C referens manual

Comments

```
/* Comment */
```

Operators

Assignment Operators		Arithmetic Operators	
Assignment	Something **=** Expression	**Plus**	+
Plus one	Something**++**	**Minus**	-
Minus one	Something**--**	**Multiplyed**	*
It self plus	It_Self **+=** Expression	**Divaded**	/
It self minus	It_Self **-=** Expression	**Modelus**	%
It self multiplyd	It_Self ***=** Expression		
It self divided	It_Self **/=** Expression		

Relational Operators		Logical Operators	
Equal	Comparand_One **==** Comparand_Two	**And**	Expression **&&** Expression
Not Equal	Comparand_One **!=** Comparand_Two	**Or**	Expression **||** Expression
Greater then	Comparand_Big **>** Comparand_Smal	**Not**	**!**Expression
Greater or Equal to	Comparand_Big **>=** Comparand_Smal		
Less then	Comparand_Smal **<** Comparand_Big		
Less or Equal to	Comparand_Smal **<=** Comparand_Big		

Printf operators	
%i, %d	integer
%u	Decimal unsigned int
%f, %F	Double fixt point
%e, %E	Double standard

%g, %G	Double no exponent
%x, %X	Integer in hex
%o	Integer in octal
%s	Character string
%c	character
%p	Void pointer
%n	Print nothing, but write number of characters successfully written so far into an integer pointer parameter.
%%	Print a %

Main funktion

return_operator **main()** { /*stuff*/ **return** return_operator**;** }	return_operator = int or void (standard)

return_operator **main(** int argc, char *argv[] **)**
{ /*stuff*/
return return_operator**;**}
	argc = number of arguments
	argv = argument strings
	
	argv[0] = program name
	argv[1] = first program argument
	argv[n] = blankspace
	
	argv[n] => correct
	*argv[n] => segmentation fault

Type definitions

void	**Void** Name_of_Nothing;
Integer	**Int** Integer_Name;
Char	**Char** Character_Name;
Floating Point	**Float** Floatingpoint_Name**;**
Double precision floating point	**Double** Double_Name**;**

Array	**Array_type** Array_Name **[**Nr_of_Elements**];**	**Deklaration**
	Array_Name**[**Element**]**	**Usage**

Pointer	**Type** ***Pointer_Name;**
	
	type Normal_Type**;**
	
	struct Record_Pointer { **type** ***point** };
		Deklaration

	Record_Pointer record_type**;**	
	Pointer_Name /* returns adress */ *Pointer_Name /* returns type value */ **&**Normal_type /* returns adress of Normal_Type */ record_type**.**point /* returns adress of point*/ record_type**.(*point)** /* returns value of point */ record_type**->**point /* also returns value of point */	**Usage**

Record	Record_Name**.**Type_Name = value_one;
	Record_Name.Type_two_Name = value_two;
	Record_two_Name->Type_Name = value_one;
	Record_two_Name->Type_two_Name = value_two
	Record_Name = { valu_one, value_two };
		Usage

	Struct Record_Type_A_Name Record_Name;
	Record_Type_B_Name Record_Name;
	Record_Type_B_Name *Record_two_Name;
		Deklaration

	Struct Record_Type_A_Name
	{
	type *Type_Name;*
	type_two **Type_two_Name;*
	};
	
	typdef Struct Record_type_B_Name
	{
	type *Type_Name;*
	type_two **Type_two_Name;*
	}Record_type_B_Name;
		Deklaration body

Selections

If- statement	**if** (Boolean_Expression **)**
	{ /* Statements */ }
	else if *(Boolean_Expression)*
	{ /* Statements */ }
	else
	{ /* Statments */ }

Switch/case- statement	**Switch** (Expression)
	{
	case Constant_Value**:**
	/* Statements */
	break;
	case Constant_Value_two**:**
	/* Statements */
	break;
	default:
	/* Statements */

	break;
	}

Repetition

While- loop	**While** (Boolean_Expression) { /* Statements */ }
For- loop	**For** (statement; Boolean_expression; looped_Statement) { /* Statements */ }

Procedures and funktions

Funktion	Funktion_name **(** Parameter_List **);**	**Usage**
	return_type Funktion_Name (Parameter_List **);**	**Deklaration**

	return_type Funktion_Name (Parameter_List)
	{
	/* Statements */
	return Expression_of_Return_Type;
	}
		Deklaration Body

Library structures

Libary	**#include <** Libary_Name **>**	**Use**
	#ifndef Libary_Definition **#define** Libary_Definition /* Deklarationlist */ **#endif**	**Definition** **in file:** Libary_Name.h
	#include < Libary_Name **>** /* funktion bodys */	**Body** **in file:** Libary_Name.c