Pemrograman Dasar

Week 4 : *Introduce of C Programming*

OUTLINES

1. Data Types
2. Variable Declaration
3. Constants
4. Operators
5. Input
6. Escape Sequences
7. Output

1. Data Types

Based on the type, data can be divided into five groups (called the basic data type).

1. Integer
2. Float
3. Double
4. Char
5. Void

Memory Size for data type

Data Type	Number of bits	Range Value
char	8 (1 byte)	-128 s/d 127 (**signed**) | 0 s/d 255 (**unsigned**)
(short) int	16 (2 byte)	-32768 s/d 32767 (**signed**) | 0 s/d 65535 (**unsigned**)
(long) int	32 (4 byte)	-2147483648 s/d 2147483647 (**signed**) | 0 s/d 4,294,967,295 (**unsigned**)
float	32 (4 byte)	1.7E-38 s/d 3.4E+38 (6 decimal places)
double	64 (8 byte)	2.2E-308 s/d 1.7E+308 (15 decimal places)
void	0	-

Data Types [PROGRAM]

ProgDas3-1.c

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

int main() {
  printf("Storage size for char : %d \n", sizeof(char));
  printf("Storage size for short : %d \n", sizeof(short));
  printf("Storage size for int : %d \n", sizeof(int));
  printf("Storage size for float : %d \n", sizeof(float));
  printf("Storage size for double : %d \n", sizeof(double));

   return 0;
}
```

2. Variable Declaration

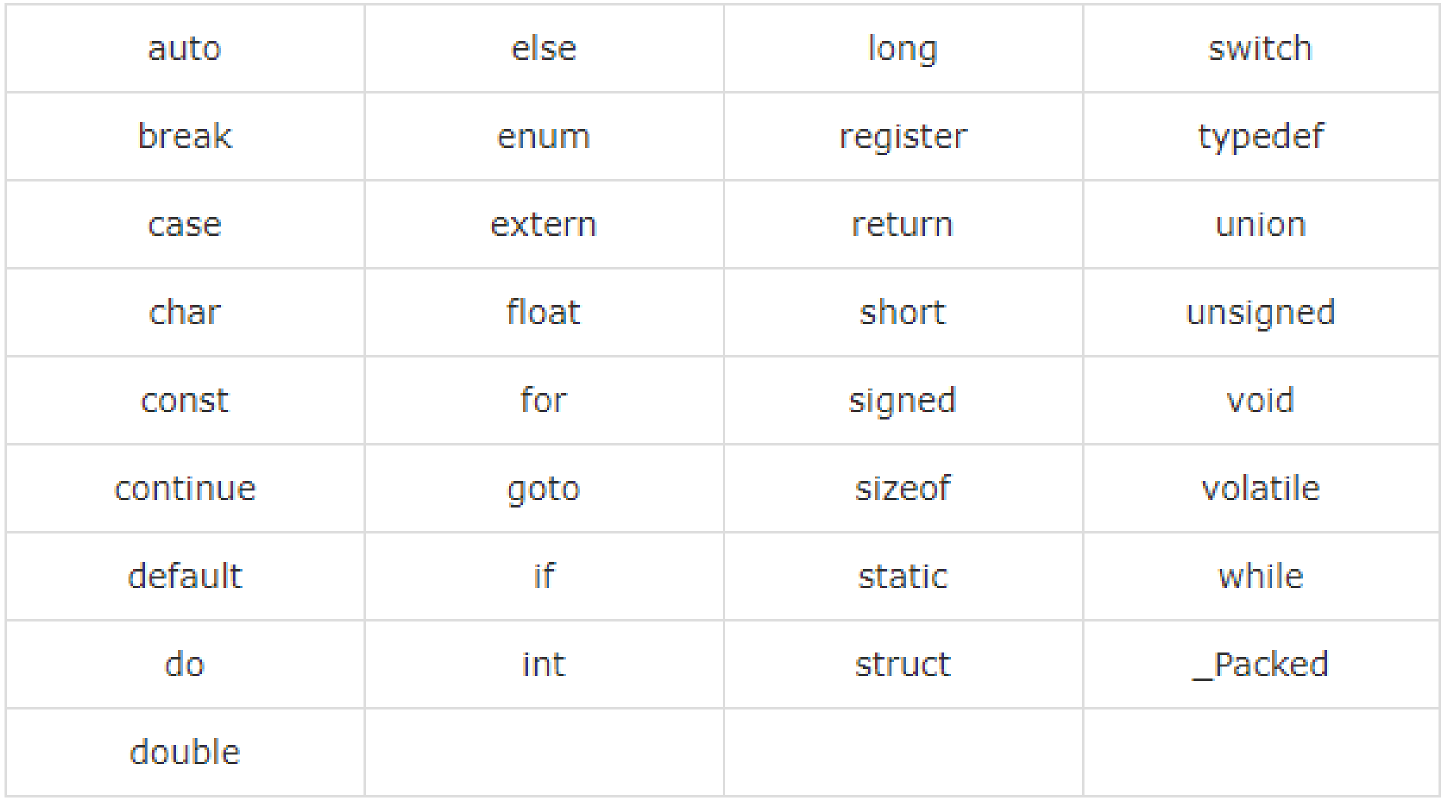
Writing Rules Variable

- The name must begin with a letter (A..Z, a..z), an underscore character (_) or dollar marks ($).

- Variable names may not use names that are classified as reserved words such as `printf`, `int`, `if`, `while` and so on.

Keywords

The following list shows the reserved words in C. **These reserved words may not be used** as constants or variables or any other identifier names.



auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned
const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	_Packed
double			

Variable Declaration

Variables that will be used in the program must be declared first.

The definition of declaration here means ordering memory and determining the type of data that can be stored in it.

```
data_type variable_name;
```

Example:

```
int var_bulat1;
float var_pecahan1, var_pecahan2;
```

Give Variable Values

```
variable_name = value;
```

or

```
data_type variable_name = value;
```

Example:

```
int a;
a = 34;
```

or

```
int b = 889;
```

Variable Declaration [PROGRAM]

ProgDas3-2.c

```
#include <stdio.h>
main()
{
  int jumlah;
  float harga_unit, harga_total;
  jumlah=10;
  harga_unit=17.5;
  harga_total=jumlah*harga_unit;
  printf("Harga total = %f\n",harga_total);
}
```

3. Constants

Constants

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called **literals**.

Constants can be of any of the basic data types like *an integer constant*, *a floating constant*, *a character constant*, *or a string literal*. There are enumeration constants as well.

There are two simple ways in C to define constants:

- Using **#define** preprocessor.
- Using **const** keyword.

Constants [PROGRAM]

ProgDas3-3.c

```
#include <stdio.h>

#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'

int main() {
   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);

   return 0;
}
```

Constants [PROGRAM]

ProgDas3-4.c

```
#include <stdio.h>

int main() {
   const int  LENGTH = 10;
   const int  WIDTH = 5;
   const char NEWLINE = '\n';
   int area;

   area = LENGTH * WIDTH;
   printf("value of area : %d", area);
   printf("%c", NEWLINE);

   return 0;
}
```

4. Arithmetic Operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A holds 10** and variable **B holds 20**

Arithmetic Operators

Operator	Description	Example
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10
*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0
++	Increment operator increases the integer value by one.	A++ = 11
--	Decrement operator decreases the integer value by one.	A-- = 9

Relational Operators

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Logical Operators

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Following table shows all the logical operators supported by C language. Assume variable **A holds 1** and variable **B holds 0**

Bitwise Operators

Assume A = 60 and B = 13 in binary format, they will be as follows:

A = 0011 1100
B = 0000 1101

A&B = 0000 1100
A|B = 0011 1101
A^B = 0011 0001
~A = 1100 0011

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Bitwise Operators (2)

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = -60, i.e,. 1100 0100 in 2's complement form.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

Left-Shift | Right-Shift [PROGRAM]

ProgDas3-5.c

```c
#include<stdio.h>
int main()
{
    // a = 5(00000101), b = 9(00001001)
    char a = 5, b = 9;

    // The result is 00001010
    printf("a = %d\n", a);
    printf("a<<1 = %d\n", a<<1);
    printf("a>>1 = %d\n", a>>1);

    // The result is 00010010
    printf("b = %d\n", b);
    printf("b<<1 = %d\n", b<<1);
    printf("b>>1 = %d\n", b>>1);

    return 0;
}
```

Negative Binary

00000001 = 1-> 11111110= -1

00000010 = 2-> 11111101= -2

00000011 = 3-> 11111100= -3

00000100 = 4-> 11111011= -4

00000101 = 5-> 11111010= -5

00000111 = 6-> 11111000= -6

Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A

Assignment Operators (2)

Operator	Description	Example
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
&	Returns the address of a variable.	&a; returns the actual address of the variable.
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y

5. Ouput

`printf()` used to display data to the screen.

Types of Format:

`%d` or `%i` = represent integer values

`%f` = represent fractional values (float/double)

`%c` = represent characters

`%s` = represent strings

Types of Format [PROGRAM]

ProgDas3-6.c

```
#include <stdio.h>

int main () {
   int ch;

   for( ch = 75 ; ch <= 100; ch++ ) {
      printf("ASCII value = %d, Character = %c\n", ch , ch );
   }

   return(0);
}
```

integer field

To determine the field length from the data display after the “%” sign in the format determinant can be inserted with an integer that states the length of the field.

```
printf(“Abad %4d”, 20);
```

A	b	a	d				2	0

float field

For data in the form of real numbers, the specifications of the field are:

```
printf("Harga : Rp %8.2f\n", 500.0);
```

H	a	r	g	a	:	R	p				5	0	0	.	0	0

String field

printf(“%12s”, “Bahasa C”);

					B	a	h	a	s	a		C

Align right

printf(“%-12s”, “Bahasa C”);

B	a	h	a	s	a		C					

Align left

puts() & putchar()

puts() : display string

```
puts (“Selamat mencoba”);
```

same as

```
printf (“Selamat mencoba\n”);
```

Fungsi putchar() : display character

```
putchar (‘F’);
```

same as

```
printf (“%c”,’F’);
```

gets()

- The `gets()` function is used to enter data of the character type and cannot be used to enter numeric data.
- It must end with pressing the enter key
- The cursor will automatically move the line
- Does not require formatting

gets() [PROGRAM]

ProgDas3-6.2.c

```
#include <stdio.h>
#include <conio.h>
int main()
{
  char nama[20];
  printf("Masukkan nama Anda : ");
  gets(nama);
  printf("Hello, Nama Anda adalah %s", nama);
  getch();

  return 0;
}
```

getchar()

- The `getchar()` function is used to read data of the character type
- It must end with pressing the enter key
- The entered characters are visible on the screen
- Automatic line change

getch() & getche()

- The `getch()` and `getche()` functions are used to read character data.
- The characters entered do not need to end with pressing the enter key.
- Does not give effect to row switching automatically
- If you use the `getch()` function, the entered character will not be displayed on the layer so it is often used to request input in the form of a password.
- Whereas in the `getche()` the entered character will be displayed on the screen.

ProgDas3-6.3.c

```
#include <stdio.h>
#include <conio.h>
int main()
{
  char huruf1, huruf2;
  printf("Masukkan sebuah karakter : ");

  huruf1 = getche(); // karakter yang dimasukkan akan terlihat di layar
  printf("\nKarakter yang Anda masukkan adalah %c\n", huruf1);
  printf("\nMasukkan sebuah karakter lagi : ");

  huruf2 = getch(); // karakter yang dimasukkan tidak terlihat di layar
  printf("\nKarakter yang Anda masukkan adalah : %c", huruf2);
  getch();

  return 0;
}
```

If there are multiple input processes at once, it is recommended to add the **fflush(stdin)** function; after the `scanf()` function. The **fflush (stdin)** function removes the buffer in the I/O device.

6. Escape Sequence

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark

ProgDas3-7.c

```
#include <stdio.h>
int main(void)
{
  printf("Hello\a\n");
  printf("Hello Geeks\b\b\b\bF\n");
  printf("Hello \t GFG\n");
  printf("\v Welcome to GFG\n");
  printf("Hello\\GFG\n");
  printf("\' Hello Geeks\n");
    printf("\" Hello Geeks\n");
  printf("Hello \f GFG\n");
  printf("Hello \r GFG\n");

    return (0);
}
```

7. Input

The `scanf()` function allows you to accept input from standard in, which for us is generally the keyboard.

```
scanf("%d", &b);
scanf ("%d %d",&data1, &data2);
```

The scanf function uses the same placeholders as printf:

`int` uses `%d`

`float` uses `%f`

`char` uses `%c`

`string` uses `%s`

You MUST put "&" in front of the variable used in scanf.

ProgDas3-8.c

```
#include <stdio.h>
int main()
{
  int luas, keliling, panjang_sisi;
  printf("Masukkan panjang sisi bujursangkar : ");
  scanf("%d", &panjang_sisi);

  luas = panjang_sisi * panjang_sisi;
  keliling = panjang_sisi * 4;

  printf("\nData bujursangkar\n");
  printf("Panjang sisi = %6d\n", panjang_sisi);
  printf("Luas = %6d\n", luas);
  printf("Keliling = %6d\n", keliling);

  return 0;
}
```

What happens if C tries to scan character in integer variable

```
scanf("%d", &number);
printf("The result is:%d", number);
```

In this case, `scanf()` directive just fails.

As you didn't initialize it with some defined value (like int number = 0), it'll be just some random **garbage value**.

Check the return of scanf():

```
if (1 == scanf("%d", &number)) {
  printf("The result is:%d", number);
}
else {
  printf("Invalid data entered.  `number` not
changed\n");
}
```

THANKYOU

UNIVERSITAS MUHAMMADIYAH MALANG