Lab 1: Introduction To Embedded Systems

Onur Kilincceker (MSKU, Computer Engineering)

Credits

- https://www.unf.edu/~wkloster/
- https://beginnersbook.com/2014/01/c-tutorial-for-beginners-with-examples/

Contents

1. Course syllabus and project teams
2. Lab 1: C programming
3. Lab 2: Basics of Arduino Programming and Simulation (Toolchain)
4. Lab 3: Example for Arduino Programming and Simulation

Lab 1: C programming

Onur Kilincceker (MSKU, Computer Engineering)

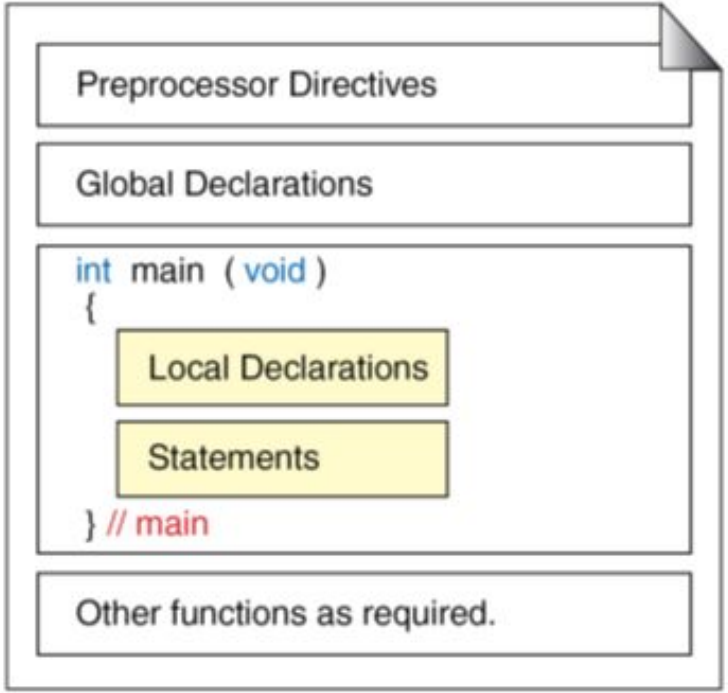


FIGURE 2-2 Structure of a C Program

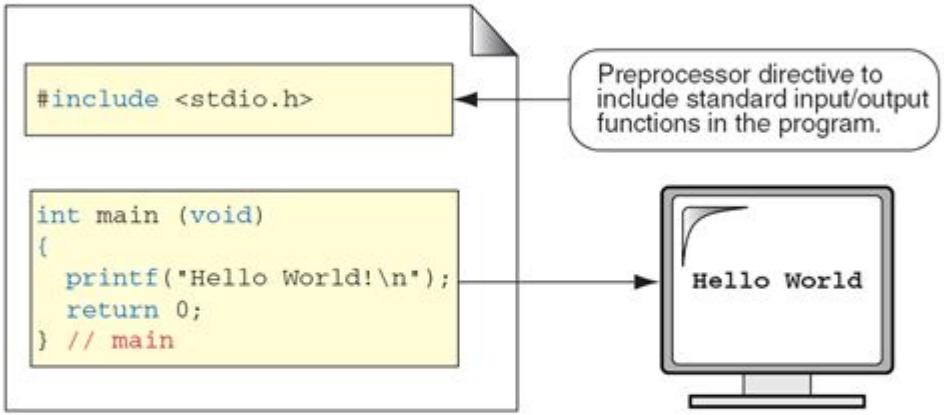


FIGURE 2-3 The Greeting Program

PROGRAM 2-1 The Greeting Program

```
1  /* The greeting program. This program demonstrates
2     some of the components of a simple C program.
3        Written by:  your name here
4        Date:        date program written
5  */
6  #include <stdio.h>
7
8  int main (void)
9  {
10 // Local Declarations
11
12 // Statements
13
14    printf("Hello World!\n");
15
16    return 0;
17 } // main
```

2-3 Identifiers

One feature present in all computer languages is the identifier. Identifiers allow us to name data and other objects in the program. Each identified object in the computer is stored at a unique address.

1. First character must be alphabetic character or underscore.
2. Must consist only of alphabetic characters, digits, or underscores.
3. First 63 characters of an identifier are significant.
4. Cannot duplicate a keyword.

Table 2-1 Rules for Identifiers

Note

An identifier must start with a letter or underscore: it may not have a space or a hyphen.

An identifier must start with a letter or underscore: it may not have a space or a hyphen.

Valid Names		Invalid Name	
a	// Valid but poor style	$sum	// $ is illegal
student_name		2names	// First char digit
_aSystemName		sum-salary	// Contains hyphen
_Bool	// Boolean System id	stdnt Nmbr	// Contains spaces
INT_MIN	// System Defined Value	int	// Keyword

2-4 Types

A type defines a set of values and a set of operations that can be applied on those values.

Topics discussed in this section:

Void Type
Integral Type
Floating-Point Types

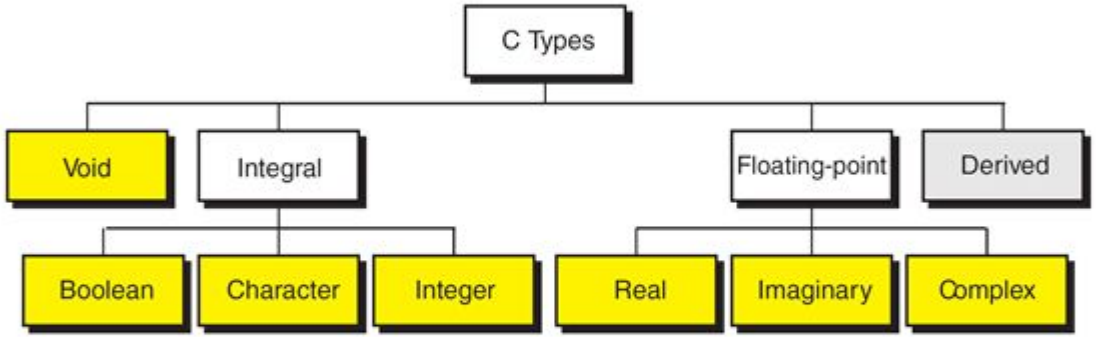


FIGURE 2-7 Data Types

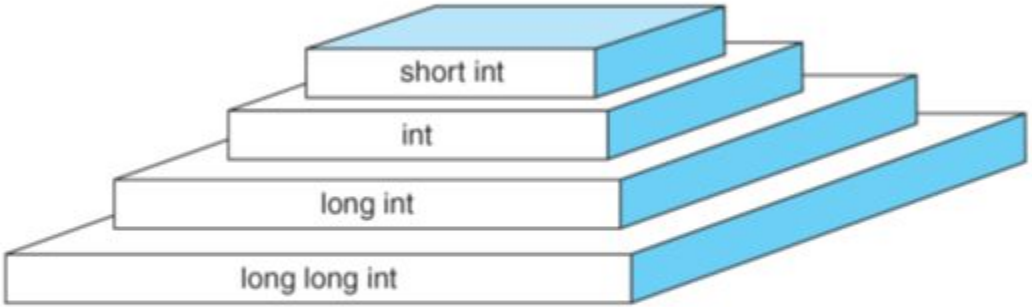


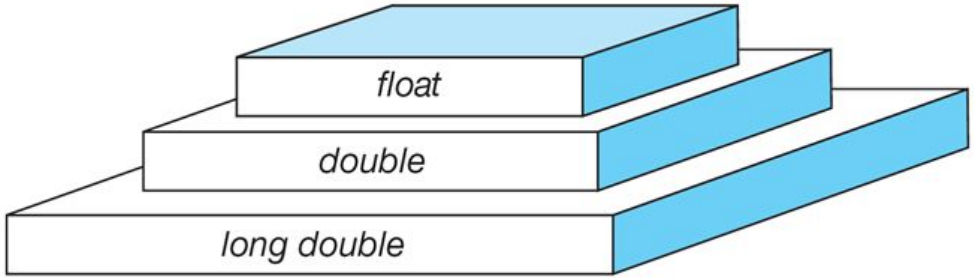
FIGURE 2-9 Integer Types

Note

sizeof (short) ≤ sizeof (int) ≤ sizeof (long) ≤ sizeof (long long)

Type	Byte Size	Minimum Value	Maximum Value
short int	2	–32,768	32,767
int	4	–2,147,483,648	2,147,483,647
long int	4	–2,147,483,648	2,147,483,647
long long int	8	–9,223,372,036,854,775,807	9,223,372,036,854,775,806

Table 2-3 Typical Integer Sizes and Values for Signed Integers



Note

sizeof (float) ≤ sizeof (double) ≤ sizeof (long double)

Category	Type	C Implementation
Void	Void	*void*
Integral	Boolean	*bool*
	Character	*char, wchar_t*
	Integer	*short int, int, long int, long long int*
Floating-Point	Real	*float, double, long double*
	Imaginary	*float imaginary, double imaginary, long double imaginary*
	Complex	*float complex, double complex, long double complex*

2-5 Variables

Variables are named memory locations that have a type, such as integer or character, which is inherited from their type. The type determines the values that a variable may contain and the operations that may be used with its values.

Topics discussed in this section:

Variable Declaration

Variable Initialization

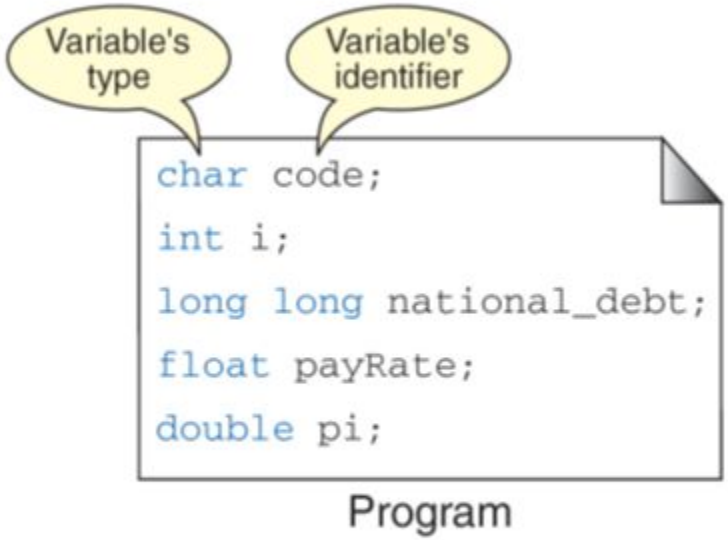
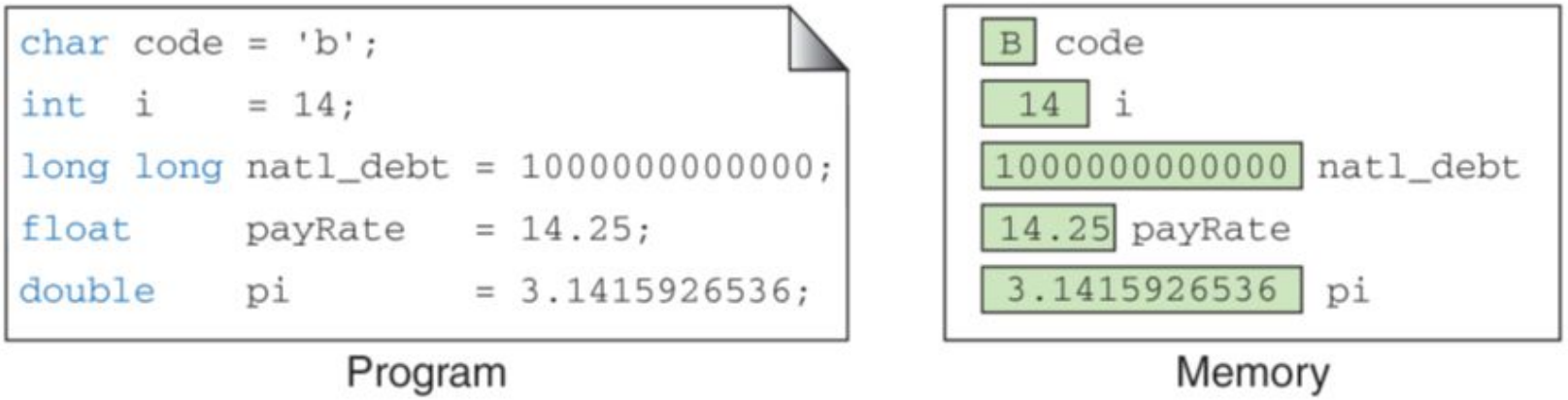


FIGURE 2-11 Variables

```
bool   fact;
short  maxItems;              // Word separator: Capital
long   long national_debt;    // Word separator: underscore
float  payRate;               // Word separator: Capital
double tax;
float  complex voltage;
char   code, kind;            // Poor style—see text
int    a, b;                  // Poor style—see text
```

Table 2-5 Examples of Variable Declarations and Definitions



Note

When a variable is defined, it is not initialized. We must initialize any variable requiring prescribed data when the function starts.

PROGRAM 2-2 **Print Sum of Three Numbers**

```
/* This program calculates and prints the sum of
   three numbers input by the user at the keyboard.
      Written by:
      Date:
*/
#include <stdio.h>

int main (void)
{
// Local Declarations
   int a;
   int b;
   int c;
   int sum;

// Statements
   printf("\nWelcome. This program adds\n");
   printf("three numbers. Enter three numbers\n");
   printf("in the form: nnn nnn nnn <return>\n");
   scanf("%d %d %d", &a, &b, &c);

   // Numbers are now in a, b, and c. Add them.
   sum = a + b + c;

   printf("The total is: %d\n\n", sum);

   printf("Thank you. Have a good day.\n");
   return 0;
}  //  main
```

```
Results:
Welcome. This program adds
three numbers. Enter three numbers
in the form: nnn nnn nnn <return>
11 22 33

The total is: 66

Thank you. Have a good day.
```

2-6 Constants

Constants are data values that cannot be changed during the execution of a program. Like variables, constants have a type. In this section, we discuss Boolean, character, integer, real, complex, and string constants.

Topics discussed in this section:

Constant Representation
Coding Constants

Note

A character constant is enclosed in single quotes.

ASCII Character	Symbolic Name
null character	`'\0'`
alert (bell)	`'\a'`
backspace	`'\b'`
horizontal tab	`'\t'`
newline	`'\n'`
vertical tab	`'\v'`
form feed	`'\f'`
carriage return	`'\r'`
single quote	`'\''`
double quote	`'\"'`
backslash	`'\\'`

Symbolic Names for Control Characters

Representation	Value	Type
+123	123	int
–378	–378	int
–32271L	–32,271	long int
76542LU	76,542	unsigned long int
12789845LL	12,789,845	long long int

Examples of Integer Constants

Representation	Value	Type
0.	0.0	double
.0	0.0	double
2.0	2.0	double
3.1416	3.1416	double
–2.0f	–2.0	float
3.1415926536L	3.1415926536	long double

Examples of Real Constants

```
""                                                    // A null string
"h"
"Hello World\n"
"HOW ARE YOU"
"Good Morning!"
L"This string contains wide characters."
```

Note

Use single quotes for character constants.
Use double quotes for string constants.

```c
/* This program demonstrates three ways to use con-
stants.
      Written by:
      Date:
*/
#include <stdio.h>
#define PI 3.1415926536

int main (void)
{
// Local Declarations
   const double cPi = PI;

// Statements
   printf("Defined constant PI: %f\n", PI);
   printf("Memory constant cPi: %f\n", PI);
   printf("Literal constant:    %f\n", 3.1415926536);
   return 0;
}  // main
```

```
Results:
Defined constant PI:  3.141593
Memory constant cPi:  3.141593
Literal constant:     3.141593
```

Memory Constants

C - If statement

The statements inside the body of “if” only execute if the given condition returns true. If the condition returns false then the statements inside “if” are skipped.

```
if (condition)
{
     //Block of C statements here
     //These statements will only execute if the condition is true
}
```

C – If statement

```c
#include <stdio.h>
int main()
{
    int x = 20;
    int y = 22;
    if (x<y)
    {
        printf("Variable x is less than y");
    }
    return 0;
}
```

C If else statement

If condition returns true then the statements inside the body of “if” are executed and the statements inside body of “else” are skipped.

If condition returns false then the statements inside the body of “if” are skipped and the statements in “else” are executed.

C If else statement

```c
#include <stdio.h>
int main()
{
   int age;
   printf("Enter your age:");
   scanf("%d",&age);
   if(age >=18)
   {
        /* This statement will only execute if the
         * above condition (age>=18) returns true
         */
        printf("You are eligible for voting");
   }
   else
   {
        /* This statement will only execute if the
         * condition specified in the "if" returns false.
         */
        printf("You are not eligible for voting");
   }
   return 0;
}
```

C – Switch Case Statement

```c
#include <stdio.h>
int main()
{
     int num=2;
     switch(num+2)
     {
         case 1:
           printf("Case1: Value is: %d", num);
         case 2:
           printf("Case1: Value is: %d", num);
         case 3:
           printf("Case1: Value is: %d", num);
         default:
           printf("Default: Value is: %d", num);
    }
    return 0;
}
```

C For loop

```
for (initialization; condition test; increment or decrement)
{
        //Statements to be executed repeatedly
}
```

```c
#include <stdio.h>
int main()
{
   int i;
   for (i=1; i<=3; i++)
   {
       printf("%d\n", i);
   }
   return 0;
}
```

C – WHILE LOOP

```
while (condition test)
{
      //Statements to be executed repeatedly
      // Increment (++) or Decrement (--) Operation
}
```

```
#include <stdio.h>
int main()
{
   int count=1;
   while (count <= 4)
   {
        printf("%d ", count);
        count++;
   }
   return 0;
}
```

C – DO..WHILE LOOP

```
do
{
    //Statements

}while(condition test);
```

```
#include <stdio.h>
int main()
{
        int j=0;
        do
        {
                printf("Value of variable j is: %d\n", j);
                j++;
        }while (j<=3);
        return 0;
}
```

More details on C Programming

1. https://beginnersbook.com/2014/01/c-tutorial-for-beginners-with-examples/
2. Kernighan, B. W., & Ritchie, D. M. (1988). C Programming Language, 2nd Edition. S.l.: Pearson.