Лабораторна робота № 10

ВКАЗІВНИКИ.

Мета роботи:

Вивчити поняття і застосування вказівників.

Обладнання та програмне забезпечення:

- IBM сумісна персональна обчислювальна машина;
- онлайн компілятор мови програмування Сі, доступний за посиланням https://repl.it/languages/c

Завдання до роботи:

Написати програму на мові програмування Сі, котра використовує вказівники:

- для масиву повідомлень;
- для масиву функцій.

Теоретичні відомості

Вказівник – це змінна, значенням якої є адреса комірки пам’яті. Тобто вказівник посилається на блок даних з області пам’яті. Вказівник може посилатися на змінну або функцію. Для цього потрібно знати адресу змінної або функції. Щоб дізнатися адресу конкретної змінної в Сі існує унарна операція отримання адреси &. Така операція отримує адресу оголошених змінних, для того, щоб її присвоїти вказівнику.

Вказівники використовуються для передачі за посиланням даних, що набагато прискорює процес обробки цих даних (в тому випадку, якщо об’єм даних великий), так як їх не треба копіювати, як при передачі за значенням, тобто, використовуючи ім’я змінної. В основному вказівники використовуються для організації динамічного розподілу пам’яті, наприклад при оголошенні масиву, не треба буде його обмежувати в розмірі. Адже програміст заздалегідь не може знати, якого розміру потрібен масив тому чи іншому користувачеві, в такому випадку використовується динамічне виділення пам’яті під масив. Будь-який вказівник необхідно оголосити перед використанням, як і будь-яку змінну.

Принцип оголошення вказівників такий же, як і принцип оголошення змінних. Відмінність полягає лише в тому, що перед ім’ям ставиться символ зірочки *. Візуально вказівники відрізняються від змінних тільки одним символом. При оголошенні вказівників компілятор виділяє декілька байтів пам’яті, в залежності від типу даних, що відводяться для зберігання деякої інформації в пам’яті. Щоб отримати значення, записане в деякій області, на яке посилається вказівник потрібно скористатися операцією розіменування вказівника *. Необхідно поставити зірочку перед ім’ям і отримаємо доступ до значення вказівника. Наведемо програму, яка буде використовувати вказівники.

```c
#include <stdio.h>
int main (void) {
  int x, *p1, *p2;
  p1 = &x;
  p2 = p1;
  // виводить адресу, що зберігається в p1 і p2. Вони однакові
  printf ("p1=%p; p2=%p", p1, p2);
  return 0;
}
```

Вказівники і масиви

```c
#include <stdio.h>
int main(void) {
  char s1[] = {"First test string"};
  char s2[] = {"Second test string"};
  char *p;
  printf("Use an index: ");
  for(int i=0; s1[i]; ++i)  printf("%c",s1[i]);
  printf("\nUse a pointer: ");
  p = s2;
  while(*p)  printf("%c", *p++);
  return 0;
}
```

Вказівники на вказівники

Вказівники можуть посилатися на інші вказівники. При цьому в комірках пам'яті, на які будуть посилатися перші вказівники, будуть міститися не значення, а адреси інших вказівників. Число символів * при оголошенні вказівника показує порядок вказівника. Щоб отримати доступ до значення, на яке посилається вказівник його необхідно розіменовувати відповідну кількість разів.

```c
#include <stdio.h>
int main(void) {
  int X=10, *p, **q;
  p = &X;
  q = &p;       // вивід значення X через вказівник на вказівник
  printf("X value through pointer on pointer = %d", **q);
  return 0;
}
```

Масиви вказівників – друк повідомлення про помилку

```c
void serror(int num) {
  static char *err[] = {
    "Cannot Open File\n",
    "Read Error\n",
    "Write Error\n",
    "Media Failure\n"
  };
  printf("%s", err[num]);
}
```

Вказівники на функції

Хоча функція - це не змінна, вона все одно має фізичне місце розташування в пам'яті, яке може бути присвоєно вказівником. Адреса функції є вхідною точкою функції. Тому вказівник на функцію може використовуватися для виклику функції.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
void enter(void), del(void), review(void), quit(void);
int menu(void);

void (*options []) (void) = {
      enter,
      del,
      review,
      quit
      };

int main(void) {
      int i;
      i = menu();
      (*options[i])();
      return 0;
}

int menu(void) {
      char ch;
      do {
         printf("1. Enter\n");
         printf("2. Delete\n");
         printf("3. Review\n");
         printf("4. Quit\n");
         printf("Select a number: ");
         ch = getchar();
         printf ("\n");
      } while(!strchr("1234", ch));
      return ch-49; // перетворення до цілочисельного еквіваленту
}                   // зміщення по таблиці ASCII

void enter(void) {
      printf("In enter.");
}

void del(void) {
      printf("In del.");
}

void review(void) {
      printf("In review.");
}

void quit (void) {
      printf("In quit.");
      exit(0);
}
```

ПОРЯДОК ВИКОНАННЯ РОБОТИ:

1. Опрацювати і засвоїти матеріал наведений в теоретичних відомостях.
2. Написати функцію для обчислення довжини стрічки, не використовуючи жодних бібліотек, окрім stdio.h.
3. Написати функцію з прототипом: int* toPoint(int x, int y). Функція повинна виділити вільну пам’ять на два цілочисельні значення (використайте malloc з stdlib.h), записати в них координати точки (*x*, *y*) і повернути вказівник на проініціалізовану точку. В main() створити, вивести точку та вказівник на неї, а потім звільнити виділену пам’ять (використайте free з stdlib.h).
4. Написати програму, що виконує прості арифметичні операції. Створити чотири функції для додавання, віднімання, множення та ділення дійсних чисел. Зберегти вказівники на ці функції у масив вказівників. Зчитати з консолі ввід типу 1.28+3.14, розбити на операнди і оператор, викликати відповідну функцію з масиву вказівників на операції, вивести результат.
5. В звіті навести копії екранів та написаний код.
6. Зробити висновки.

.