**Использование указателей в качестве аргументов функций**

Определяя некоторую функцию, мы описываем и ее параметры, которые называются "формальными" аргументами. Фактически – это новые объекты программы, и при вызове функции для них выделяется отдельная область памяти. Такими объектами могут быть константы, переменные, массивы.

При вызове некоторой функции в качестве списка параметров выступают фактические аргументы, значение которых вычисляется и присваивается формальным аргументам. Фактический аргумент может быть константой, переменной и более сложным выражением. Независимо от типа фактического аргумента, он вначале вычисляется, а затем его величина передается функции. Такая передача аргументов (фактических параметров) называется передачей по значению.

Определим функцию **spaсe(),** которая оставляет при печати задаваемое количество пробелов:

|  |
| --- |
| void space(int number){ int count; for(count=1;count<=number;count++) putchar(' ');} Приведем несколько примеров возможных вызовов этой функции: space(25); // функция использует в качестве аргумента константу  int spaces;spaces=25;space(spaces); // аргументом является переменная //аргументом является выражение space((37-strlen("Examples of calls"))/2); |

где **strlen()** – стандартная функция определения длины строки. Передача значения из вызванной функции в вызывающую происходит с помощью оператора возврата.

Возврат значения в вызывающую функцию осуществляется оператором **return;** или **return(<выражение>);.** Например, функция, которая вычисляет абсолютную величину числа, может быть описана так:

|  |
| --- |
| int abs(int x){ if(x<0) x =- x; return(x);} |

Возвращаемое значение можно присвоить переменной или использовать как часть некоторого выражения. Например, **y** **= abs(x);** или **y** **= 2\*abs(x)+3;**

***Задание*.** Перепишите функцию abs() так, чтобы ее тело состояло из одного оператора.

Значение выражения в **return(<выражение>)** всегда преобразуется к типу функции, и только после этого следует возврат.

Оператор **return** завершает выполнение вызываемой функции и передает управление вызывающей функции (оператору, следующему за вызовом). Это происходит даже в том случае, если оператор **return** является не последним оператором тела функции.

Для завершения выполнения функции можно использовать просто оператор **return;.** Поскольку у данного оператора отсутствует выражение в скобках, никакое значение при этом не передается вызывающей функции.

Управление возвращается в вызывающую функцию и в случае выхода по достижению конца функции. Значение при этом не передается.

***Описание типа возвращаемого значения.*** Тип функции определяется типом возвращаемого ею значения, а не типом ее аргументов. Функция, которая не возвращает значения, должна быть описана как имеющая тип **void.** Например,

**void ErrorMsg(char \*s)**/\* функция не возвращает значения \*/
**{**

**printf("==>%s\n",s);**
**}**

Если указание типа отсутствует, то по умолчанию предполагается, что функция имеет тип int. В языке С++ не поддерживается тип функции int по умолчанию.

**9.4.2. Передача аргументов по умолчанию**

При объявлении функции в С++ одному или нескольким аргументам может быть назначено значение по умолчанию. Это означает, что при вызове, аргументы имеющие значения по умолчанию, можно опускать.

Например:

**void func(double m, char ch = ’\*’, int i = 2);**

Два параметра – **ch** и **i** имеют значения по умолчанию, которые будут использованы в случае вызова функции **func** с неполным списком параметров:
**func(6.78, ’d’, 5);**
**func(6.78, ’d’);**
**func(6.78);**

Вызов

**func(6.78,***22***);**

будет неправильным, т.к. второй параметр должен иметь тип **char**.

При объявлении функции имена формальных параметров указывать необязательно, поэтому функцию можно объявит и так:

**void func(double, char = ’\*’, int = 2);**

Если вы не указываете имена параметров, то надо быть осторожным при инициализации параметров указателей и ссылок. Например, в объявлении

**void f(double, char \* = ”Hello”, int = 2);**

если “забыть” пробел перед знаком =, то можно получить составную операцию умножения: \*=;

Параметры со значениями по умолчанию должны быть последними в списке параметров. Кроме того, к моменту вызова функции с неполным списком значение по умолчанию уже должно быть определено и видимо, потому что значения по умолчанию могут быть заданы **либо в** **объявлении**, **либо в определении**, но лишь в **одном**месте. Хороший стиль программирования требует задавать параметры по умолчанию в прототипе функции.

**9.4.3. Указатели в качестве аргументов**

С помощью оператора **return** в вызывающую функцию можно передать только одну величину. А как вернуть две или более величины? Для этого нужно воспользоваться переменными типа указатель и передавать не данные, а их адреса или ссылки. Передача параметров по ссылке рассматривается в следующем разделе.

Пусть, например, мы хотим с помощью некоторой функции **swap()** выполнить обмен значений у переменных **х** и **y.** Ясно, что мы должны передать этой функции в качестве аргументов две величины и получить в качестве результата тоже две величины. Если мы определим функцию **swap()** таким образом:

|  |
| --- |
| void swap(int u, int v){ int temp; temp=u; u=v; v=temp;} |

и обратимся к ней с аргументами **х,у: swap(x, y);** то произойдет передача параметров по значению, т.е. переменная **u** получит значение из переменной **х,** а **v** – из переменной **у**, функция выполнит обмен значений над переменными **u** и **v**, которые существуют только в этой функции и вне её недоступны. Следовательно, в результате вызова **swap(x, y)** переменные **х** и **у** не изменяют своих значений.

Рассмотрим, как следует воспользоваться указателями для возврата из функций нескольких значений. Определим нашу функцию следующим образом:

|  |
| --- |
| void swap(int \*u, int \*v) // u и v являются указателями { int temp; temp = \*u; // temp получает значение, на которое указывает u \*/ \*u = \*v; \*v = temp;}//Тогда вызов этой функции будет оформлен так:int x,y;swap(&x, &y); // передача адресов |

Посмотрим, как работает теперь эта функция. В этом случае вместо передачи значений **x** и **y** мы передаем их адреса. Это значит, что формальные аргументы **u** и **v** при обращении будут заменены адресами, и, следовательно, они должны быть описаны как указатели. Поскольку **x** и **y** – целого типа, **u** и **v** являются указателями на переменную целого типа, и они описаны так: **int \*u,\*v;**

Далее рассмотрим оператор **temp = \*u;**
Переменная **temp** целого типа и используется для временного хранения обмениваемых величин.

Значение переменной **u** – это **&x** (следует из вызова функции), поэтому переменная и является указателем на x. Значит, операция \* и дает значение **х**. (Если бы мы написали **temp = u**;, то произошло бы запоминание адреса переменной **x**, а не ее значения.) Точно так же, желая присвоить переменной y значение переменной **x**, мы пользуемся оператором **\*u = \*v**; который соответствует оператору **x = y** ;

Итак, путем передачи функции адресов переменных **x** и **y** мы предоставили ей возможность доступа к ним. Используя указатели и операцию \*(разадресации), функция смогла извлечь величину, помещенную в соответствующие ячейки памяти, и поменять их местами.

*Обратите внимание!* При вызове функции информация о переменной может передаваться функции в двух видах. Если мы используем форму обращения **function1(x);** происходит передача значений переменной **x**. Если же мы используем форму обращения **function2(&x);** происходит передача адреса переменной **x**. Первая форма обращения требует, чтобы определение функции включало в себя формальный аргумент того же типа, что и **x**: **function1(int** **num);**

Вторая форма обращения требует, чтобы определение функции включало в себя формальный аргумент, являющийся указателем на объект соответствующего типа: **function2(int \*ptr).**

Пользуйтесь первой формой, если входное значение необходимо функции для некоторых вычислений или действий, значение будет копироваться в функцию и вызывающей функции не изменяется. Пользуйтесь второй формой, если функция должна будет изменить значение переменных в вызывающей программе, т.е. по месту их нахождения, сами значения в функцию копироваться не будут.

Таким образом, **передача указателей** на данные вместо самих данных может быть использована в следующих случаях:

* для передачи в функцию больших объемов (структур) данных, чтобы избежать копирования аргументов в стек;
* для передачи в функцию аргументов, которые будут использованы не для передачи данных, а для возврата значений из функции, что позволяет вернуть из функции более одного значения;
* для передачи в функцию адресов массивов

Дополнительная информация по ссылке: http://proguroki.ru/urok9/urok9\_4/urok9\_4\_2/urok-9-4-argumenty-fuctsii