***Часть 1. Программирование на языке СИ***

**Создание языка Си**

Си был создан в период с 1970 г. по 1972 г. сотрудниками фирмы Bell Labs Денисом Ритчи и Кеном Томпсоном во время написания ОС Unix. Языку Си предшествовали два языка BCPL и B. Автором языка B был Кен Томпсон. В 1973г. Денис Ритчи и Кен Томпсон переписали ядро системы Unix на языке Си и отошли от принятого стандарта использовать язык Ассемблер для написания операционных систем.

Сначала фактическим стандартом языка Си была версия для операционной системы UNIX, которая была впервые описана в 1978г. С 1983 года велась работа по стандартизации языка Си. В 1989г. Американский институт национальных стандартов (American National Standards Institute – ANSI) одобрил стандарт языка Си. Эту версию языка Си обычно называют стандартом С89. В 1999 году появился новый стандарт языка Си – С99.

**Алфавит языка Си**

В основе языка Си лежат неделимые элементарные частицы ­— символы алфавита языка программирования, которые составляют таблицу кодов ASCII.

Из символов строятся элементарные смысловые понятия (слова).

Алфавит Си включает:

1. Строчные и прописные буквы латинского алфавита.
2. Цифры от 0 до 9.
3. Символ ”\_” (подчеркивания).
4. Набор специальных символов

**“ {} , | [] + - % / \ ; ‘ : ? < > = ! & # ~ ^ . \***

1. Прочие символы.

**Типы слов:**

1. Ключевые слова.
2. Идентификаторы (имена переменных).
3. Символы (знаки) операций.
4. Разделители.
5. Литералы (константы) — фиксированные значения, которые программа не может изменять. Способ представления литерала зависит от его типа (например 'a' – символьная константа).

Из слов строятся выражения (предложения), команды.

В Си признаком конца команды является “**;**”.

Из команд строятся программные модули.

В Си нет подпрограмм, есть только функции. Язык Си не является блочно-стуктурированным, так как не позволяет объявлять одну функцию внутри других, но считается структурированным.

Для образования ключевых слов и идентификаторов используются латинские символы, знак подчеркивания и цифры.

Идентификаторы могут начинаться с символа и \_ (знак подчеркивания).

Одинаковые прописные и строчные буквы считаются *разными символами.*

**Типы данных:**

1. Простые.
2. Составные.
3. **Простые данные:**
   1. **Числа**: целые, вещественные.
   2. **Символы** — символ=1 байт=8 бит. Набор символов <=256.
   3. **Адреса** (ссылка, указатель) — целое число без знака.
4. **Составные данные:**
   1. Массивы данных. Одномерный массив — последовательность чисел. Двумерный массив — таблица чисел. Существует понятие массив массивов.
   2. Файлы:

Файл — логическая совокупность данных + физическое место хранения. Файл состоит из символов, чисел, массивов,….

* 1. Структуры, объединения, перечисления:

Структура — совокупность логически связанных данных разных типов. Существуют массивы структур. Все простые типы могут быть использованы для организации составных данных.

**Целочисленный литерал** — служит для записи целочисленных значений и является соответствующей последовательностью цифр (возможно со знаком “-“).

Ц.л., начинающийся с 0, воспринимается как восьмеричное целое.

Ц.л., начинающийся с 0x или 0X, воспринимается как шестнадцатиричное целое. Непосредственно за литералом может располагаться один из (или два) специальных суффикса: U (u) и L (l).

**Вещественный литерал** — служит для отображения вещественных значений в обычной десятичной или научной нотациях. Непосредственно за литералом могут располагаться один из двух специальных суффиксов: F (f) и L(l).

**Символьный литерал** — служит для вывода соответствующего значения ASCII кода и представляет собой последовательность из одной или нескольких литер, заключенных в кавычки. Например, литера Z может быть представлена литералом 'Z', '\132', '\x5A'.

Рассмотрим список литер, которые используются в качестве служебных символов или не имеют графического представления.

|  |  |  |
| --- | --- | --- |
| \0 | \x00 | Нулевая литера |
| \a | \x07 | Сигнал |
| \b | \x08 | Возврат на шаг |
| \f | \x0C | Перевод страницы |
| \n | \x0A | Перевод строки |
| \r | \x0D | Возврат каретки |
| \t | \x09 | Горизонтальная табуляция |
| \v | \x0B | Вертикальная табуляция |
| \\ | \x5C | Обратная косая черта |
| \’ | \x27 | ‘ |
| \” | \x22 | “ |
| \? | \x3F | ? |

Строковый литерал — является последовательностью литер, заключенных в двойные кавычки.

'x' — символ x, “x” — два символа (x и \0), где \0 — терминатор (конец строки).

**Пример простой программы на языке Си**

#include<stdio.h>

int main()

{

int a,b;

a=10;

b=2\*a;

printf(“b=%d\n”,b);

return 0;

}

Программу на языке Си можно условно разделить на две части:

1. декларативную (описание);
2. императивную (исполняемую).

Часть I – директива препроцессора

#include<stdio.h>

#include – включить;

<stdio.h> – объект включения (прототипы функций стандартного ввода, вывода).

Компилятор получает директиву, ищет текст, берет копию и присоединяет ее к тексту программы.

Часть II – исполняемая.

main() – имя главной функции, т.е. выполнение программы начинается с этой функции. У функции есть имя и тело. В теле функции тоже может быть две части – декларативная и императивная.

Декларативная – int a,b;

Императивная – остальная часть тела функции.

Рассмотрим функцию вывода на стандартный терминал:

printf(“I”,II);

I – строка формата (в кавычках).

II – объекты вывода (через запятые).

%d – спецификатор вывода (целочисленное значение).

\n – управляющая последовательность, перевод курсора на начало следующей строки.

**Замечание**: Возможно задание в спецификаторе ширины поля:

%nd %n.mf %ne %ns

**Пример: /\* Элементы программы \*/**

#include<stdio.h>

# include<windows.h> // CharToOem()

int main()

{

int iVar=56;

float fVar=45.567F;

char cStr[]=”Русский текст!!!”;

CharToOem(cStr,cStr);

printf(“iVar=%5d\tfVar=%5.2f\tfVar=%6e\n”,iVar,fVar,fVar);

printf("%25s\n",cStr);

printf("%-25s\n",cStr);

printf("%25.6s\n",cStr);

printf("%-25.6s\n",cStr);

return 0;

}

**/\* Нотация Windows\*/**

Рассмотрим элементы ”венгерской нотации” (венгерский программист фирмы Microsoft Чарльз Симони) — имена переменных начинаются со строчной буквы или букв, описывающих тип данных переменных (префикс).

|  |  |
| --- | --- |
| Префикс | Соответствующий тип данных |
| **b** | BOOL (целое число) |
| **c** | Character (символ) |
| **i** | Integer (целое число) |
| **p** | Указатель |
| **s** | Строка |
| **v** | Void |
| **w** | Word |

При работе с компилятором Visual C++ существует проблема вывода на экран букв русского алфавита. Для решения этой проблемы обычно используют функцию CharToOem(), прототип которой описан в <windows.h>

**Пример**: /\*Использование русского текста при выводе\*/

#include<stdio.h>

#include<windows.h>

int main()

{

char str[20];

CharToOem("Русский текст!!!",str);

printf("%s\n",str);

char str1[20]="Второй вариант !!!";

CharToOem(str1,str);

printf("%s\n",str);

return 0;

}

**Типы данных. Операции над данными**

**Стандартные типы данных:** В языке Си существует пять элементарных типов данных. Это

1. символ – **char**;
2. целое число – **int**;
3. число с плавающей точкой – **float**;
4. число с плавающей точкой удвоенной точности – **double**;
5. переменная, не имеющая значений – **void**.

**Замечание**: Стандарт С99 добавляет еще три основных типа:

1. \_Bool;
2. \_Complex;
3. \_Imaginary.

Все другие типы данных в Си создаются на основе элементарных типов.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название типа | Нижняя граница | Верхняя граница | Точность | Байт |
| bool | False | True | No | 1 |
| char | -128 | 127 | No | 1 |
| short | -32768 | 32767 | No | 2 |
| int | -2 147 483 648 | 2 147 483 647 | No | 4 |
| long | -2 147 483 648 | 2 147 483 647 | No | 4 |
| float | 3.4\*10-38 | 3.4\*1038 | 7 | 4 |
| double | 1.7\*10-308 | 1.7\*10308 | 15 | 8 |

**Беззнаковые целые типы:**

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Нижняя граница | Верхняя граница | Размер (байт) |
| unsigned char | 0 | 255 | 1 |
| unsigned short | 0 | 65535 | 2 |
| unsigned int | 0 | 4 294 967 295 | 4 |
| unsigned long | 0 | 4 294 967 295 | 4 |

**Неявное преобразование типов:**

|  |  |
| --- | --- |
| Тип данных | Старшинство |
| long double | Высший |
| double |  |
| float |  |
| long |  |
| int |  |
| short |  |
| char | Низший |

**Спецификатор типов:**

|  |  |
| --- | --- |
| Спецификатор | Форма вывода |
| %d | десятичное число со знаком |
| %i | десятичное число со знаком |
| %u | беззнаковое целое десятичное число |
| %f | вещественное число с плавающей точкой |
| %e | экспоненциальная форма |
| %c | cимвол. |
| %s | cтрока символов |
| %o | восьмеричное число |
| %x | шестнадцатиричное число |
| %p | указатель |
| %ld | long int |
| %lf | long double |

**Пример: /\* Программа определения номера кода и размеров типов \*/**

#include<stdio.h>

int main()

{

char ch;

unsigned char ch1;

int iVar1=90;

int iVar2=045;

int iVar3=0x45;

printf(" iVar1=%d, iVar2=%d, iVar3=%d\n", iVar1,iVar2,iVar3);

printf("Input symbol \n");

scanf("%c",&ch);

ch1=ch;

printf("Kod %c = %d %d.\n",ch,ch,ch1);

printf("char = %d\n",sizeof(char));

printf("short = %d\n",sizeof(short));

printf("int = %d\n",sizeof(int));

printf("long = %d\n",sizeof(long));

printf("float = %d\n",sizeof(float));

printf("double = %d\n",sizeof(double));

printf("long double = %d\n",sizeof(long double));

printf("dec=%d oct=%o hex=%x char=%c\n",iVar1,iVar1,iVar1,iVar1);

printf("z= %c 132=%c x5a=%c\n",'Z','\132','x5a');

printf("\'Z\'= %c \'\\132\'=%c \'x5a\'=%c\n",'Z','\132','\x5a');

// dec=90 oct=132 hex=5a char=Z

return 0;

}

Если на запрос программы вести русскую букву "я", то код этой буквы будет для типа char – -17, а для типа unsigned char – 239. Для объяснения этого результата необходимо рассмотреть представление числа в дополнительном обратном коде.

**Классификация операций**

Классификация операций идет по числу операндов.

**Группа унарных операций**

|  |  |
| --- | --- |
| Знак операции | Операция |
| - | Арифметическое отрицание |
| ~ | Поразрядное логическое отрицание (дополнение) |
| ! | Логическое отрицание |
| \* | Разадресация |
| & | Адрес |
| + | Унарный плюс |
| Sizeof | Размер |
| ++ -- | Операции единичного приращения |

Операции разадресации и адреса используются для работы с переменными типа указатель.

Операция ”адрес” (”&”) дает адрес своего операнда.

Операция разадресации (”\*”) осуществляет косвенный доступ к адресуемой величине через указатель. Операнд должен быть указателем. Результатом операции является величина переменной, на которую указывает операнд.

Операция \* — операция обращения по адресу (не знаем адрес, но обращаемся к нему).

Пример:

int x, y =10, \*address;

address=&x;

\*address=y;

printf(“y=%d x=%d”,y,x);

Операции единичного приращения:

++ – приращение положительно (инкремент).

-- – приращение отрицательно (декремент).

Операции увеличения (++) и уменьшения (--) являются унарными операциями присваивания. Они соответственно увеличивают или уменьшают значение операнда на единицу.

Операнд должен быть:

1. Целого типа.
2. Плавающего типа.
3. Адрес (указатель).

Для целого и плавающего типа увеличение или уменьшение происходит на "1".Операнд адресного типа увеличивается или уменьшается на размер объекта, который он адресует.

Существует две формы:

1. Префиксная — сначала приращение, потом действие.
2. Постфиксная — сначала действие, потом приращение.

**Пример:**

#include<stdio.h>

#include<windows.h>

int main()

{

int a,b,c,d,r,s;

float g,f,h;

char \*pChar;

int \*pInt;

a=b=1;

s=a+b++;

printf("a=%d b=%d s=%d\n",a,b,s);

c=d=3;

r=c+ ++d;

printf("c=%d d=%d r=%d\n",c,d,r);

c=d=3;

r=c+++d;

printf("c=%d d=%d r=%d\n",c,d,r);

c=d=3;

r=c+++d++;

printf("c=%d d=%d r=%d\n",c,d,r);

c=d=3;

r= --c+d++;

printf("c=%d d=%d r=%d\n",c,d,r);

g=3.5;

f=4.2;

h=++g+--f;

printf("g=%f f=%f h=%f\n",g,f,h); // c=4.5 f=3.2 h=7.7

char cChar='a';

pChar=&cChar;

printf("cChar=%c, pChar=%p cChar=%c pChar=%p\n",cChar,&cChar,\*pChar,pChar);

cChar++;

pChar=&cChar;

printf("cChar=%c cChar=%p \n",cChar,&cChar);

// Поразрядное логическое отрицание (дополнение)

unsigned char cVar1=140,cVar2;

cVar2=~cVar1;

printf("cVar1=%d\tcVar2=%d\n",cVar1,cVar2); // 140 115

printf("cVar1=%c\t\tcVar2=%c\n",cVar1,cVar2); // M s

return 0;

}

**Группа бинарных операций**

|  |  |
| --- | --- |
| Знак операции | Операция |
| \* | Умножение |
| / | Деление |
| % | Остаток от деления |
| + | Сложение |
| - | Вычитание |
| << | Сдвиг влево |
| >> | Сдвиг вправо |
| < | Меньше |
| > | Больше |
| <= | Меньше или равно |
| >= | Больше или равно |
| = = (=) | Логическое равно |
| != | Не равно |
| & | Поразрядное логическое И |
| | | Поразрядное логическое ИЛИ |
| ^ | Поразрядное исключающее ИЛИ |
| && | Логическое И |
| || | Логическое ИЛИ |
| , | Последовательное вычисление |