L	Place before any of the floating-point conversion specifiers to indicate that a double or long double value is to be input.
<i>Characters and strings</i>	
c	Read a character. The corresponding argument is a pointer to char , no null (' \0 ') is added.
s	Read a string. The corresponding argument is a pointer to an array of type char that is large enough to hold the string and a terminating null (' \0 ') character—which is automatically added.
<i>Scan set</i>	
[scan characters]	Scan a string for a set of characters that are stored in an array.
<i>Miscellaneous</i>	
p	Read an address of the same form produced when an address is output with %p in a printf statement.
n	Store the number of characters input so far in this scanf . The corresponding argument is a pointer to integer
%	Skip a percent sign (%) in the input.

• Example of scanf()

```
int d,m,y,x;
char ch1,ch2;
float f;
scanf("%d", &x);

scanf("%2d%2d%4d", &d,&m,&y);

scanf("%d/%d/%d", &d,&m,&y);

scanf("%c%c", &ch1,&ch2);

scanf("%f", &f);
```

Result

```
4
// x=4
22062007
// d=22, m=6, y=2007
22/06/2007
// d=22, m=6, y=2007
Ab
// ch1='A', ch2='b'
2.3
// f=2.300000
```

Formatting Input with scanf (III)

- Scan sets
 - Set of characters enclosed in square brackets []
 - Preceded by % sign
 - Scans input stream, looking only for characters in scan set
 - Whenever a match occurs, stores character in specified array
 - Stops scanning once a mismatch is found
 - Inverted scan sets
 - Use a caret ^: [^aeiou]
 - Causes characters not in the scan set to be stored

Formatting Input with scanf (IV)

- Skipping characters
 - Include character to skip in format control
 - Or, use * (assignment suppression character)
 - Skips any type of character without storing it

Example 2

•Reading characters and strings

```
1 #include <stdio.h>
2
3 int main()
4 {
5     char x, y[ 9 ];
6
7     printf( "Enter a string: " );
8     scanf( "%c%s", &x, y );
9
10    printf( "The input was:\n" );
11    printf( "the character \"%c\" ", x );
12    printf( "and the string \"%s\"\n", y );
13
14    return 0;
15 }
```

```
Enter a string: Sunday
The input was:
the character "S" and the string "unday"
```

Example 3

- Using an inverted scan set

```
2 #include <stdio.h>
3
4 int main()
5 {
6     char z[ 9 ] = { '\0' };
7
8     printf( "Enter a string: " );
9     scanf( "%[^aeiou]", z );
10    printf( "The input was \"%s\"\n", z );
11
12    return 0;
13}
```

```
Enter a string: String
The input was "Str"
```

Example 4

- Reading and discarding characters from the input stream

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int month1, day1, year1, month2, day2, year2;
6
7     printf( "Enter a date in the form mm-dd-yyyy: " );
8     scanf( "%d%c%d%c%d", &month1, &day1, &year1 );
9     printf( "month = %d  day = %d  year = %d\n\n",
10          month1, day1, year1 );
10    printf( "Enter a date in the form mm/dd/yyyy: " );
14    scanf( "%d%c%d%c%d", &month2, &day2, &year2 );
15    printf( "month = %d  day = %d  year = %d\n",
16          month2, day2, year2 );
17
18    return 0;
19 }
```

```
Enter a date in the form mm-dd-yyyy: 11-18-2000
month = 11  day = 18  year = 2000
```

```
Enter a date in the form mm/dd/yyyy: 11/18/2000
month = 11  day = 18  year = 2000
```




Exercises 4.1

- Write a program that reads a integer and a double from user, use a floating-point and an integer variable to store and then show to screen.

Solution

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

```
void main (void) {
```

```
    /* We will use a floating-point and an integer variable. */
```

```
    double x;  
    int n;
```

```
    /* Read in an integer. */
```

```
    printf("Please enter an integer: ");  
    scanf("%d", &n);  
    printf("The integer was %d\n\n", n);
```

Solution

```
/* Read in a double. */
```

```
printf("Please enter a double: ");  
scanf("%lf", &x);  
printf("The double was %g\n\n", x);
```

```
/* Read in an integer and a double. */
```

```
printf("Please enter an integer and a floating-point number: ");  
scanf("%d%lf", &n, &x);  
printf("The numbers were %d %g\n", n, x);
```

```
}
```



Exercises 4.2

- Write and run this program to see the limit of basic data types: int, long.
- Widen this program for other basic data types.
- Use `limits.h` library to build your programs.

Solution

```
#include <stdio.h>
#include <limits.h> /* defines INT_MIN, INT_MAX, LONG_MIN, LONG_MAX
*/

main()
{
    int j;
    long int k;
    float x;
    double z;

    printf("Enter an integer (between %i and %i): ", INT_MIN, INT_MAX);
    scanf("%i", &j);
    printf("You entered %d\n\n", j);
```

Solution

```
printf("Enter a long integer (between %ld and %ld): ", LONG_MIN,  
LONG_MAX);  
scanf("%ld", &k);  
printf("You entered %li\n\n", k);  
  
printf("Enter a floating point number: ");  
scanf("%f", &x);  
printf("You entered %20.10E\n\n", x);  
  
printf("Enter a double precision floating point number: ");  
scanf("%lf", &z);  
printf("You entered %20.10E\n\n", z);  
  
puts("\n\nTry again: enter invalid data and see what happens!");  
}
```



Exercises 4.3

- Write a program that reads a string from the keyboard by using a scan set.

Solution

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

```
/* function main begins program execution */
```

```
int main( void )
```

```
{
```

```
    char z[ 9 ];
```

```
    printf( "Enter string: " );
```

```
    scanf( "%[aeiou]", z ); /* search for set of characters */
```

```
    printf( "The input was \"%s\"\n", z );
```

```
    return 0;
```

```
}
```




Exercises 4.4

- Write a program that inputs data with a field width.
- Widen to all basic data types.

Solution

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

```
int main( void )
```

```
{
```

```
    int x;
```

```
    int y;
```

```
    printf( "Enter a six digit integer: " );
```

```
    scanf( "%2d%d", &x, &y );
```

```
    printf( "The integers input were %d and %d\n", x, y );
```

```
    return 0; /* indicates successful termination */
```

```
}
```



Exercise 4.5

- Write a program ask user to input the radius of a circle. Use constant for PI.
 - a) Display its area and circumference.
 - b) Now consider the input data is the radius of a sphere. Display its area and volume.

Solution

```
#include <stdio.h>
#define PI 3.142

main()
{
    double r, c, ac, as, v;
    r = 5.678;
    printf("Radius = %f\n", r);

    c = 2.0 * PI * r;
    printf("Circle's circumference = %f\n", c);

    ac = PI * r * r;
    printf("Circle's area = %f\n", ac);

    as = 4.0 * PI * r * r;
    printf("Sphere's area = %f\n", as);

    v = 4.0/3.0 * PI * r * r * r;
    printf("Sphere's volume = %f\n", v);
}
```



Exercise 4.6

- Write a program that calculates and displays an employee's total wages for week. The regular hours for the work week are 40 and any hours worked over 40 are considered overtime. The employee earns 25000 VND per hour for regular hours, and 40000 VND per hour for overtime hours. This week employee has worked 50 hours.

Solution

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

```
int main()
```

```
{
```

```
    double regWages,      // calculated regular wages
```

```
        basePay = 25000, // base pay rate
```

```
        regHours = 40.0, // hours worked less overtime
```

```
        otWages,        // overtime wages
```

```
        otPay = 40000,  // overtime pay rate
```

```
        otHours = 10,   // overtime hours worked
```

```
        totalWages;    // total wages
```

```
    regWages = basePay * regHours;
```

```
    otWages = otPay * otHours;
```

```
    totalWages = regWages + otWages;
```

```
    printf("\n Wages for this week are %0.1f $", totalWages);
```

```
    return 0;
```

```
}
```

Exercise 4.7

- Write a program that ask users for information concerning a book you buy at the shop such as: ISBN, Title, Price, Quantity. The VAT is 4%. Program should display these information as the following interface:

BK Bookseller				
Qty	ISBN	Title	Price	Total
<hr/>				
VAT				
You pay:				

Solution

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

```
int main()
```

```
{
```

```
    double regWages,      // calculated regular wages
```

```
        basePay = 25000, // base pay rate
```

```
        regHours = 40.0, // hours worked less overtime
```

```
        otWages,        // overtime wages
```

```
        otPay = 40000,  // overtime pay rate
```

```
        otHours = 10,   // overtime hours worked
```

```
        totalWages;    // total wages
```

```
    regWages = basePay * regHours;
```

```
    otWages = otPay * otHours;
```

```
    totalWages = regWages + otWages;
```

```
    printf("\n Wages for this week are %0.1f $", totalWages);
```

```
    return 0;
```

```
}
```