LECTURE 02

Data Types

Outcomes

1. To understand further of C programming structure involving identifiers, variables, constants, and data types.
2. To construct simple C program by naming, declaring and initializing different kind of data types.

Contents

1. Identifiers
2. Variables
3. Standard Data Types
4. Constants

1. Identifiers

Identifiers are the names that you give to entities such as variables, functions, structures etc. Identifier names must be unique. They are created to give unique name to a C entity to identify it during the execution of a program. For example:

```
int money;
double accountBalance;
```

Here, `int` and `double` are C reserved words, meanwhile `money` and `accountBalance` are identifiers. The most important thing about identifier is the names must be different from keywords (C reserved words). You cannot use `int` as an identifier because `int` is a keyword. Rules for writing identifiers are as follows:

1. Valid identifier can have letters (both uppercase and lowercase letters), digits and underscore only.
   Example: `student_Name1`
2. The first letter of an identifier should be either a letter or an underscore. However, it is discouraged to start an identifier name with an underscore. It is because identifier that starts with an underscore can conflict with system names. In such cases, compiler will complain about it. Some system names that start with underscore are `_fileno`, `_iob`, `_wfopen` etc.
   Examples:
   ```
   _student //valid but not encourage
   student  //valid
   ```
3. There is no rule on the length of an identifier. However, the first 31 characters of identifiers are discriminated by the compiler. So, the first 31 letters of two identifiers in a program should be different.
   Examples:
   ```
   studentComputerProgramming1 //27 characters
   studentComputerProgramming2 //27 characters
   ```
   These two identifiers can be discriminated (read as different identifier) by the compiler since only 26 characters are same which is `studentComputerProgramming`. If these two identifiers are same characters for 31 characters or more, the compiler cannot discriminate and read as same identifier.
   For example:
   ```
   students_Computer_Programming_10 //32 characters
    students_Computer_Programming_12 //32 characters
   ```
4. Cannot contains hyphen and space.
   Examples:
   ```
   sum-salary //not valid contain hyphen
   stdnt Nmbr //not valid contain space
   ```
5. Cannot begin with digit.
   Example: `2names //not valid begin with digit`
6. Can be in any language.
   Examples: `gajiPekerja  //valid`

```
workerSalary //valid
```

You can choose any name for an identifier. However, if the programmer choose meaningful name for an identifier, it will be easy to understand. This is the good programming practice you should try to do when writing programming codes. For instance, lets say if you want to declare sum for adding two numbers, you should write `double sum` or `double jumlah` or `double jum` or `double jum_keseluruhan`. Please do not write `double s` or `double j`, etc. By doing this, other people could not understand your codes and it will be harder for other people to help you when you have any errors regarding your codes and etc.

2. Variables

Variables are named of 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. Examples:

```
char code;          //char is variable type, code is variable identifier
int i;              //int is variable type, i is variable identifier
long long hutang_negara;        //long long is variable type, hutang_negara is
                                variable identifier
```

Variable must be declared and initialized. So what is declare and what is initialize? In C, a declaration is used to name an object, such as variable. Meanwhile, the initializer establishes the value the variable will contain. Examples of variable declarations are:

```
short maxItems;                 //Word separator: Capital letter
long long national_debt;        //Word separator: underscore
double tax;
int a, b;
float voltage;
```

One of important thing to be remember is when a variable is declared, it is not initialized yet. As seen in the above example, all of variables does not have any values. We must initialize any variable. We can initialize a variable at the same time that we declare it. For instance:

```
short maxItems = 10;
long long national_debt = 260000000;
double tax = 10.0;
int a = 1, b = 0;
float voltage = 1000.0;
```

Another way to put value on variable is by using `scanf` statement. As we have learned in the previous chapter, `scanf` allows user to insert input from computer keyboard. An example is given below.

```
/*
This program is to calculate and print the sum of three numbers input by the
user
Written by: AFAN, FKP, UMP
Date: September 2016
```

```
*/
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a, b, c, sum;
    printf("Enter three numbers:\n");
    scanf("%d %d %d", &a,&b,&c);
    sum = a + b + c;
    printf("\nThe total is: %d", sum);
    return 0;
}
```

```
Enter three numbers:
1 2 3
The total is: 6
```

From the above example, it is clearly show that the values of a, b and c are initialized and inserted by user as 1, 2 and 3.

3. Standard Data Types

Standard data types represent (1) character, (2) integer, (3) real number, and (4) logical data (TRUE or FALSE). A character is any value that can be presented in the computer's alphabet. Most computer use the American Standard Code (ASCII) alphabet. There are 127 characters in ASCII and extended character is up to 255 characters. You do not need to memorize this alphabet since you can just google it if you want to know them. An integer type is a number without a fraction part (e.g. 1, 2, 10, -10, and -155). The real type holds values that consist of a fractional part such as 40.08. Finally, the Boolean values represent a nonzero or zero number (TRUE or FALSE).

As programmers, we should be aware about choice of data type when we want to represent any data. For example, if we want to represent the number of human populations in Malaysia we probably should use `unsigned int` since there should not be any negative number and no floating number of human populations. Moreover, the maximum number for `unsigned int` is 4,294,967,295. This number should be large enough to represent human populations in Malaysia. On the other hand, if we use `unsigned long long`, the maximum number is too large for our application. Unused memory (too much usage of computer memory) will slower the running application, therefore it is very important to get the right choice of data type. The following table presents the sizes of the standard data types as well as their range of values.

Standard Data Type	C-implementation	Memory Size	Minimum Value	Maximum Value
Character	`char`	1 byte	-128	127
	`unsigned char`	1 byte	0	255
Integer (unsigned)	`unsigned short`	2 bytes	0	65,535
	`unsigned int`	4 bytes	0	4,294,967,295
	`unsigned long long`	8 bytes	0	$2^{64} - 1$
Integer (signed)	`short`	2 bytes	-32,768	32,767
	`int`	4 bytes	-2,147,483,648	2,147,483,647
	`long long`	8 bytes	-2^{63}	$2^{63} - 1$
Real number	`float`	4 bytes	1.17549×10^{-38}	3.40282×10^{38}
	`double`	8 bytes	2.22507×10^{-308}	1.79769×10^{308}
	`long double`	12 bytes	-	-

To understand the size of memory use, lets start at `char` which is using 1 byte memory size. In computer, 1 byte is equivalent to 8 bits. The representation of 8 bits are as below:

2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0 //This is 8 bits (1 byte) representation – power of 2

128 64 32 16 8 4 2 1 //In decimal (e.g. $2^7 = 128$)

Lets say we use `char` 'A' for this example. This character is represented by 65 in decimal number. In order to get 65, we use 64 + 1 in the above chart. Finally, char 'A' is 0100 0001 in binary number. The maximum number for 1 byte is 128 + 64 + 32+ 16 + 8 + 4 + 2 + 1 = 255. This is how the memory size works on your computer. If you want, you can calculate to explain all of memory size for the respective data type based on the above table. Examples on how to declare data types are given below.

```
char huruf = 'Z';
/*reserve one byte memory space identified as huruf and stores the ASCII value of character Z*/

char name[10] = "ahmad naim";
/*reserve one byte memory space for each char (10 char is used including space) identified as name and stores the ASCII value of character ahmad naim*/

double suhu = 40.8324;
/*reserve 8 bytes memory space identified as suhu and stores the value of 40.8324*/

short noRajah = 200;
/*reserve 2 bytes memory space identified as noRajah and stores the value of 200*/
```

In C program, any nonzero value is treated as TRUE and zero value as FALSE for logical data. Lets say we have the following example:

```
/*
This program is to explain the logical data type
Written by: AFAN, FKP, UMP
Date: September 2016
*/
#include<stdio.h>
#include<stdlib.h>
int main()
{
   int a = 5;      //nonzero value (treated as TRUE)
   int b = 20;     //nonzero value (treated as TRUE)
   int c = 0;      //zero value (treated as FALSE)
   int answ1, answ2;
   /*(&&) logical AND
```

```
   - if both are nonzero then the condition become TRUE(1)
   - if both are zero then the condition become FALSE(0)
   - if one is nonzero and one is zero then the condition become FALSE(0)*/
   asnw1 = a && b;
   printf("%d", answ1);
   answ2 = a && c;
   printf("%d", answ2);
   return 0;
}
```

```
1
0
```

The value of `a` and `b` are treated as TRUE since they are nonzero value. Meanwhile `c` is treated as FALSE since it is zero value. The conditional of `a && b` become TRUE (1) because both are nonzero value (TRUE `&&` TRUE). The conditional of `a && c` become FALSE (0) because `a` is nonzero and `c` is zero value (TRUE `&&` FALSE). You can also try this example by using OR operator (`||`).

To know the size of any variable, you can use `sizeof(datatype)` operator which returns the integer value of number of bytes for respective data types. **Data types size are varies depend upon the processor in the CPU that we use**. For instance, if we are using 16 bits processor, 2 bytes (16 bits) of memory will be allocated for `int` data type. Likewise, in 32 bits processors, 4 bytes (32 bits) of memory is allocated for `int` datatype. An example to determine size of each standard data type is given below.

```
/*
This program is to determine the size of each standard data type
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include<stdio.h>
#include<stdlib.h>
int main()
{
   char a;
   short b;
   int c;
   long long d;
   float e;
   double f;
   long double g;
   print("Integer Standard Data Type\n");
   print("Size of char:\t\t%d byte\n",sizeof(a));
```

```
    print("Size of short:\t\t%d bytes\n",sizeof(b));
    print("Size of int:\t\t%d bytes\n",sizeof(c));
    print("Size of long long:\t%d bytes\n",sizeof(d));
    print("Real Number Standard Data Type\n");
    print("Size of float:\t\t%d bytes\n",sizeof(e));
    print("Size of double:\t\t%d bytes\n",sizeof(f));
    print("Size of long double:\t%d bytes\n",sizeof(g));
    return 0;
}
```

```
Integer Standard Data Type
Size of char:           1 byte
Size of short:          2 bytes
Size of int:            4 bytes
Size of long long:      8 bytes
Real Number Standard Data type
Size of float:          4 bytes
Size of double:         8 bytes
Size of long double:    12 bytes
```

Finally, as an additional knowledge, we touch a little bit on `void` data type. The `void` data type is an empty data type which mean has no values and operations. Usually, the `void` type is use in functions. You will learn functions later on in the next chapter. A simple example of using `void` is as show below.

When we use a function return type, `void` means that the function does not return a value. For example:

```
void hello (char *name)
{
  printf("Hello, %s.", name);
}
```

When found in a function heading, `void` means the function does not take any parameters (value 1 is not return for this function). For example:

```
int init (void)
{
  return 1;
}
```

To return the value 1, this function must be initialized as `int` with no `void` in `()` that as follow:

```
int init ()
{
  return 1;
}
```

I have an example regarding the usage of `void`. This program is to calculate the area of rectangle (using `int computeArea()` function) and display the calculated of rectangle area (using `void outputArea()` function). In `int computeArea()` function, the rectangle area value is return to `main` function while in `void outputArea()`, there is no value is return to `main` function. Below is the source codes for this example.

```
/*
This program is to calculate and display the area of a rectangle using
functions
Written by: AFAN, FKP, UMP
Date: September 2016
*/
#include<stdio.h>
#include<stdlib.h>
int computeArea(int width, int height);
void outputArea(int area);
int main()
{
   int rectWidth, rectHeight, rectArea;
   scanf("%d%d", &rectWidth, &rectHeight);
   rectArea = computeArea(rectWidth, rectHeight);
   outputArea(rectArea);
   return 0;
}
int computeArea(int width, int height)
{
   return width * height;
}
void outputArea(int area)
{
   printf("Rectangle area: %d\n", area);
}
```

```
4 3
Rectangle area
```

4. Constants

Constants are data values that cannot be changed during the execution of a program. When you declare a constants, it is a bit like a variable declaration except the value in constants cannot be changed. The declaration of a constants is shows below:

```
const int a = 5;
const float PI = 3.14;
const char letter = 'A';
```

Another way to define constants is by using `#define` pre-processor directive. It is declare after the `#include` and before `int main()` in the body of C program. Example:

```
#include<stdio.h>
#include<stdlib.h>
#define a 5
#define PI 3.14
#define letter 'A'
int main()
{
   return 0;
}
```

An example to differentiate between normal variable (`int a`) and constants (`const int a`) is given below.

```
/*
This program is to differentiate between normal variable and constants
Written by: AFAN, FKP, UMP
Date: September 2016
*/
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a = 3;
    int b = 1;
    const int c = 2;
    int ans1;
```

```
    ans1 = a + b;
    printf("Original a: %d\n",a);
    //printf("Original c: %d\n",c);
    printf("a + b: %d\n",ans1);
    a = ans1 + b;       //int a has changed to sum value of ans1 and b
    //c = ans1 + b;     /*I put comment since this statement cannot be
                        executed (give error) for const – cannot change the
                        value for const int c*/
    printf("Changed a: %d\n",a);
    //printf("Changed c: %d\n",c);
    return 0;
}
```

```
Original a: 3
a + b: 4
Changed a: 5
```

As we see in the above example, the value of `a` can be changed after we have initialized it (`int a = 3`). In contrast, the value of `c` cannot be changed since it has been initialized as constants (`const int c = 2`). The compiler gives an error when we try to change this value.