

C Programming

Learning Package 2

Data Types and Variables

Dr Scott Turner

School of Science and Technology

University of Northampton



**Introduction**

In Learning Package 1, we considered what parts make up a C program, and how a program can be written.

Within this Learning Package we are going to look at some of the basic elements of C programming, what are data types, variables and statements. We also look how numbers can be used and displayed.

**Learning outcomes**

At the end of this Learning Package, the reader should be able to:

* Describe the terms statements and data types.
* Describe the difference between a floating point, integer and character data types.
* Demonstrate how C programs can be used to perform calculations and display the results.
* Describe and demonstrate the use of simple and compound statements.
* Demonstrate the ability to write programs that allow users to enter information from a keyboard to be used by a program.

**Study guide**

Session 1:

* 1. Language SAQ 2.1 SAQ 2.2
	2. Data types SAQ 2.3 Task 2.1 SAQ 2.4

Session 2:

* 1. Using numbers SAQ 2.5

SAQ 2.6

Task 2.2

Session 3:

* 1. Type conversions, expressions and statements
	2. SAQ 2.7
	3. Scanf and printf Task 2.3

Session 4:

Exercises

**2.1 Language**

Read from the start of Chapter 3 on page 43 to 48 of the module textbook, stopping just before Data Types. The textbook talks about objects these should be confused with objects in object-orientated programming. Objects in this module are items of data.

SAQ 2.1

Fill in the missing words.

* + 1. The six categories of tokens are **k s**, **i s**, **c ts**, **s g l ls**, **o tors** and **s ors**.
		2. One of the fundamental building blocks of C programming is characters. The characters out of which all C programs are built can be separated into **l r** and **u r c\_se letters**; **d\_g\_ts**;

**pu ion**; **s al c s**; and **n\_n-p ting c s**.

* + 1. An identifier is a **s e** of **c s** used to name **d a** to be manipulated by the program. There are **r s** for **i r n s**.
		2. A **c t** will remain the **s\_\_e** throughout the program, whereas

**v s** can be **c d**.

* + 1. **K s** are reserved **w s** that can **n\_t** be used for **v e n s** or **c ts**.

SAQ 2.2

1. For each state whether these are true or false
	1. E is a lower case letter
	2. 1f is a valid variable name
	3. f1 is a valid variable name
	4. if is a valid variable name
	5. l 1 is a valid variable name
	6. l\_1 is valid variable name
2. What does the term **meaningful variable name** mean?
	1. **Data Types**

Read the module textbook up to the section labelled 3.2 Data typesstarting on page 48 and 3.3 Making Declarationsending on page 58 including Charlie’s musings. When dealing with integers, a lot of the time 32 bits will be used to store an integer, giving a range of -2147483647 to 2147483647.

SAQ 2.3

For each of the following state the difference between the terms:

* + 1. *int* and *unsigned int*
		2. *int* and *float*
		3. *float* and *double*

Task 2.1

Type in, compile, and execute the following programs

* + - 1. Program 3.1 on page 47 of the module textbook
			2. Program 3.2 on page 50 of the module textbook

SAQ 2.4

Fill in the missing words.

The **d a t e** *char* is used represent a **s e c r**. **O\_e** byte is normally used to store **e h c r**.

* 1. **Using the numbers**

Read the module textbook from the section labelled 3.4 Doing a little calculationon page 58 to the end of the section labelled 3.6 Some new operatorson page 70.

Calculations can be done within a printf () function. The following code

D=1\*2;

printf(“Number1=%d Number2=%d D= %d”,1,2,D);

Could also be written as

printf(“Number1=%d Number2=%d D= %d”,1,2,1\*2);

This is useful when a calculation result needs to be displayed but not stored.

SAQ 2.5

Fill in the missing words.

The **f n** scanf() allows the user to enter data from the **k d**. An important point is the parts of the **a t l t** do not represent the

**v s**, but **p t** to the **a s** in **m y** where the **d a** is to be **s d**. The **v e n e** has to be prefixed by an &.

SAQ 2.6

1. State the order of precedence of the arithmetic operators.
2. For each of the following give another way that the C statements can be written:
	1. x++;
	2. x=x+2;
	3. x=x\*3;
	4. x- -
	5. x-=4
3. In a scanf() function what do the following do:
	1. %d
	2. %f
	3. %s
	4. &

Task 2.2

1. Type in, compile, and execute the following programs
2. Program 3.3 on page 62 of the module textbook
3. Program 3.4, which start on page 67. Do not worry about what the line **for (i=0;i<4;i++)**does. For the moment, it makes the program carry out parts of the program four times. We will look at this in more detail later in the module (Learning Package 4).
4. Program 3.5 on page 68 of the module textbook
	1. **Type conversions, expressions and statements**

Read the module textbook from the section labelled 3.7 Type conversions on page 70 to the end of the section labelled 3.9 statementson page 76.

SAQ 2.7

Fill in the missing words.

* + 1. In C, mixing data **t s** in the same **s t** is **a d**. If different **d a t s** are mixed, make sure the **v e** that will store the final value is of **h r** (in a list of precedence) that any of the other variables.
		2. Casting is another way of **c ting** one **d a t e** to **a r** by including the **s t** in **b ts** the new **d a t e** next to the **v e** to **c ted**.
		3. Statements can be simple, structured, or compound. A **s e s t** is a **c e i n** and ends in a

**s n**. A **c d s t** contains **t\_o or m e** simple

**s ts** enclosed within a set of **b s** ({ }). A **s d s t** is more **c x** than a **s e s t**.

* 1. **Scanf and printf**

Read the section labelled 3.10 Formatted input and output: scanf () and printf () on pages 76-81 of the module textbook.

Task 2.3

Type in, compile and execute the program on page 79 of the module textbook (you should include the line #include <stdio.h>). What appears of the screen when you executed (ran) the program? Does it match what the book says?

**Exercises**

1. Do exercises 2, 4, 5, 6, 7, 8, 9, 10, 11, 12 from the module textbook. Remember to add system (“PAUSE”); to the end of your programs.
2. Do exercise 3 but use the modified program below:

main()

{

printf("\n\tData Type\tSize in Bytes\n");

printf("\n\t==========\t=============\n"); printf("\n\tShort int\t%Ld",sizeof(short)); printf("\n\tint\t\t%Ld",sizeof(int)); printf("\n\tlong int\t%Ld",sizeof(long)); printf("\n\tchar\t\t%Ld",sizeof(char)); printf("\n\tfloat\t\t%Ld",sizeof(float)); printf("\n\tdouble\t\t%Ld\n",sizeof(double)); system("PAUSE");

}

**Answers to SAQs**

SAQ2.1

1. The six categories of tokens are **keywords**, **identifiers**, **constants**, **string literals**, **operators** and **separators**.
2. One of the fundamental building blocks of C programming is characters. The characters out of which all C programs are built can separated into **lower** and **upper case letters**; **digits**; **punctuation**; **special characters**; and **non-printing characters**.
3. An identifier is a **sequence** of **characters** used to name **data** to be manipulated by the program. There are **rules** for **identifier names**.
4. A **constant** will remain the **same** throughout the program, whereas

**variables** can be **changed**.

1. **Keywords** are reserved **words** that cannot be used for **variable names** or **constants**.

SAQ 2.2

1. For each state, whether these are true or false
	1. FALSE
	2. FALSE
	3. TRUE
	4. FALSE
	5. FALSE
	6. TRUE
2. A meaningful variable name is a name for a variable that has some meaning to what is being store in that variable. For example, a variable called **age** could be used to store the age of a person. These help to make a program easier to understand.

SAQ 2.3

1. Assuming a 32-bit number. *int* has a sign bit to signify whether it is a negative or positive, so out of 32 bits 31 are used to represent the actual size of the number (range -2147483647 to 2147483647). When *unsigned int* is used, all 32 bits can be used to represent the actual size of the number (0 to 4294967295).
2. *int* stores whole numbers such as 1, 123, and 567678. *float* stores real numbers which have fractional part (1.1, 456.788. 1.0). A whole number can be stored in a variable defined as float but the fractional part is zero for example 1.0), and integer cannot store a floating-point number.
3. *float* and *double* both store floating-point numbers but in the *double,*

more bits are used to represent the number.

SAQ 2.4

The **data type** *char* is used represent a **single character**. **One** byte is normally used to store **each character**.

SAQ 2.5

The **function** scanf() allows the user to enter data from the **keyboard**. An important point is the parts of the **argument list** do not represent the **variables**, but **point** to the **address** in **memory** where the **data** is to be **stored**. The **variable name** has to be prefixed by an **&**.

SAQ 2.6

1. ()

++ -- -(when it is used, represent negative number e.g. -3)

\* / %

+ -

=

1. For each of the following give another way that the C statements can be written:
	1. x=x+1; b. x+=2; c. x\*=3;
2. x=x-1;
3. x=x-4;
4. In scanf() function what do the following do:
	1. %d – the value entered will be stored in an integer variable.
	2. %f – the value entered will be stored in a floating-point variable.
	3. %s – the value entered will be stored as a string.
	4. & – the value entered will be stored in memory location of the variable that follows the &.

SAQ 2.7

Filling in the missing words

1. In C, mixing data **types** in the same **statement** is **allowed**. If different **data types** are mixed, make sure the **variable** that will store the final value is of **higher** (in a list of precedence) that any of the other variables.
2. Casting is another way of **converting** one **data type** to **another** by including the **statement** in **brackets** the new **data type** next to the **variable** to **converted**.
3. Statements can be simple, structured, or compound. A **simple statement** is a **complete instruction** and ends in a **semicolon**. A **compound statement** contains **two or more** simple **statements** enclosed within a set of **braces** ({ }). A **structured statement** is more **complex** than a **simple statement**.

**Selected exercise solutions**

2 (a) valid, (b) valid, (c) valid, (d) invalid starts with an arithmetic operator,

(e) invalid – int is a reserved word, (f) valid, (g)valid, (h) valid, (I )invalid – starts with %, (j) invalid – as ‘ is seen as the start of a character assignment.

4(a) invalid – begins with a number, (b) valid, (c) invalid the calculation should be on the right-hand side of the equation and the variable total should be on the left-handside on its own, (d) valid (see program below) (e) valid.

main()

{

int elephant,giraffe,monkey; elephant=giraffe=monkey=0;

printf("%d %d %d",elephant,giraffe,monkey);

}

7 Hint:

main()

{

}

10.

main()

{

char test1='%'; printf("%c %%",test1);

float chain=34.0,rear1=14.0,rear2=16.0,rear3=18.0; float ratio1,ratio2,ratio3;

ratio1=chain/rear1; ratio2=chain/rear2; ratio3=chain/rear3;

printf("\n\nratio=%f\tdistance=%f inches",ratio1,ratio1\*27); printf("\n\nratio=%f\tdistance=%f inches",ratio2,ratio2\*27); printf("\n\nratio=%f\tdistance=%f inches\n",ratio3,ratio3\*27);

}

12

main()

{

int sec\_min=60,sec\_hour,hours,minutes,seconds,inp\_sec; int temp1,temp2;

sec\_hour=sec\_min\*60; printf("\nEnter seconds "); scanf("%d",&inp\_sec); hours=inp\_sec/sec\_hour; temp1=hours\*sec\_hour; minutes=(inp\_sec-temp1)/sec\_min; temp2=temp1+(minutes\*sec\_min); seconds=inp\_sec%temp2;

printf("\n\n %d:%d:%d",hours,minutes,seconds); system("PAUSE");

}