**1.Basic C**

1. **C Basic Commands**

#include<stdio.h> : Include standard input output header file from the C library before compiling a C program

Int main() : main function from where C program execution begins.

{ : indicates the beginning of the main function.

/\*\_some\_comments\_\*/

printf(“Hello\_World! “); : print the output on the screen

getch(); : used for any character input from keyboard

return 0; : used to terminate a C program (main function) and it returns 0

} : It is used to indicate the end of the main function

; : each individual statement must be ended with a semicolon

1. **C data types**

|  |  |  |
| --- | --- | --- |
| **Type** | **Storage size** | **Value range** |
| Char | 1 byte | 128 to 127 or 0 to 255 |
| Int | 2 or 4bytes | 2,147,483,648 to 2,147,483,647 |
| Float | 4 bytes | 3.4E-38 to 3.4E+38 |
| double | 8 bytes | 3.4E-4932 to 1.1E+4932 |

**3. C operators**

**1. Arithmetic Operators**

+ : addition or unary plus

- : substraction or unary minus

\* : multiplication

/ : division

% : remainder after division

**2. Assignment Operators**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same as** |
| = | A=b | A=b |
| += | A+=B | A=A+B |
| -= | A-=B | A=A-B |
| \*= | A\*=B | A=A\*B |
| /= | A/=B | A=A/B |
| %= | A%=B | A=A%B |

**3. Relationational Operatiors**

< : less than

<= : less than equal to

> : greater than

>= : greater than or equal to

== : equal to

!= : not equal to

**4. Logical Operators**

&& : Logical AND. True only if all operands are true

|| : Logical OR. True only if either one operand is true

! : Logical NOT. True only if the operand is 0

**5. Bitwise Operators**

& : Bitwise and

| : Bitwise OR

^ : Bitwise exclusive OR

“ : Bitwise complement

<< : Shift left

>> : Shift right

**6.Sizeof operator**

Sizeof(constants,variables,array, etc) : returns the size of data

**2.IF**

**1. If statement**

If(condition)

Instruction;

**2. If-else statement**

If(condition)

{

True block of statements

}

Else

{

False block of statements

}

**3. If…else ladder**

If(condition)

{

True block of statements

}

Else if(condition)

{

True block of statements

}

Else

{

**4.Nested if…else**

-It is possible to include an ‘If…else’ statement inside the body of another ‘if…else’ statement

**5.switch Statement**

- You can do the same thing with the if...else..if ladder.

However, the syntax of the switch statement is much easier to read and write.

switch (expression)

​{

case constant1:

// statements

break;

case constant2:

// statements

break;

.

.

.

default:

// default statements

}

**Loop**

1. **While statement**

[form]

while( boolean\_expression )

{

loop\_statement;

boolean\_expression-alerting statement;

}

ex)

int i = 10;

while(i>0)

{

total = total + i;

i = i – 1;

}

[infinite loop]

while(1)

{

}

1. **For statement**

-form

for (loop\_var = n; loop continuation condition; loop\_var = loop\_var + increment)

{

loop\_statement;

}

<infinite loop>

for( ;; )

{

}

1. While v.s. For

- While

int i = 10; for(i = 10; i > 0; i = i – 1){

while(i>0) total=total+i;

{ }

total = total + i;

i = i – 1;

}

-For

for(i = 10; i > 0; i = i – 1){

1. **Nested loop**

for (loop\_var = n; loop continuation condition; loop\_var = loop\_var + increment)

{

loop\_statement;

for (loop\_var = n; loop continuation condition; loop\_var = loop\_var + increment)

{

loop\_statement;

}

}

1. **Do while loop**

[form]

do

loop\_statement;

while(boolean\_expression);

* It’s executed once and for all.

1. Continue

🡪 The loop is not exited, but control moves to the next iteration of the loop.

Ex)

while(1)

{

printf(“Enter a grade”);

scanf(“%f”, &grade)

if(grade<0||grade>100)

continue;

total=total+grade;

}

1. **Break**

The loop is exited, and control moves to the statement immediately after the loop

Ex)

while(1)

{

printf(“Enter a grade”);

scanf(“%f”, &grade);

if(grade<0||grade>100){

bad=1;

break; }

total=total+grade;

}

If(bad==0)

printf(“total grade = %f”, total);

else

printf(”There was an invalid value”);

**Array**

1. **1-dimensional array**

<form>

data\_type array\_name[index];

index: number of elements in the array

ex)

int ary1[50];

* Array index starts at 0 in C
* Each array elements is a variable
* Each array elements must be initialized before use

1. 2-dimensional array

<form>

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Array\_name[i][j];

🡪 i is rows, j is columns

<initializing>

int value[3][4] = {{8,16,24,32}

{3,15,27,6}

{14,25,2,10}};

1. **Character array**

<ASCII code>

* ‘A’=65 ‘a’=97

<string>

* String is a char array

<form>

Char name[30]=”Jack”;

* End of string vharacter(\0) is appended to the string.

< input & output>

Ex)

char arr;

arr=getchar();

putchar(arr);

<gets & puts>

gets(string\_variable);

🡪reads character input until newline

puts(string\_variable);

🡪print string variable with new line

<string handling library>

🡪#include <string.h>

strlen(str) length of str

strcpy(str1,str2) copies str2 to str1

strcat(str1,str2) appends str2 to str1

strcmp(str1,str2) -int if str1<str2

0 if str1==str2

+int if str1>str2

Strchr(str,char) pointer to the first occurrence of char in str

**Function**

<standard library function>

-void main(){

}

-#include <stdio.h>

-#include <string.h>

-#include <ctype.h>

-#include <stdlib.h>

-#include <math.h>

<programmer-defined functions)

-requirment

Function definition

Function call

Function prototype

-type

With no return value

With return value

With global variables

\* With no return value

Function definition: void function\_name (parameter\_data\_type parameter\_name)

Function call: function\_name (argument1, argument2)

Function prototype: void function\_name(list of argument\_data\_type);

\* With return value

Function definition: return\_data\_type function\_name(parameter\_data\_type parameter\_name)

Function call: function\_name(argument1, argument2)

Function prototype: return\_data\_type function\_name(list of argument\_data\_type);

Return: return(expression)

>returns only one value

\* With global variables

Local variable: Defined within the function that used it.

Global variable: Defined outside the function that used it.